Référence de l'API JavaScript / EcmaScript 2017

Table des matières

1. Introduction

Ce manuel est votre référence pour l'utilisation de la librairie JavaScript / EcmaScript 2017 de Yoctopuce pour interfacer vos senseurs et contrôleurs USB.

Vous pourrez utiliser cette librairie aussi bien depuis vos applications JavaScript destinées à fonctionner dans un navigateur Internet, que pour vos applications Node.js.

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 / EcmaScript

EcmaScript est le nom officiel de la version standardisée du langage de programmation communément appelé JavaScript. Cette librairie de programmation Yoctopuce utilise les nouvelles fonctionnalités introduites dans la version EcmaScript 2017. La librairie porte ainsi le nom *Librairie pour JavaScript / EcmaScript 2017*, afin de la différentier de la précédente *Librairie pour JavaScript* qu'elle remplace.

Cette librairie permet d'accéder aux modules Yoctopuce depuis tous les environnements JavaScript modernes. Elle fonctionne aussi bien depuis un navigateur internet que dans un environnement Node.js. La librairie détecte automatiquement à l'initialisation si le contexte d'utilisation est un browser ou une machine virtuelle Node.js, et utilise les librairies systèmes les plus appropriées en conséquence.

Les communications asynchrones avec les modules sont gérées dans toute la librairie à l'aide d'objets *Promise*, en utilisant la nouvelle syntaxe EcmaScript 2017 `async` / `await` non bloquante pour la gestion des entrées/sorties asynchrones (voir ci-dessous). Cette syntaxe est désormais disponible sans autres dans la plupart des moteurs JavaScript: il n'est plus nécessaire de transpiler le code avec Babel ou `jspm`. Voici la version minimum requise de vos moteurs JavaScript préférés, tous disponibles au téléchargement:

- Node.js v7.6 and later
- Firefox 52
- Opera 42 (incl. Android version)
- Chrome 55 (incl. Android version)
- Safari 10.1 (incl. iOS version)
- Android WebView 55
- Google V8 Javascript engine v5.5

Si vous avez besoin de la compatibilité avec des anciennes versions, vous pouvez toujours utiliser Babel pour transpiler votre code et la libriairie vers un standard antérieur de JavaScript, comme décrit un peu plus bas.

Nous ne recommendons plus l'utilisation de `jspm 0.17` puisque cet outil est toujours en version Beta après 18 mois, et que solliciter l'utilisation d'un outil supplémentaire pour utiliser notre librairie ne se justifie plus dès lors que `async` / `await` sont standardisés.

2.1. Fonctions bloquantes et fonctions asynchrones en JavaScript

JavaScript a été conçu pour éviter toute situation de *concurrence* durant l'exécution. Il n'y a jamais qu'un seul *thread* en JavaScript. Cela signifie que si un programme effectue une attente active durant une communication réseau, par exemple pour lire un capteur, le programme entier se trouve bloqué. Dans un navigateur, cela peut se traduire par un blocage complet de l'interface utilisateur. C'est pourquoi l'utilisation de fonctions d'entrée/sortie bloquantes en JavaScript est sévèrement découragée de nos jours, et les API bloquantes se font toutes déclarer *deprecated*.

Plutôt que d'utiliser des *threads* parallèles, JavaScript utilise les opérations asynchrones pour gérer les attentes dans les entrées/sorties: lorsqu'une fonction potentiellement bloquante doit être appelée, l'opération est uniquement déclenchée mais le flot d'exécution est immédiatement terminé. La moteur JavaScript est alors libre pour exécuter d'autres tâches, comme la gestion de l'interface utilisateur par exemple. Lorsque l'opération bloquante se termine finalement, le système relance le code en appelant une fonction de callback, en passant en paramètre le résultat de l'opération, pour permettre de continuer la tâche originale.

Lorsqu'on les utilises avec des simples fonctions de callback, comme c'est fait quasi systématiquement dans les librairies Node.js, les opérations asynchrones ont la fâcheuse tendance de rentre le code illisible puisqu'elles découpent systématiquement le flot du code en petites fonctions de callback déconnectées les unes des autres. Heureusement, de nouvelles idées sont apparues récemment pour améliorer la situation. En particulier, l'utilisation d'objets *Promise* pour travailler avec les opérations asynchrones aide beaucoup. N'importe quelle fonction qui effectue une opération potentiellement longue peut retourner une *promesse* de se terminer, et cet objet *Promise* peut être utilisé par l'appelant pour chaîner d'autres opérations en un flot d'exécution. La classe *Promise* fait partie du standard EcmaScript 2015.

Les objets *Promise* sont utiles, mais ce qui les rend vraiment pratique est la nouvelle syntaxe `async` / `await` pour la gestion des appels asynchrones:

- une fonction déclarée *async* encapsule automatiquement son résultat dans une promesse
- dans une fonction *async*, tout appel préfixé par *await* a pour effet de chaîner automatiquement la promesses retournées par la fonction appelée à une promesse de continue l'exécution de l'appelant
- tout exception durant l'exécution d'une fonction *async* déclenche le flot de traitrment d'erreur de la promesse.

En clair, *async* et *await* permettent d'écrire du code EcmaScript avec tous les avantages des entrées/sorties asynchrones, mais sans interrompre le flot d'écriture du code. Cela revient quasiment à une exécution multi-tâche, mais en garantissant que le passage de contrôle d'une tâche à l'autre ne se produira que là où le mot-clé *await* apparaît.

Nous avons donc décidé d'écrire cette nouvelle librairie EcmaScript en utilisant les objets *Promise* et des fonctions *async*, pour vous permettre d'utiliser la notation *await* si pratique. Et pour ne pas devoir vous poser la question pour chaque méthode de savoir si elle est asynchrone ou pas, la convention est la suivante: **toutes les méthodes publiques** de la librairie EcmaScript **sont *async***, c'est-à-dire qu'elles retournent un objet Promise, **sauf**:

- `GetTickCount()`, parce que mesurer le temps de manière asynchrone n'a pas beaucoup de sens...
- `FindModule()`, `FirstModule()`, `nextModule()`,... parce que la détection et l'énumération des modules est faite en tâche de fond sur des structures internes qui sont gérées de manière transparente, et qu'il n'est donc pas nécessaire de faire des opérations bloquantes durant le simple parcours de ces listes de modules.

2.2. Utiliser la librairie Yoctopuce pour JavaScript / EcmaScript 2017

JavaScript fait partie de ces langages qui ne vous permettront pas d'accéder directement aux couches matérielles de votre ordinateur. C'est pourquoi si vous désirez travailler avec des modules USB branchés par USB, vous devrez faire tourner la passerelle de Yoctopuce appelée VirtualHub sur la machine à laquelle sont branchés les modules.

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

- La librairie de programmation pour Javascript / EcmaScript 2017[1]
- Le programme VirtualHub[2] 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 et lancez le programme VirtualHub. Vous n'avez pas besoin d'installer de driver.

Utiliser la librairie Yoctopuce officielle pour node.js

Commencez par installer sur votre machine de développement la version actuelle de Node.js (7.6 ou plus récente), C'est très simple. Vous pouvez l'obtenir sur le site officiel: http://nodejs.org. Assurez vous de l'installer entièrement, y compris npm, et de l'ajouter à votre system path.

Vous pouvez ensuite prendre l'exemple de votre choix dans le répertoire `example_nodejs` (par exemple `example_nodejs/Doc-Inventory`). Allez dans ce répertoire. Vous y trouverez un fichier décrivant l'application (`package.json`) et le code source de l'application (`demo.js`). Pour charger automatiquement et configurer les librairies nécessaires à l'exemple, tapez simplement:

```
npm install
```

Une fois que c'est fait, vous pouvez directement lancer le code de l'application:

```
node demo.js
```

Utiliser une copie locale de la librairie Yoctopuce avec node.js

Si pour une raison ou une autre vous devez faire des modifications au code de la librairie, vous pouvez facilement configurer votre projet pour utiliser le code source de la librairie qui se trouve dans le répertoire `lib/` plutôt que le package npm officiel. Pour cela, lancez simplement la commande suivante dans le répertoire de votre projet:

```
npm link ../../lib
```

Utiliser la librairie Yoctopuce dans un navigateur (HTML)

Pour les exemples HTML, c'est encore plus simple: il n'y a rien à installer. Chaque exemple est un simple fichier HTML que vous pouvez ouvrir directement avec un navigateur pour l'essayer. L'inclusion de la librairie Yoctopuce ne demande rien de plus qu'un simple tag HTML `<script>`.

Utiliser la librairie Yoctopuce avec des anciennes version de JavaScript

Si vous avez besoin d'utiliser cette librairie avec des moteurs JavaScript plus anciens, vous pouvez utiliser Babel[3] pour transpiler votre code et la librairie dans une version antérieure du langage. Pour installer Babel avec les réglages usuels, tapez:

[1] www.yoctopuce.com/FR/libraries.php
[2] www.yoctopuce.com/FR/virtualhub.php
[3] http://babeljs.io

```
npm instal -g babel-cli
npm instal babel-preset-env
```

Normalement vous demanderez à Babel de poser les fichiers transpilés dans un autre répertoire, nommé `comopat` par exemple. Pour ce faire, utilisez par exemple les commandes suivantes:

```
babel --presets env demo.js --out-dir compat/
babel --presets env ../../lib --out-dir compat/
```

Bien que ces outils de transpilation soient basés sur node.js, ils fonctionnent en réalité pour traduire n'importe quel type de fichier JavaScript, y compris du code destiné à fonctionner dans un navigateur. La seule chose qui ne peut pas être faite aussi facilement est la transpilation de sciptes codés en dure à l'intérieur même d'une page HTML. Il vous faudra donc sortir ce code dans un fichier `.js` externe si il utiliser la syntaxe EcmaScript 2017, afin de le transpiler séparément avec Babel.

Babel dipose de nombreuses fonctionnalités intéressantes, comme un mode de surveillance qui traduite automatiquement au vol vos fichiers dès qu'il détecte qu'un fichier source a changé. Consultez les détails dans la documentation de Babel.

Compatibilité avec l'ancienne librairie JavaScript

Cette nouvelle librairie n'est pas compatible avec l'ancienne librairie JavaScript, car il n'existe pas de possibilité d'implémenter l'ancienne API bloquante sur la base d'une API asynchrone. Toutefois, les noms des méthodes sont les mêmes, et l'ancien code source synchrone peut facilement être rendu asynchrone simplement en ajoutant le mot-clé `await` devant les appels de méthode. Remplacez par exemple:

```
beaconState = module.get_beacon();
```

par

```
beaconState = await module.get_beacon();
```

Mis à part quelques exceptions, la plupart des méthodes redondantes `XXX_async` ont été supprimées, car elles auraient introduit de la confusion sur la manière correcte de gérer les appels asynchrones. Si toutefois vous avez besoin d'appeler un callback explicitement, il est très facile de faire appeler une fonction de callback à la résolution d'une méthode *async*, en utilisant l'objet Promise retourné. Par exemple, vous pouvez réécrire:

```
module.get_beacon_async(callback, myContext);
```

par

```
module.get_beacon().then(function(res) { callback(myContext, module, res); });
```

Si vous portez une application vers la nouvelle librairie, vous pourriez être amené à désirer des méthodes synchrones similaires à l'ancienne librairie (sans objet Promise), quitte à ce qu'elles retournent la dernière valeur reçue du capteur telle que stockée en cache, puisqu'il n'est pas possible de faire des communications bloquantes. Pour cela, la nouvelle librairie introduit un nouveau type de classes appelés *proxys synchrones*. Un proxy synchrone est un objet qui reflète la dernière value connue d'un objet d'interface, mais peut être accédé à l'aide de fonctions synchrones habituelles. Par exemple, plutôt que d'utiliser:

```
async function logInfo(module)
{
    console.log('Name: '+await module.get_logicalName());
    console.log('Beacon: '+await module.get_beacon());
}

...
```

```
logInfo(myModule);
...
```

on peut utiliser:

```
function logInfoProxy(moduleSyncProxy)
{
    console.log('Name: '+moduleProxy.get_logicalName());
    console.log('Beacon: '+moduleProxy.get_beacon());
}

logInfoSync(await myModule.get_syncProxy());
```

Ce dernier appel asynchrone peut aussi être formulé comme:

```
myModule.get_syncProxy().then(logInfoProxy);
```

2.3. Contrôle de la fonction Led

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

```
// En Node.js, on utilise la fonction require()
// En HTML, on utiliserait &lt;script src="..."&gt;
require('yoctolib-es2017/yocto_api.js');
require('yoctolib-es2017/yocto_led.js');

// On récupère l'objet représentant le module, à travers le VirtualHub local
await YAPI.RegisterHub('127.0.0.1');
var led = YLed.FindLed("YCTOPOC1-123456.led1");

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

Voyons maintenant en détail ce que font ces quelques lignes.

Require de yocto_api et yocto_led

Ces deux imports permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api` doit toujours être inclus, `yocto_led` est nécessaire pour gérer les modules contenant une LED, comme le Yocto-Demo. D'autres classes peuvent être utiles dans d'autres cas, comme `YModule` qui vous permet de faire une énumération de n'importe quel type de module Yoctopuce.

YAPI.RegisterHub

La méthode `RegisterHub` permet d'indiquer sur quelle machine se trouvent 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, ou d'un YoctoHub. Si l'hôte n'est pas joignable, la fonction déclanche une exception.

YLed.FindLed

La méthode `FindLed`, permet de retrouver une LED en fonction du numéro de série de son module hôte et de son nom de fonction. Mais vous pouvez tout aussi bien utiliser des noms logiques que vous auriez préalablement configurés. Imaginons un module Yocto-Demo avec le numéros de série *YCTOPOC1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction

led1 "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
led = YLed.FindLed("YCTOPOC1-123456.led1")
led = YLed.FindLed("YCTOPOC1-123456.MaFonction")
led = YLed.FindLed("MonModule.led1")
led = YLed.FindLed("MonModule.MaFonction")
led = YLed.FindLed("MaFonction")
```

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

isOnline

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

set_power

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

Un exemple concret, en Node.js

Ouvrez une fenêtre de commande (un terminal, un shell...) et allez dans le répertoire **example_nodejs/Doc-GettingStarted-Yocto-Demo** de la librairie Yoctopuce pour JavaScript / EcmaScript 2017. Vous y trouverez un fichier nommé `demo.js` avec le code d'exemple ci-dessous, qui reprend les fonctions expliquées précédemment, mais cette fois utilisées avec le décorum nécessaire à en faire un petit programme d'exemple concret.

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.

```
"use strict";

require('yoctolib-es2017/yocto_api.js');
require('yoctolib-es2017/yocto_led.js');

async function startDemo(args)
{
    await YAPI.LogUnhandledPromiseRejections();
    await YAPI.DisableExceptions();

    // Setup the API to use the VirtualHub on local machine
    let errmsg = new YErrorMsg();
    if(await YAPI.RegisterHub('127.0.0.1', errmsg) != YAPI.SUCCESS) {
        console.log('Cannot contact VirtualHub on 127.0.0.1: '+errmsg.msg);
        return;
    }

    // Select the relay to use
    let target;
    if(args[0] == "any") {
        let anyLed = YLed.FirstLed();
        if (anyLed == null) {
            console.log("No module connected (check USB cable)\n");
            process.exit(1);
        }
        let module = await anyLed.get_module();
        target = await module.get_serialNumber();
    } else {
        target = args[0];
    }

    // Switch relay as requested
    console.log("Turn LED " + args[1]);
    let led = YLed.FindLed(target + ".led");
    if(await led.isOnline()) {
        await led.set_power(args[1] == "ON" ? YLed.POWER_ON : YLed.POWER_OFF);
```

```
    } else {
        console.log("Module not connected (check identification and USB cable)\n");
    }

    await YAPI.FreeAPI();
}

if(process.argv.length < 4) {
    console.log("usage: node demo.js <serial_number> [ ON | OFF ]");
    console.log("       node demo.js <logical_name> [ ON | OFF ]");
    console.log("       node demo.js any [ ON | OFF ]              (use any discovered
device)");
} else {
    startDemo(process.argv.slice(process.argv.length - 2));
}
```

Comme décrit au début de ce chapitre, vous devez avoir installé Node.js v7.6 ou suivant pour essayer ces exemples. Si vous l'avez fait, vous pouvez maintenant taper les deux commandes suivantes pour télécharger automatiquement les librairies dont cet exemple dépend:

```
npm install
```

Une fois terminé, vous pouvez lancer votre code d'exemple dans Node.js avec la commande suivante, en remplaçant les [...] par les arguments que vous voulez passer au programme:

```
node demo.js [...]
```

Le même exemple, mais dans un navigateur

Si vous voulez voir comment utiliser la librairie dans un navigateur plutôt que dans Node.js, changez de répertoire et allez dans **example_html/Doc-GettingStarted-Yocto-Demo**. Vous y trouverez un fichier html, avec une section JavaScript similaire au code précédent, mais avec quelques variantes pour permettre une interaction à travers la page HTML plutôt que sur la console JavaScript

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Hello World</title>
  <script src="../../lib/yocto_api.js"></script>
  <script src="../../lib/yocto_display.js"></script>
  <script>
    var led;

    async function startDemo()
    {
      await YAPI.LogUnhandledPromiseRejections();
      await YAPI.DisableExceptions();

      // Setup the API to use the VirtualHub on local machine
      let errmsg = new YErrorMsg();
      if(await YAPI.RegisterHub('127.0.0.1', errmsg) != YAPI.SUCCESS) {
      alert('Cannot contact VirtualHub on 127.0.0.1: '+errmsg.msg);
    }
      refresh();
    }

    async function refresh()
    {
      let serial = document.getElementById('serial').value;
      if(serial == '') {
        // by default use any connected module suitable for the demo
        let anyLed = YLed.FirstLed();
        if(anyLed) {
          let module = await anyLed.module();
          serial = await module.get_serialNumber();
          document.getElementById('serial').value = serial;
        }
      }
      led = YLed.FindLed(serial+".led");
      if(await led.isOnline()) {
```

```
      document.getElementById('msg').value = '';
    } else {
      document.getElementById('msg').value = 'Module not connected';
    }
      setTimeout(refresh, 500);
    }

    window.sw = function(state)
    {
      led.set_power(state ? YLed.POWER_ON : YLed.POWER_OFF);
    };

    startDemo();
  </script>
</head>
<body>
 Module to use: <input id='serial'>
 <input id='msg' style='color:red;border:none;' readonly><br>
 Turn LED <a href='javascript:sw(0);' >OFF</a> / <a href='javascript:sw(1);'>ON</a><br>
</body>
</html>
```

Aucune installation n'est nécessaire pout utiliser cet exemple, il suffit d'ouvrir la page HTML avec un navigateur web.

2.4. Contrôle de la partie module

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

```
"use strict";

require('yoctolib-es2017/yocto_api.js');

async function startDemo(args)
{
    await YAPI.LogUnhandledPromiseRejections();

    // Setup the API to use the VirtualHub on local machine
    let errmsg = new YErrorMsg();
    if(await YAPI.RegisterHub('127.0.0.1', errmsg) != YAPI.SUCCESS) {
        console.log('Cannot contact VirtualHub on 127.0.0.1: '+errmsg.msg);
        return;
    }

    // Select the relay to use
    let module = YModule.FindModule(args[0]);
    if(await module.isOnline()) {
        if(args.length > 1) {
            if(args[1] == 'ON') {
                await module.set_beacon(YModule.BEACON_ON);
            } else {
                await module.set_beacon(YModule.BEACON_OFF);
            }
        }
        console.log('serial:       '+await module.get_serialNumber());
        console.log('logical name: '+await module.get_logicalName());
        console.log('luminosity:   '+await module.get_luminosity()+'%');
        console.log('beacon:       '+(await module.get_beacon()==YModule.BEACON_ON
?'ON':'OFF'));
        console.log('upTime:       '+parseInt(await module.get_upTime()/1000)+' sec');
        console.log('USB current:  '+await module.get_usbCurrent()+' mA');
        console.log('logs:');
        console.log(await module.get_lastLogs());
    } else {
        console.log("Module not connected (check identification and USB cable)\n");
    }
    await YAPI.FreeAPI();
}

if(process.argv.length < 2) {
    console.log("usage: node demo.js <serial or logicalname> [ ON | OFF ]");
} else {
```

```
    startDemo(process.argv.slice(2));
}
```

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.

```
"use strict";

require('yoctolib-es2017/yocto_api.js');

async function startDemo(args)
{
    await YAPI.LogUnhandledPromiseRejections();

    // Setup the API to use the VirtualHub on local machine
    let errmsg = new YErrorMsg();
    if(await YAPI.RegisterHub('127.0.0.1', errmsg) != YAPI.SUCCESS) {
        console.log('Cannot contact VirtualHub on 127.0.0.1: '+errmsg.msg);
        return;
    }

    // Select the relay to use
    let module = YModule.FindModule(args[0]);
    if(await module.isOnline()) {
        if(args.length > 1) {
            let newname = args[1];
            if (!await YAPI.CheckLogicalName(newname)) {
                console.log("Invalid name (" + newname + ")");
                process.exit(1);
            }
            await module.set_logicalName(newname);
            await module.saveToFlash();
        }
        console.log('Current name: '+await module.get_logicalName());
    } else {
        console.log("Module not connected (check identification and USB cable)\n");
    }
    await YAPI.FreeAPI();
}

if(process.argv.length < 2) {
    console.log("usage: node demo.js <serial> [newLogicalName]");
} else {
    startDemo(process.argv.slice(2));
}
```

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

Énumération des modules

Obtenir la liste des modules connectés se fait à l'aide de la fonction `YModule.FirstModule()` 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

```
"use strict";

require('yoctolib-es2017/yocto_api.js');

async function startDemo()
{
    await YAPI.LogUnhandledPromiseRejections();
    await YAPI.DisableExceptions();

    // Setup the API to use the VirtualHub on local machine
    let errmsg = new YErrorMsg();
    if (await YAPI.RegisterHub('127.0.0.1', errmsg) != YAPI.SUCCESS) {
        console.log('Cannot contact VirtualHub on 127.0.0.1');
        return;
    }
    refresh();
}

async function refresh()
{
    try {
        let errmsg = new YErrorMsg();
        await YAPI.UpdateDeviceList(errmsg);

        let module = YModule.FirstModule();
        while(module) {
            let line = await module.get_serialNumber();
            line += '(' + (await module.get_productName()) + ')';
            console.log(line);
            module = module.nextModule();
        }
        setTimeout(refresh, 500);
    } catch(e) {
        console.log(e);
    }
}

try {
    startDemo();
} catch(e) {
    console.log(e);
}
```

2.5. Gestion des erreurs

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

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

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

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

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `YAPI.DisableExceptions()` 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 même informations qui auraient été associées à l'exception si elles avaient été actives.

3. Reference

3.1. Fonctions générales

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_api.js'></script>
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;
uwp	import com.yoctopuce.YoctoAPI.YModule;
py	from yocto_api import *
php	require_once('yocto_api.php');
es	in HTML: <script src="../../lib/yocto_api.js"></script> in node.js: require('yoctolib-es2017/yocto_api.js');

Fonction globales
yCheckLogicalName(**name**) Vérifie si un nom donné est valide comme nom logique pour un module ou une fonction.
yClearHTTPCallbackCacheDir(**bool_removeFiles**) Désactive le cache de callback HTTP.
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 tolerante 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**)

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

yRegisterLogFunction(**logfun**)

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

ySelectArchitecture(**arch**)

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

ySetDelegate(**object**)

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

ySetHTTPCallbackCacheDir(**str_directory**)

Active le cache du callback HTTP.

ySetTimeout(**callback**, **ms_timeout**, **args**)

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

ySetUSBPacketAckMs(**pktAckDelay**)

Active la quittance des paquets USB reçus par la librairie Yoctopuce.

ySleep(**ms_duration**, **errmsg**)

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

yTestHub(**url**, **mstimeout**, **errmsg**)

Test si un hub est joignable.

yTriggerHubDiscovery(**errmsg**)

Relance une détection des hubs réseau.

yUnregisterHub(**url**)

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

yUpdateDeviceList(**errmsg**)

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

yUpdateDeviceList_async(**callback**, **context**)

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

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

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

```
function CheckLogicalName( name)
```

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

Paramètres :

name une chaîne de caractères contenant le nom vérifier.

Retourne :

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

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

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

```
function DisableExceptions( )
```

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

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

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

function **EnableExceptions**( )

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

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

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

```
function FreeAPI( )
```

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

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

Retourne la version de la librairie Yoctopuce utilisée.

function **GetAPIVersion**( )

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() YAPI yGetTickCount()YAPI.GetTickCount() YAPI.GetTickCount()

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

function **GetTickCount**( )

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

Retourne :
un long entier contenant la valeur du compteur de millisecondes.

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

Maintient la communication de la librairie avec les modules Yoctopuce.

function **HandleEvents**(**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() YAPI yInitAPI()YAPI.InitAPI()YAPI.InitAPI()

Initialise la librairie de programmation de Yoctopuce explicitement.

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

Paramètres :

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

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

Retourne :

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

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

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

Alternative plus tolerante à RegisterHub().

function **PreregisterHub**(**url**, **errmsg**)

Cette fonction a le même but et la même paramètres que la fonction `RegisterHub`, mais contrairement à celle-ci `PreregisterHub()` ne déclanche 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 indépendemment de la connectivité, afin de tenter de ne le contacter que lorsqu'on cherche réellement un module.

Paramètres :

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

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

Retourne :

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

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

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

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

function **RegisterDeviceArrivalCallback**(**arrivalCallback**)

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

Paramètres :

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

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

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

function **RegisterDeviceRemovalCallback**(**removalCallback**)

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

Paramètres :

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

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

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

function **RegisterHub**(**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 languages comme PHP, Javascript et Java ne permettent pas un accès direct aux couches matérielles, **usb** ne marchera donc pas avec ces languages. 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 fonctionnner l'API dans un mode appélé "*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 configuer 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 controle d'accès a été activé, vous devez donner le paramètre url sous la forme:

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

Vous pouvez appeller *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() YAPI ySetTimeout()YAPI.SetTimeout()

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

function **SetTimeout**(**callback**, **ms_timeout**, **args**)

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
args	des arguments supplémentaires peuvent être fournis, pour être passés à la fonction de callback si nécessaire.

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

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

```
function Sleep( 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.TestHub() YAPI yTestHub()YAPI.TestHub()YAPI.TestHub()

Test si un hub est joignable.

function **TestHub**(**url**, **mstimeout**)

Cette méthode n'enregistre pas le hub, elle ne fait que de vérifier que le hub est joignable. Le paramètre url suit les mêmes conventions que la méthode `RegisterHub`. Cette méthode est utile pour vérifier les paramètres d'authentification d'un hub. Il est possible de forcer la méthode à rendre la main après mstimeout millisecondes.

Paramètres :

url	une chaîne de caractères contenant **"usb"**,**"callback",** ou l'URL racine du VirtualHub à utiliser.
mstimeout	le nombre de millisecondes disponible pour tester la connexion.
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 retourne un code d'erreur négatif.

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

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

function **UnregisterHub**(**url**)

Paramètres :

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

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

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

function **UpdateDeviceList**(**errmsg**)

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

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

Paramètres :

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

Retourne :

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

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

3.2. Interface de la fonction Accelerometer

La classe YSensor est la classe parente de tous les senseurs Yoctopuce. Elle permet de lire la valeur courante et l'unité de n'importe quel capteur, de lire les valeurs min/max, de configurer la fréquence d'enregistrement autonome des données et de récupérer les mesures enregistrées. Elle permet aussi d'enregistrer un callback appelé lorsque la valeur mesurée change ou à intervalle prédéfini. L'utilisation de cette classe plutôt qu'une de ces sous-classes permet de créer des application génériques, compatibles même avec les capteurs Yoctopuce futurs. Note: la classe YAnButton est le seul type d'entrée analogique qui n'hérite pas de YSensor.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_accelerometer.js'></script>
cpp	#include "yocto_accelerometer.h"
m	#import "yocto_accelerometer.h"
pas	uses yocto_accelerometer;
vb	yocto_accelerometer.vb
cs	yocto_accelerometer.cs
java	import com.yoctopuce.YoctoAPI.YAccelerometer;
uwp	import com.yoctopuce.YoctoAPI.YAccelerometer;
py	from yocto_accelerometer import *
php	require_once('yocto_accelerometer.php');
es	in HTML: <script src="../../lib/yocto_accelerometer.js"></script> in node.js: require('yoctolib-es2017/yocto_accelerometer.js');

Fonction globales

yFindAccelerometer(**func**)

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

yFindAccelerometerInContext(**yctx**, **func**)

Permet de retrouver un accéléromètre d'après un identifiant donné dans un Context YAPI.

yFirstAccelerometer()

Commence l'énumération des accéléromètres accessibles par la librairie.

yFirstAccelerometerInContext(**yctx**)

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→clearCache()

Invalide le cache.

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_bandwidth()

Retourne la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

accelerometer→get_currentRawValue()

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

accelerometer→get_currentValue()

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

accelerometer→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

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_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

accelerometer→load(**msValidity**)

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

accelerometer→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

accelerometer→nextAccelerometer()

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

accelerometer→registerTimedReportCallback(**callback**)

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

accelerometer→registerValueCallback(**callback**)

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

accelerometer→set_bandwidth(**newval**)

Modifie la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

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→startDataLogger()

Démarre l'enregistreur de données du module.

accelerometer→stopDataLogger()

Arrête l'enregistreur de données du module.

accelerometer→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

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

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

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

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

function **FindAccelerometer**(**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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindAccelerometerInContext() YAccelerometer yFindAccelerometerInContext() YAccelerometer.FindAccelerometerInContext() YAccelerometer.FindAccelerometerInContext()

Permet de retrouver un accéléromètre d'après un identifiant donné dans un Context YAPI.

function **FindAccelerometerInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

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

Retourne :

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

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

Commence l'énumération des accéléromètres accessibles par la librairie.

function **FirstAccelerometer**( )

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.

YAccelerometer.FirstAccelerometerInContext() YAccelerometer yFirstAccelerometerInContext() YAccelerometer.FirstAccelerometerInContext() YAccelerometer.FirstAccelerometerInContext()

Commence l'énumération des accéléromètres accessibles par la librairie.

function **FirstAccelerometerInContext**(**yctx**)

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

Paramètres :

yctx un contexte YAPI.

Retourne :

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

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

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

function **calibrateFromPoints**(**rawValues**, **refValues**)

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

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

Paramètres :

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

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

Retourne :

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

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

accelerometer→clearCache() YAccelerometer
accelerometer.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'accéléromètre. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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

accelerometer→get_advertisedValue() YAccelerometer
accelerometer→advertisedValue()
accelerometer.get_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_bandwidth() YAccelerometer
accelerometer→bandwidth()
accelerometer.get_bandwidth()
accelerometer.get_bandwidth()

Retourne la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

function **get_bandwidth**( )

Retourne :
un entier représentant la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement)

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

accelerometer→get_currentRawValue() YAccelerometer
accelerometer→currentRawValue()
accelerometer.get_currentRawValue()
accelerometer.get_currentRawValue()

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

function **get_currentRawValue**( )

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

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

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

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

function **get_currentValue**( )

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

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

accelerometer→get_dataLogger() YAccelerometer
accelerometer→dataLogger()
accelerometer.get_dataLogger()
accelerometer.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

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'identifant matériel de l'accéléromètre (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant l'accéléromètre en utilisant les noms logiques (ex: MyCustomName.relay1)

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

accelerometer→get_functionDescriptor() YAccelerometer
accelerometer→functionDescriptor()
accelerometer.get_functionDescriptor()

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

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 example `relay1`.

Retourne :

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

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

accelerometer→get_hardwareId() 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 example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant l'accéléromètre (ex: RELAYLO1-123456.relay1)

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

accelerometer→get_highestValue() YAccelerometer
accelerometer→highestValue()
accelerometer.get_highestValue()
accelerometer.get_highestValue()

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()
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()
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()
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_recordedData() YAccelerometer
accelerometer→recordedData()
accelerometer.get_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**)

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 find de l'intercalle 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()
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()
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_sensorState() YAccelerometer
accelerometer→sensorState()
accelerometer.get_sensorState()
accelerometer.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

```
function get_sensorState( )
```

Retourne :

un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

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

accelerometer→get_unit() YAccelerometer
accelerometer→unit()accelerometer.get_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() YAccelerometer
accelerometer→userData()
accelerometer.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.

accelerometer→get_xValue() YAccelerometer
accelerometer→xValue()accelerometer.get_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()
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()
accelerometer.get_zValue()

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

```
function get_zValue( )
```

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

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

accelerometer→isOnline()accelerometer.isOnline() YAccelerometer

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

function **isOnline**( )

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

Retourne :

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

accelerometer→load()accelerometer.load() YAccelerometer

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

function **load**(**msValidity**)

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

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

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

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

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

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

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

Retourne :

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

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

accelerometer→muteValueCallbacks() YAccelerometer
accelerometer.muteValueCallbacks()
accelerometer.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

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

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

accelerometer→nextAccelerometer() YAccelerometer
accelerometer.nextAccelerometer()
accelerometer.nextAccelerometer()

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() YAccelerometer
accelerometer.registerTimedReportCallback()
accelerometer.registerTimedReportCallback()

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() YAccelerometer
accelerometer.registerValueCallback()
accelerometer.registerValueCallback()

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_bandwidth() YAccelerometer
accelerometer→setBandwidth()
accelerometer.set_bandwidth()
accelerometer.set_bandwidth()

Modifie la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

function **set_bandwidth**(**newval**)

Lorsque la fréquence est plus basse, un moyennage est effectué.

Paramètres :

newval un entier représentant la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement)

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_highestValue() YAccelerometer
accelerometer→setHighestValue()
accelerometer.set_highestValue()
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()
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()
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() YAccelerometer
accelerometer→setLowestValue()
accelerometer.set_lowestValue()
accelerometer.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.

accelerometer→set_reportFrequency() YAccelerometer
accelerometer→setReportFrequency()
accelerometer.set_reportFrequency()
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() YAccelerometer
accelerometer→setResolution()
accelerometer.set_resolution()
accelerometer.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.

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→startDataLogger() YAccelerometer
accelerometer.startDataLogger()
accelerometer.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

accelerometer→stopDataLogger() YAccelerometer
accelerometer.stopDataLogger()
accelerometer.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

accelerometer→unmuteValueCallbacks() YAccelerometer
accelerometer.unmuteValueCallbacks()
accelerometer.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→wait_async() YAccelerometer
accelerometer.wait_async()

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 Altitude

La classe YAltitude permet de lire et de configurer les capteurs d'altitude Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, pour les capteurs barométriques, elle permet de faire la configuration de la pression de référence au niveau de la mer (QNH).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_altitude.js'></script>
cpp	#include "yocto_altitude.h"
m	#import "yocto_altitude.h"
pas	uses yocto_altitude;
vb	yocto_altitude.vb
cs	yocto_altitude.cs
java	import com.yoctopuce.YoctoAPI.YAltitude;
uwp	import com.yoctopuce.YoctoAPI.YAltitude;
py	from yocto_altitude import *
php	require_once('yocto_altitude.php');
es	in HTML: <script src="../../lib/yocto_altitude.js"></script> in node.js: require('yoctolib-es2017/yocto_altitude.js');

Fonction globales

yFindAltitude(**func**)

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

yFindAltitudeInContext(**yctx**, **func**)

Permet de retrouver un altimetre d'après un identifiant donné dans un Context YAPI.

yFirstAltitude()

Commence l'énumération des altimètres accessibles par la librairie.

yFirstAltitudeInContext(**yctx**)

Commence l'énumération des altimètres accessibles par la librairie.

Méthodes des objets `YAltitude`

altitude→calibrateFromPoints(**rawValues**, **refValues**)

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

altitude→clearCache()

Invalide le cache.

altitude→describe()

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

altitude→get_advertisedValue()

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

altitude→get_currentRawValue()

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

altitude→get_currentValue()

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

altitude→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

altitude→get_errorMessage()

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

altitude→get_errorType()

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

altitude→get_friendlyName()

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

altitude→get_functionDescriptor()

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

altitude→get_functionId()

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

altitude→get_hardwareId()

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

altitude→get_highestValue()

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

altitude→get_logFrequency()

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

altitude→get_logicalName()

Retourne le nom logique de l'altimètre.

altitude→get_lowestValue()

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

altitude→get_module()

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

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

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

altitude→get_qnh()

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

altitude→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.

altitude→get_reportFrequency()

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

altitude→get_resolution()

Retourne la résolution des valeurs mesurées.

altitude→get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

altitude→get_technology()

Renvoie la technologie employée par la fonction pour calculer l'altitude.

altitude→get_unit()

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

altitude→get_userData()

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

altitude→isOnline()

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

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

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

altitude→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

altitude→load(**msValidity**)

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

altitude→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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

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

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

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

altitude→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

altitude→nextAltitude()

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

altitude→registerTimedReportCallback(**callback**)

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

altitude→registerValueCallback(**callback**)

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

altitude→set_currentValue(**newval**)

Modifie l'altitude actuelle supposée.

altitude→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

altitude→set_logFrequency(**newval**)

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

altitude→set_logicalName(**newval**)

Modifie le nom logique de l'altimètre.

altitude→set_lowestValue(**newval**)

Modifie la mémoire de valeur minimale observée.

altitude→set_qnh(**newval**)

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

altitude→set_reportFrequency(**newval**)

Modifie la fréquence de notification périodique des valeurs mesurées.

altitude→set_resolution(**newval**)

Modifie la résolution des valeurs physique mesurées.

altitude→set_userData(**data**)

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

altitude→startDataLogger()

Démarre l'enregistreur de données du module.

altitude→stopDataLogger()

Arrête l'enregistreur de données du module.

altitude→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

altitude→wait_async(callback, context)

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

YAltitude.FindAltitude() YAltitude yFindAltitude()YAltitude.FindAltitude() YAltitude.FindAltitude()

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

```
function FindAltitude( 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'altimètre soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YAltitude.isOnline()` pour tester si l'altimètre est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

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

Retourne :

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

YAltitude.FindAltitudeInContext() YAltitude
yFindAltitudeInContext()
YAltitude.FindAltitudeInContext()
YAltitude.FindAltitudeInContext()

Permet de retrouver un altimetre d'après un identifiant donné dans un Context YAPI.

function **FindAltitudeInContext**(**yctx**, **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'altimètre soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YAltitude.isOnline()` pour tester si l'altimètre est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

yctx un contexte YAPI

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

Retourne :

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

YAltitude.FirstAltitude() YAltitude yFirstAltitude()YAltitude.FirstAltitude() YAltitude.FirstAltitude()

Commence l'énumération des altimètres accessibles par la librairie.

function **FirstAltitude**( )

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

Retourne :

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

YAltitude.FirstAltitudeInContext() YAltitude
yFirstAltitudeInContext()
YAltitude.FirstAltitudeInContext()
YAltitude.FirstAltitudeInContext()

Commence l'énumération des altimètres accessibles par la librairie.

function **FirstAltitudeInContext**(**yctx**)

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

Paramètres :

yctx un contexte YAPI.

Retourne :

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

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

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

function **calibrateFromPoints**(**rawValues**, **refValues**)

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

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

Paramètres :

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

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

Retourne :

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

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

altitude→clearCache()altitude.clearCache() YAltitude

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'altimètre. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

altitude→describe()altitude.describe() YAltitude

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

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

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

altitude→get_advertisedValue() YAltitude
altitude→advertisedValue()
altitude.get_advertisedValue()
altitude.get_advertisedValue()

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

function **get_advertisedValue**( )

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

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

altitude→get_currentRawValue() YAltitude
altitude→currentRawValue()
altitude.get_currentRawValue()
altitude.get_currentRawValue()

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

function **get_currentRawValue**( )

Retourne :

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

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

altitude→get_currentValue() YAltitude
altitude→currentValue()altitude.get_currentValue()
altitude.get_currentValue()

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

function **get_currentValue**( )

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

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

altitude→get_dataLogger() YAltitude
altitude→dataLogger()altitude.get_dataLogger()
altitude.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

altitude→get_errorMessage() YAltitude
altitude→errorMessage()altitude.get_errorMessage()

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

```
function get_errorMessage( )
```

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

Retourne :

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

altitude→get_errorType() YAltitude
altitude→errorType()altitude.get_errorType()

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

function **get_errorType**( )

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

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

altitude→get_friendlyName() YAltitude
altitude→friendlyName()altitude.get_friendlyName()

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

function **get_friendlyName**( )

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

Retourne :
une chaîne de caractères identifiant l'altimètre en utilisant les noms logiques (ex: MyCustomName.relay1)

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

altitude→get_functionDescriptor() YAltitude
altitude→functionDescriptor()
altitude.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

altitude→get_functionId() YAltitude
altitude→functionId()altitude.get_functionId()

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

function **get_functionId**( )

Par example `relay1`.

Retourne :

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

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

altitude→get_hardwareId() YAltitude
altitude→hardwareId()altitude.get_hardwareId()

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

function **get_hardwareId**( )

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

Retourne :

une chaîne de caractères identifiant l'altimètre (ex: RELAYLO1-123456.relay1)

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

altitude→get_highestValue() YAltitude
altitude→highestValue()altitude.get_highestValue()
altitude.get_highestValue()

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

function **get_highestValue**( )

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

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

altitude→get_logFrequency() YAltitude
altitude→logFrequency()altitude.get_logFrequency()
altitude.get_logFrequency()

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

function **get_logFrequency**( )

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`.

altitude→get_logicalName() YAltitude
altitude→logicalName()altitude.get_logicalName()
altitude.get_logicalName()

Retourne le nom logique de l'altimètre.

function **get_logicalName**( )

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

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

altitude→get_lowestValue() YAltitude
altitude→lowestValue()altitude.get_lowestValue()
altitude.get_lowestValue()

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

function **get_lowestValue**( )

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

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

altitude→get_module() YAltitude
altitude→module()altitude.get_module()

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

function **get_module**( )

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

Retourne :
une instance de YModule

altitude→get_qnh() YAltitude
altitude→qnh()altitude.get_qnh()altitude.get_qnh()

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

function **get_qnh**( )

Retourne :

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

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

altitude→get_recordedData() YAltitude
altitude→recordedData()altitude.get_recordedData()
altitude.get_recordedData()

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

function **get_recordedData**(**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 find de l'intercalle 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.

altitude→get_reportFrequency() YAltitude
altitude→reportFrequency()
altitude.get_reportFrequency()
altitude.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`.

altitude→get_resolution() YAltitude
altitude→resolution()altitude.get_resolution()
altitude.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`.

altitude→get_sensorState() YAltitude
altitude→sensorState()altitude.get_sensorState()
altitude.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :

un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

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

altitude→get_technology() YAltitude
altitude→technology()altitude.get_technology()
altitude.get_technology()

Renvoie la technologie employée par la fonction pour calculer l'altitude.

```
function get_technology( )
```

les valeur possible sont "barometric" et "gps"

Retourne :
une chaîne de caractères

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

altitude→get_unit() YAltitude
altitude→unit()altitude.get_unit()altitude.get_unit()

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

function **get_unit**( )

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

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

altitude→get_userData() YAltitude
altitude→userData()altitude.get_userData()

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

function **get_userData**( )

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.

altitude→isOnline()altitude.isOnline() YAltitude

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

function **isOnline**( )

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

Retourne :
`true` si l'altimètre est joignable, `false` sinon

altitude→load()altitude.load() YAltitude

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

function **load**(**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 occasionellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→loadAttribute()altitude.loadAttribute() YAltitude altitude.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

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

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

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

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

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

Retourne :

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

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

altitude→muteValueCallbacks() YAltitude
altitude.muteValueCallbacks()
altitude.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

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

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

altitude→nextAltitude()altitude.nextAltitude() altitude.nextAltitude() — YAltitude

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

function **nextAltitude**( )

Retourne :

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

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

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.

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

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.

altitude→set_currentValue() YAltitude
altitude→setCurrentValue()
altitude.set_currentValue()altitude.set_currentValue()

Modifie l'altitude actuelle supposée.

function **set_currentValue**(**newval**)

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

Paramètres :

newval une valeur numérique représentant l'altitude actuelle supposée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→set_highestValue() YAltitude
altitude→setHighestValue()
altitude.set_highestValue()
altitude.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.

altitude→set_logFrequency() YAltitude
altitude→setLogFrequency()
altitude.set_logFrequency()
altitude.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.

altitude→set_logicalName() YAltitude
altitude→setLogicalName()altitude.set_logicalName()
altitude.set_logicalName()

Modifie le nom logique de l'altimètre.

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'altimètre.

Retourne :

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

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

altitude→set_lowestValue() YAltitude
altitude→setLowestValue()altitude.set_lowestValue()
altitude.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.

altitude→set_qnh() YAltitude
altitude→setQnh()altitude.set_qnh()altitude.set_qnh()

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

function **set_qnh**(**newval**)

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→set_reportFrequency() YAltitude
altitude→setReportFrequency()
altitude.set_reportFrequency()
altitude.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.

altitude→set_resolution() YAltitude
altitude→setResolution()altitude.set_resolution()
altitude.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.

altitude→set_userData() YAltitude
altitude→setUserData()altitude.set_userData()

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

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

altitude→startDataLogger()altitude.startDataLogger() YAltitude altitude.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

altitude→stopDataLogger()altitude.stopDataLogger() YAltitude altitude.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

altitude→unmuteValueCallbacks() YAltitude
altitude.unmuteValueCallbacks()
altitude.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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

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

altitude→wait_async()altitude.wait_async() YAltitude

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 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 exmple un bouton rotatif continu, une poignée de commande de gaz ou un joystick. Le module est capable de se calibrer sur les valeurs minimales et maximales du potentiomètre, et de restituer une valeur calibrée variant proportionnellement avec la position du potentiomètre, indépendant de sa résistance totale.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_anbutton.js'></script>
cpp	#include "yocto_anbutton.h"
m	#import "yocto_anbutton.h"
pas	uses yocto_anbutton;
vb	yocto_anbutton.vb
cs	yocto_anbutton.cs
java	import com.yoctopuce.YoctoAPI.YAnButton;
uwp	import com.yoctopuce.YoctoAPI.YAnButton;
py	from yocto_anbutton import *
php	require_once('yocto_anbutton.php');
es	in HTML: <script src="../../lib/yocto_anbutton.js"></script> in node.js: require('yoctolib-es2017/yocto_anbutton.js');

Fonction globales

yFindAnButton(**func**)

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

yFindAnButtonInContext(**yctx**, **func**)

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

yFirstAnButton()

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

yFirstAnButtonInContext(**yctx**)

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

Méthodes des objets YAnButton

anbutton→clearCache()

Invalide le cache.

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 déclanchement 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→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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

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

anbutton→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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**)

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 déclanchement 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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

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

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

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

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

```
function FindAnButton( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindAnButtonInContext() YAnButton
yFindAnButtonInContext()
YAnButton.FindAnButtonInContext()
YAnButton.FindAnButtonInContext()

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

function **FindAnButtonInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

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

Retourne :

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

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

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

function **FirstAnButton**( )

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.

YAnButton.FirstAnButtonInContext() YAnButton
yFirstAnButtonInContext()
YAnButton.FirstAnButtonInContext()
YAnButton.FirstAnButtonInContext()

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

function **FirstAnButtonInContext**(**yctx**)

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

Paramètres :

yctx un contexte YAPI.

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→clearCache()anbutton.clearCache() YAnButton

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'entrée analogique. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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()
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()
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()
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()
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() YAnButton
anbutton→calibrationMin()
anbutton.get_calibrationMin()
anbutton.get_calibrationMin()

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.get_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'identifant 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() YAnButton
anbutton→functionDescriptor()
anbutton.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

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 example `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 example 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()
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() YAnButton
anbutton→lastTimePressed()
anbutton.get_lastTimePressed()
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).

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()
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_LASTTIMERELEASED_INVALID`.

anbutton→get_logicalName() YAnButton
anbutton→logicalName()anbutton.get_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_pulseCounter() YAnButton
anbutton→pulseCounter()
anbutton.get_pulseCounter()
anbutton.get_pulseCounter()

Retourne la valeur du compteur d'impulsions.

function **get_pulseCounter**( )

La valeur est codée sur 32 bits. En cas de dépassement de capacite (>=2^32), le compteur repart à zéro. Le compteur peut être réinitialisé en appelant la méthode resetCounter().

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()
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()
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()
anbutton.get_sensitivity()

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

function **get_sensitivity**( )

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

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

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

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

function **get_userData**( )

Cet attribut n'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→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 occasionellement 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→loadAttribute()anbutton.loadAttribute() YAnButton anbutton.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

anbutton→muteValueCallbacks() YAnButton
anbutton.muteValueCallbacks()
anbutton.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

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

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

anbutton→nextAnButton()anbutton.nextAnButton() YAnButton anbutton.nextAnButton()

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() YAnButton
anbutton.registerValueCallback()
anbutton.registerValueCallback()

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 anbutton.resetCounter()

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() YAnButton
anbutton→setAnalogCalibration()
anbutton.set_analogCalibration()
anbutton.set_analogCalibration()

Enclenche ou déclenche le procédure de calibration.

function **set_analogCalibration**(**newval**)

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

Paramètres :

newval soit `Y_ANALOGCALIBRATION_OFF`, soit `Y_ANALOGCALIBRATION_ON`

Retourne :

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

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

anbutton→set_calibrationMax() YAnButton
anbutton→setCalibrationMax()
anbutton.set_calibrationMax()
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()
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() YAnButton
anbutton→setLogicalName()
anbutton.set_logicalName()
anbutton.set_logicalName()

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()
anbutton.set_sensitivity()

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

function **set_sensitivity**(**newval**)

La sensibilité sert à filtrer les variations autour d'une valeur fixe, mais ne prétérite 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 déclanchement 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→unmuteValueCallbacks() YAnButton
anbutton.unmuteValueCallbacks()
anbutton.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.5. Interface de la fonction AudioIn

La librairie de programmation Yoctopuce permet de configurer le gain de l'entrée audio.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_audioin.js'></script>
cpp	#include "yocto_audioin.h"
m	#import "yocto_audioin.h"
pas	uses yocto_audioin;
vb	yocto_audioin.vb
cs	yocto_audioin.cs
java	import com.yoctopuce.YoctoAPI.YAudioIn;
uwp	import com.yoctopuce.YoctoAPI.YAudioIn;
py	from yocto_audioin import *
php	require_once('yocto_audioin.php');
es	in HTML: <script src="../../lib/yocto_audioin.js"></script> in node.js: require('yoctolib-es2017/yocto_audioin.js');

Fonction globales
yFindAudioIn(**func**) Permet de retrouver une entrée audio d'après un identifiant donné.
yFindAudioInInContext(**yctx**, **func**) Permet de retrouver une entrée audio d'après un identifiant donné dans un Context YAPI.
yFirstAudioIn() Commence l'énumération des entrées audio accessibles par la librairie.
yFirstAudioInInContext(**yctx**) Commence l'énumération des entrées audio accessibles par la librairie.
Méthodes des objets `YAudioIn`
audioin→clearCache() Invalide le cache.
audioin→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de la entrée audio au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
audioin→get_advertisedValue() Retourne la valeur courante de la entrée audio (pas plus de 6 caractères).
audioin→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la entrée audio.
audioin→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la entrée audio.
audioin→get_friendlyName() Retourne un identifiant global de la entrée audio au format `NOM_MODULE.NOM_FONCTION`.
audioin→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
audioin→get_functionId() Retourne l'identifiant matériel de la entrée audio, sans référence au module.
audioin→get_hardwareId() Retourne l'identifiant matériel unique de la entrée audio au format `SERIAL.FUNCTIONID`.
audioin→get_logicalName()

Retourne le nom logique de la entrée audio.

audioin→get_module()

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

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

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

audioin→get_mute()

Retourne l'état de la fonction silencieux (mute) de l'entrée audio.

audioin→get_noSignalFor()

Retourne le nombre de secondes sans signal détecté.

audioin→get_signal()

Retourne l'amplitude du signal d'entrée détecté.

audioin→get_userData()

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

audioin→get_volume()

Retourne le gain de l'entrée audio, en pour cents.

audioin→get_volumeRange()

Retourne la plage de valeurs supportées pour le volume.

audioin→isOnline()

Vérifie si le module hébergeant la entrée audio est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant la entrée audio est joignable, sans déclencher d'erreur.

audioin→load(**msValidity**)

Met en cache les valeurs courantes de la entrée audio, avec une durée de validité spécifiée.

audioin→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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

Met en cache les valeurs courantes de la entrée audio, avec une durée de validité spécifiée.

audioin→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

audioin→nextAudioIn()

Continue l'énumération des entrées audio commencée à l'aide de `yFirstAudioIn()`.

audioin→registerValueCallback(**callback**)

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

audioin→set_logicalName(**newval**)

Modifie le nom logique de la entrée audio.

audioin→set_mute(**newval**)

Modifie l'état de la fonction silencieux (mute) de l'entrée audio.

audioin→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`.

audioin→set_volume(**newval**)

Modifie le gain de la l'entrée audio, en pour cents.

audioin→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

audioin→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.

YAudioIn.FindAudioIn() YAudioIn yFindAudioIn()YAudioIn.FindAudioIn() YAudioIn.FindAudioIn()

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

```
function FindAudioIn( 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 entrée audio soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YAudioIn.isOnline()` pour tester si la entrée audio 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence la entrée audio sans ambiguïté

Retourne :

un objet de classe `YAudioIn` qui permet ensuite de contrôler la entrée audio.

YAudioIn.FindAudioInInContext() YAudioIn
yFindAudioInInContext()
YAudioIn.FindAudioInInContext()
YAudioIn.FindAudioInInContext()

Permet de retrouver une entrée audio d'après un identifiant donné dans un Context YAPI.

```
function FindAudioInInContext( yctx, 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 entrée audio soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YAudioIn.isOnline()` pour tester si la entrée audio 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence la entrée audio sans ambiguïté

Retourne :

un objet de classe `YAudioIn` qui permet ensuite de contrôler la entrée audio.

YAudioIn.FirstAudioIn() YAudioIn yFirstAudioIn()YAudioIn.FirstAudioIn() YAudioIn.FirstAudioIn()

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

function **FirstAudioIn**( )

Utiliser la fonction `YAudioIn.nextAudioIn()` pour itérer sur les autres entrées audio.

Retourne :

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

YAudioIn.FirstAudioInInContext() YAudioIn
yFirstAudioInInContext()
YAudioIn.FirstAudioInInContext()
YAudioIn.FirstAudioInInContext()

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

function **FirstAudioInInContext**(**yctx**)

Utiliser la fonction `YAudioIn.nextAudioIn()` pour itérer sur les autres entrées audio.

Paramètres :

yctx un contexte YAPI.

Retourne :

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

audioin→clearCache()audioin.clearCache() YAudioIn

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de la entrée audio. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

audioin→describe()audioin.describe() YAudioIn

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

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant la entrée audio (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

audioin→get_advertisedValue() YAudioIn
audioin→advertisedValue()
audioin.get_advertisedValue()
audioin.get_advertisedValue()

Retourne la valeur courante de la entrée audio (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante de la entrée audio (pas plus de 6 caractères).

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

audioin→get_errorMessage() YAudioIn
audioin→errorMessage()audioin.get_errorMessage()

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

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 entrée audio.

audioin→get_errorType() YAudioIn
audioin→errorType()audioin.get_errorType()

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

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 entrée audio.

audioin→get_friendlyName() YAudioIn
audioin→friendlyName()audioin.get_friendlyName()

Retourne un identifiant global de la entrée audio au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

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

Retourne :

une chaîne de caractères identifiant la entrée audio en utilisant les noms logiques (ex: MyCustomName.relay1)

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

audioin→get_functionDescriptor() YAudioIn
audioin→functionDescriptor()
audioin.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

audioin→get_functionId() YAudioIn
audioin→functionId()audioin.get_functionId()

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

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant la entrée audio (ex: `relay1`)

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

audioin→get_hardwareId() YAudioIn
audioin→hardwareId()audioin.get_hardwareId()

Retourne l'identifiant matériel unique de la entrée audio 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 entrée audio (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant la entrée audio (ex: RELAYLO1-123456.relay1)

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

audioin→get_logicalName() YAudioIn
audioin→logicalName()audioin.get_logicalName()
audioin.get_logicalName()

Retourne le nom logique de la entrée audio.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique de la entrée audio.

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

audioin→get_module() YAudioIn
audioin→module()audioin.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

audioin→get_mute() YAudioIn
audioin→mute()audioin.get_mute()
audioin.get_mute()

Retourne l'état de la fonction silencieux (mute) de l'entrée audio.

function **get_mute**( )

Retourne :

soit Y_MUTE_FALSE, soit Y_MUTE_TRUE, selon l'état de la fonction silencieux (mute) de l'entrée audio

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

audioin→get_noSignalFor() YAudioIn
audioin→noSignalFor()audioin.get_noSignalFor()
audioin.get_noSignalFor()

Retourne le nombre de secondes sans signal détecté.

function **get_noSignalFor**( )

Retourne :
un entier représentant le nombre de secondes sans signal détecté

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

audioin→get_signal() YAudioIn
audioin→signal()audioin.get_signal()
audioin.get_signal()

Retourne l'amplitude du signal d'entrée détecté.

function **get_signal**( )

Retourne :
un entier représentant l'amplitude du signal d'entrée détecté

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

audioin→get_userData() YAudioIn
audioin→userData()audioin.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.

audioin→get_volume() YAudioIn
audioin→volume()audioin.get_volume()
audioin.get_volume()

Retourne le gain de l'entrée audio, en pour cents.

function **get_volume**( )

Retourne :
un entier représentant le gain de l'entrée audio, en pour cents

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

audioin→get_volumeRange() YAudioIn
audioin→volumeRange()audioin.get_volumeRange()
audioin.get_volumeRange()

Retourne la plage de valeurs supportées pour le volume.

function **get_volumeRange**( )

La valeur basse de l'intervalle correspond au volume minimal audible. Pour couper complètement le son, utilisez `set_mute()` et non le `set_volume()`.

Retourne :
une chaîne de caractères représentant la plage de valeurs supportées pour le volume

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

audioin→isOnline()audioin.isOnline() YAudioIn

Vérifie si le module hébergeant la entrée audio est joignable, sans déclencher d'erreur.

function **isOnline**( )

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

audioin→load()audioin.load() YAudioIn

Met en cache les valeurs courantes de la entrée audio, 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 occasionellement 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.

audioin→loadAttribute()audioin.loadAttribute() YAudioIn audioin.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

audioin→muteValueCallbacks() YAudioIn
audioin.muteValueCallbacks()
audioin.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

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

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

audioin→nextAudioIn()audioin.nextAudioIn() YAudioIn audioin.nextAudioIn()

Continue l'énumération des entrées audio commencée à l'aide de `yFirstAudioIn()`.

function **nextAudioIn**( )

Retourne :

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

audioin→registerValueCallback() YAudioIn
audioin.registerValueCallback()
audioin.registerValueCallback()

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.

audioin→set_logicalName() YAudioIn
audioin→setLogicalName()audioin.set_logicalName()
audioin.set_logicalName()

Modifie le nom logique de la entrée audio.

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 entrée audio.

Retourne :

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

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

audioin→set_mute() YAudioIn
audioin→setMute()audioin.set_mute()
audioin.set_mute()

Modifie l'état de la fonction silencieux (mute) de l'entrée audio.

function **set_mute**(**newval**)

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si vous désirez que le réglage soit préservé au prochain redémarrage.

Paramètres :

newval soit `Y_MUTE_FALSE`, soit `Y_MUTE_TRUE`, selon l'état de la fonction silencieux (mute) de l'entrée audio

Retourne :

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

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

audioin→set_userData() YAudioIn
audioin→setUserData()audioin.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

audioin→set_volume() YAudioIn
audioin→setVolume()audioin.set_volume()
audioin.set_volume()

Modifie le gain de la l'entrée audio, en pour cents.

function **set_volume**(**newval**)

Paramètres :

newval un entier représentant le gain de la l'entrée audio, 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.

audioin→unmuteValueCallbacks() YAudioIn
audioin.unmuteValueCallbacks()
audioin.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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

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

audioin→wait_async()audioin.wait_async() YAudioIn

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 AudioOut

La librairie de programmation Yoctopuce permet de configurer le volume de la sortie audio.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_audioout.js'></script>
cpp	#include "yocto_audioout.h"
m	#import "yocto_audioout.h"
pas	uses yocto_audioout;
vb	yocto_audioout.vb
cs	yocto_audioout.cs
java	import com.yoctopuce.YoctoAPI.YAudioOut;
uwp	import com.yoctopuce.YoctoAPI.YAudioOut;
py	from yocto_audioout import *
php	require_once('yocto_audioout.php');
es	in HTML: <script src="../../lib/yocto_audioout.js"></script> in node.js: require('yoctolib-es2017/yocto_audioout.js');

Fonction globales
yFindAudioOut(**func**) Permet de retrouver une sortie audio d'après un identifiant donné.
yFindAudioOutInContext(**yctx**, **func**) Permet de retrouver une sortie audio d'après un identifiant donné dans un Context YAPI.
yFirstAudioOut() Commence l'énumération des sorties audio accessibles par la librairie.
yFirstAudioOutInContext(**yctx**) Commence l'énumération des sorties audio accessibles par la librairie.
Méthodes des objets YAudioOut
audioout→clearCache() Invalide le cache.
audioout→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de la sortie audio au format TYPE(NAME)=SERIAL.FUNCTIONID.
audioout→get_advertisedValue() Retourne la valeur courante de la sortie audio (pas plus de 6 caractères).
audioout→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la sortie audio.
audioout→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la sortie audio.
audioout→get_friendlyName() Retourne un identifiant global de la sortie audio au format NOM_MODULE.NOM_FONCTION.
audioout→get_functionDescriptor() Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
audioout→get_functionId() Retourne l'identifiant matériel de la sortie audio, sans référence au module.
audioout→get_hardwareId() Retourne l'identifiant matériel unique de la sortie audio au format SERIAL.FUNCTIONID.
audioout→get_logicalName()

Retourne le nom logique de la sortie audio.

audioout→get_module()

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

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

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

audioout→get_mute()

Retourne l'état de la fonction silencieux (mute) de la sortie audio.

audioout→get_noSignalFor()

Retourne le nombre de secondes sans signal détecté.

audioout→get_signal()

Retourne l'amplitude du courant émis détecté.

audioout→get_userData()

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

audioout→get_volume()

Retourne le volume de la sortie audio, en pour cents.

audioout→get_volumeRange()

Retourne la plage de valeurs supportées pour le volume.

audioout→isOnline()

Vérifie si le module hébergeant la sortie audio est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant la sortie audio est joignable, sans déclencher d'erreur.

audioout→load(**msValidity**)

Met en cache les valeurs courantes de la sortie audio, avec une durée de validité spécifiée.

audioout→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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

Met en cache les valeurs courantes de la sortie audio, avec une durée de validité spécifiée.

audioout→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

audioout→nextAudioOut()

Continue l'énumération des sorties audio commencée à l'aide de `yFirstAudioOut()`.

audioout→registerValueCallback(**callback**)

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

audioout→set_logicalName(**newval**)

Modifie le nom logique de la sortie audio.

audioout→set_mute(**newval**)

Modifie l'état de la fonction silencieux (mute) de la sortie audio.

audioout→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`.

audioout→set_volume(**newval**)

Modifie le volume de la sortie audio, en pour cents.

audioout→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

audioout→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.

YAudioOut.FindAudioOut() YAudioOut yFindAudioOut()YAudioOut.FindAudioOut() YAudioOut.FindAudioOut()

Permet de retrouver une sortie audio d'après un identifiant donné.

function **FindAudioOut**(**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 sortie audio soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YAudioOut.isOnline()` pour tester si la sortie audio 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence la sortie audio sans ambiguïté

Retourne :

un objet de classe `YAudioOut` qui permet ensuite de contrôler la sortie audio.

YAudioOut.FindAudioOutInContext() YAudioOut
yFindAudioOutInContext()
YAudioOut.FindAudioOutInContext()
YAudioOut.FindAudioOutInContext()

Permet de retrouver une sortie audio d'après un identifiant donné dans un Context YAPI.

function **FindAudioOutInContext**(**yctx**, **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 sortie audio soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YAudioOut.isOnline()` pour tester si la sortie audio 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence la sortie audio sans ambiguïté

Retourne :

un objet de classe `YAudioOut` qui permet ensuite de contrôler la sortie audio.

YAudioOut.FirstAudioOut() YAudioOut yFirstAudioOut()YAudioOut.FirstAudioOut() YAudioOut.FirstAudioOut()

Commence l'énumération des sorties audio accessibles par la librairie.

function **FirstAudioOut**( )

Utiliser la fonction `YAudioOut.nextAudioOut()` pour itérer sur les autres sorties audio.

Retourne :

un pointeur sur un objet `YAudioOut`, correspondant à la première sortie audio accessible en ligne, ou `null` si il n'y a pas de sorties audio disponibles.

YAudioOut.FirstAudioOutInContext() YAudioOut
yFirstAudioOutInContext()
YAudioOut.FirstAudioOutInContext()
YAudioOut.FirstAudioOutInContext()

Commence l'énumération des sorties audio accessibles par la librairie.

function **FirstAudioOutInContext**(**yctx**)

Utiliser la fonction `YAudioOut.nextAudioOut()` pour itérer sur les autres sorties audio.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YAudioOut`, correspondant à la première sortie audio accessible en ligne, ou `null` si il n'y a pas de sorties audio disponibles.

audioout→clearCache()audioout.clearCache() YAudioOut

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de la sortie audio. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

audioout→describe()audioout.describe() YAudioOut

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

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant la sortie audio (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

audioout→get_advertisedValue() YAudioOut
audioout→advertisedValue()
audioout.get_advertisedValue()
audioout.get_advertisedValue()

Retourne la valeur courante de la sortie audio (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante de la sortie audio (pas plus de 6 caractères).

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

audioout→get_errorMessage() YAudioOut
audioout→errorMessage()
audioout.get_errorMessage()

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

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 sortie audio.

audioout→get_errorType() YAudioOut
audioout→errorType()audioout.get_errorType()

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

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 sortie audio.

audioout→get_friendlyName() YAudioOut
audioout→friendlyName()
audioout.get_friendlyName()

Retourne un identifiant global de la sortie audio au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

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

Retourne :

une chaîne de caractères identifiant la sortie audio en utilisant les noms logiques (ex: MyCustomName.relay1)

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

audioout→get_functionDescriptor() YAudioOut
audioout→functionDescriptor()
audioout.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

audioout→get_functionId() YAudioOut
audioout→functionId()audioout.get_functionId()

Retourne l'identifiant matériel de la sortie audio, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant la sortie audio (ex: `relay1`)

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

audioout→get_hardwareId() YAudioOut
audioout→hardwareId()audioout.get_hardwareId()

Retourne l'identifiant matériel unique de la sortie audio 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 sortie audio (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant la sortie audio (ex: RELAYLO1-123456.relay1)

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

audioout→get_logicalName() YAudioOut
audioout→logicalName()audioout.get_logicalName()
audioout.get_logicalName()

Retourne le nom logique de la sortie audio.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique de la sortie audio.

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

audioout→get_module() YAudioOut
audioout→module()audioout.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

audioout→get_mute() YAudioOut
audioout→mute()audioout.get_mute()
audioout.get_mute()

Retourne l'état de la fonction silencieux (mute) de la sortie audio.

function **get_mute**( )

Retourne :

soit Y_MUTE_FALSE, soit Y_MUTE_TRUE, selon l'état de la fonction silencieux (mute) de la sortie audio

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

audioout→get_noSignalFor() YAudioOut
audioout→noSignalFor()audioout.get_noSignalFor()
audioout.get_noSignalFor()

Retourne le nombre de secondes sans signal détecté.

function **get_noSignalFor**( )

Retourne :
un entier représentant le nombre de secondes sans signal détecté

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

audioout→get_signal() YAudioOut
audioout→signal()audioout.get_signal()
audioout.get_signal()

Retourne l'amplitude du courant émis détecté.

function **get_signal**( )

Retourne :
un entier représentant l'amplitude du courant émis détecté

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

audioout→get_userData() YAudioOut
audioout→userData()audioout.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.

audioout→get_volume() YAudioOut
audioout→volume()audioout.get_volume()
audioout.get_volume()

Retourne le volume de la sortie audio, en pour cents.

function **get_volume**( )

Retourne :
un entier représentant le volume de la sortie audio, en pour cents

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

audioout→get_volumeRange() YAudioOut
audioout→volumeRange()
audioout.get_volumeRange()
audioout.get_volumeRange()

Retourne la plage de valeurs supportées pour le volume.

function **get_volumeRange**( )

La valeur basse de l'intervalle correspond au volume minimal audible. Pour couper complètement le son, utilisez `set_mute()` et non le `set_volume()`.

Retourne :
une chaîne de caractères représentant la plage de valeurs supportées pour le volume

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

audioout→isOnline()audioout.isOnline() YAudioOut

Vérifie si le module hébergeant la sortie audio est joignable, sans déclencher d'erreur.

function **isOnline**( )

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

audioout→load()audioout.load() YAudioOut

Met en cache les valeurs courantes de la sortie audio, 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 occasionellement 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.

audioout→loadAttribute()audioout.loadAttribute() YAudioOut audioout.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

audioout→muteValueCallbacks() YAudioOut
audioout.muteValueCallbacks()
audioout.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

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

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

audioout→nextAudioOut()audioout.nextAudioOut() YAudioOut audioout.nextAudioOut()

Continue l'énumération des sorties audio commencée à l'aide de `yFirstAudioOut()`.

function **nextAudioOut**( )

Retourne :

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

audioout→registerValueCallback() YAudioOut
audioout.registerValueCallback()
audioout.registerValueCallback()

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.

audioout→set_logicalName() YAudioOut
audioout→setLogicalName()
audioout.set_logicalName()
audioout.set_logicalName()

Modifie le nom logique de la sortie audio.

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 sortie audio.

Retourne :

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

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

audioout→set_mute() YAudioOut
audioout→setMute()audioout.set_mute()
audioout.set_mute()

Modifie l'état de la fonction silencieux (mute) de la sortie audio.

function **set_mute**(**newval**)

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si vous désirez que le réglage soit préservé au prochain redémarrage.

Paramètres :

newval soit `Y_MUTE_FALSE`, soit `Y_MUTE_TRUE`, selon l'état de la fonction silencieux (mute) de la sortie audio

Retourne :

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

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

audioout→set_userData() YAudioOut
audioout→setUserData()audioout.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

audioout→set_volume() YAudioOut
audioout→setVolume()audioout.set_volume()
audioout.set_volume()

Modifie le volume de la sortie audio, en pour cents.

function **set_volume**(**newval**)

Paramètres :

newval un entier représentant le volume de la sortie audio, 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.

audioout→unmuteValueCallbacks() YAudioOut
audioout.unmuteValueCallbacks()
audioout.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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

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

audioout→wait_async()audioout.wait_async() YAudioOut

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 BluetoothLink

La fonction BluetoothLink permet de configurer et de contrôler une liaison bluetooth sur les modules Yoctopuce qui en sont dotés.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_bluetoothlink.js'></script>
cpp     #include "yocto_bluetoothlink.h"
m       #import "yocto_bluetoothlink.h"
pas     uses yocto_bluetoothlink;
vb      yocto_bluetoothlink.vb
cs      yocto_bluetoothlink.cs
java    import com.yoctopuce.YoctoAPI.YBluetoothLink;
uwp     import com.yoctopuce.YoctoAPI.YBluetoothLink;
py      from yocto_bluetoothlink import *
php     require_once('yocto_bluetoothlink.php');
es      in HTML: <script src="../../lib/yocto_bluetoothlink.js"></script>
        in node.js: require('yoctolib-es2017/yocto_bluetoothlink.js');
```

Fonction globales
yFindBluetoothLink(**func**) Permet de retrouver une interface cellulaire d'après un identifiant donné.
yFindBluetoothLinkInContext(**yctx**, **func**) Permet de retrouver une interface cellulaire d'après un identifiant donné dans un Context YAPI.
yFirstBluetoothLink() Commence l'énumération des interfaces réseau cellulaire accessibles par la librairie.
yFirstBluetoothLinkInContext(**yctx**) Commence l'énumération des interfaces réseau cellulaire accessibles par la librairie.
Méthodes des objets `YBluetoothLink`
bluetoothlink→clearCache() Invalide le cache.
bluetoothlink→connect() Etablit la connection bluetooth avec l'appareil distant préalablement choisi.
bluetoothlink→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface cellulaire au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
bluetoothlink→disconnect() Interrompt la connection bluetooth avec l'appareil distant.
bluetoothlink→get_advertisedValue() Retourne la valeur courante de l'interface cellulaire (pas plus de 6 caractères).
bluetoothlink→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface cellulaire.
bluetoothlink→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface cellulaire.
bluetoothlink→get_friendlyName() Retourne un identifiant global de l'interface cellulaire au format `NOM_MODULE.NOM_FONCTION`.
bluetoothlink→get_functionDescriptor()

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

bluetoothlink→get_functionId()

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

bluetoothlink→get_hardwareId()

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

bluetoothlink→get_linkQuality()

Retourne la qualité du signal bluetooth reçu, en pourcents, ou 0 si aucune connection n'est active.

bluetoothlink→get_linkState()

Retourne l'état du lien bluetooth reçu.

bluetoothlink→get_logicalName()

Retourne le nom logique de l'interface cellulaire.

bluetoothlink→get_module()

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

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

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

bluetoothlink→get_mute()

Retourne l'état de la fonction silencieux (mute) de la sortie audio.

bluetoothlink→get_ownAddress()

Retourne l'adresse MAC-48 de l'interface bluetooth, unique sur le réseau bluetooth.

bluetoothlink→get_pairingPin()

Retourne une string opaque si un code PIN a été configuré dans le module pour le pairing, ou une chaîne vide il n'a pas été configuré ou si la SIM a rejeté le code indiqué.

bluetoothlink→get_preAmplifier()

Retourne le volume du pré-amplificateur audio, en pour cents.

bluetoothlink→get_remoteAddress()

Retourne l'adresse MAC-48 bluetooth de l'appareil distant à connecter.

bluetoothlink→get_remoteName()

Retourne le nom bluetooth de l'appareil distant, si vu sur le réseau bluetooth.

bluetoothlink→get_userData()

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

bluetoothlink→get_volume()

Retourne le volume de l'écouteur connecté, en pour cents.

bluetoothlink→isOnline()

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

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

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

bluetoothlink→load(**msValidity**)

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

bluetoothlink→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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

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

bluetoothlink→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

bluetoothlink→nextBluetoothLink()

Continue l'énumération des interfaces réseau cellulaire commencée à l'aide de `yFirstBluetoothLink()`.

bluetoothlink→registerValueCallback(callback)

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

bluetoothlink→set_logicalName(newval)

Modifie le nom logique de l'interface cellulaire.

bluetoothlink→set_mute(newval)

Modifie l'état de la fonction silencieux (mute) de la sortie audio.

bluetoothlink→set_pairingPin(newval)

Modifie le code PIN utilisé par le module pour le pairing bluetooth.

bluetoothlink→set_preAmplifier(newval)

Modifie le volume du pré-amplificateur audio, en pour cents.

bluetoothlink→set_remoteAddress(newval)

Modifie l'adresse MAC-48 bluetooth définissant l'appareil distant à connecter.

bluetoothlink→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`.

bluetoothlink→set_volume(newval)

Modifie le volume de l'écouteur connecté, en pour cents.

bluetoothlink→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

bluetoothlink→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.

YBluetoothLink.FindBluetoothLink() YBluetoothLink
yFindBluetoothLink()
YBluetoothLink.FindBluetoothLink()
YBluetoothLink.FindBluetoothLink()

Permet de retrouver une interface cellulaire d'après un identifiant donné.

function **FindBluetoothLink**(**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 cellulaire soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YBluetoothLink.isOnline()` pour tester si l'interface cellulaire 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence l'interface cellulaire sans ambiguïté

Retourne :

un objet de classe `YBluetoothLink` qui permet ensuite de contrôler l'interface cellulaire.

YBluetoothLink.FindBluetoothLinkInContext() YBluetoothLink yFindBluetoothLinkInContext() YBluetoothLink.FindBluetoothLinkInContext() YBluetoothLink.FindBluetoothLinkInContext()

Permet de retrouver une interface cellulaire d'après un identifiant donné dans un Context YAPI.

function **FindBluetoothLinkInContext**(**yctx**, **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 cellulaire soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YBluetoothLink.isOnline()` pour tester si l'interface cellulaire 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'interface cellulaire sans ambiguïté

Retourne :

un objet de classe `YBluetoothLink` qui permet ensuite de contrôler l'interface cellulaire.

YBluetoothLink.FirstBluetoothLink() YBluetoothLink
yFirstBluetoothLink()
YBluetoothLink.FirstBluetoothLink()
YBluetoothLink.FirstBluetoothLink()

Commence l'énumération des interfaces réseau cellulaire accessibles par la librairie.

function **FirstBluetoothLink**( )

Utiliser la fonction `YBluetoothLink.nextBluetoothLink()` pour itérer sur les autres interfaces réseau cellulaire.

Retourne :

un pointeur sur un objet `YBluetoothLink`, correspondant à la première interface cellulaire accessible en ligne, ou `null` si il n'y a pas de interfaces réseau cellulaire disponibles.

YBluetoothLink.FirstBluetoothLinkInContext() YBluetoothLink yFirstBluetoothLinkInContext() YBluetoothLink.FirstBluetoothLinkInContext() YBluetoothLink.FirstBluetoothLinkInContext()

Commence l'énumération des interfaces réseau cellulaire accessibles par la librairie.

function **FirstBluetoothLinkInContext**(**yctx**)

Utiliser la fonction `YBluetoothLink.nextBluetoothLink()` pour itérer sur les autres interfaces réseau cellulaire.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YBluetoothLink`, correspondant à la première interface cellulaire accessible en ligne, ou `null` si il n'y a pas de interfaces réseau cellulaire disponibles.

bluetoothlink→clearCache()
bluetoothlink.clearCache()

YBluetoothLink

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'interface cellulaire. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

bluetoothlink→connect()bluetoothlink.connect() YBluetoothLink bluetoothlink.connect()

Etablit la connection bluetooth avec l'appareil distant préalablement choisi.

function **connect**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

bluetoothlink→describe()bluetoothlink.describe() YBluetoothLink

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

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant l'interface cellulaire (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

bluetoothlink→disconnect() YBluetoothLink
bluetoothlink.disconnect()bluetoothlink.disconnect()

Interrompt la connection bluetooth avec l'appareil distant.

function **disconnect**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

bluetoothlink→get_advertisedValue() YBluetoothLink
bluetoothlink→advertisedValue()
bluetoothlink.get_advertisedValue()
bluetoothlink.get_advertisedValue()

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

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante de l'interface cellulaire (pas plus de 6 caractères).

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

bluetoothlink→get_errorMessage() YBluetoothLink
bluetoothlink→errorMessage()
bluetoothlink.get_errorMessage()

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

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 cellulaire.

bluetoothlink→get_errorType() YBluetoothLink
bluetoothlink→errorType()
bluetoothlink.get_errorType()

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

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 cellulaire.

bluetoothlink→get_friendlyName() YBluetoothLink
bluetoothlink→friendlyName()
bluetoothlink.get_friendlyName()

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

function **get_friendlyName**( )

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

Retourne :

une chaîne de caractères identifiant l'interface cellulaire en utilisant les noms logiques (ex: `MyCustomName.relay1`)

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

bluetoothlink→get_functionDescriptor() YBluetoothLink
bluetoothlink→functionDescriptor()
bluetoothlink.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

bluetoothlink→get_functionId() YBluetoothLink
bluetoothlink→functionId()
bluetoothlink.get_functionId()

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

function **get_functionId**( )

Par example `relay1`.

Retourne :

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

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

bluetoothlink→get_hardwareId() YBluetoothLink
bluetoothlink→hardwareId()
bluetoothlink.get_hardwareId()

Retourne l'identifiant matériel unique de l'interface cellulaire 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 cellulaire (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant l'interface cellulaire (ex: RELAYLO1-123456.relay1)

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

bluetoothlink→get_linkQuality() YBluetoothLink
bluetoothlink→linkQuality()
bluetoothlink.get_linkQuality()
bluetoothlink.get_linkQuality()

Retourne la qualité du signal bluetooth reçu, en pourcents, ou 0 si aucune connection n'est active.

function **get_linkQuality**( )

Retourne :

un entier représentant la qualité du signal bluetooth reçu, en pourcents, ou 0 si aucune connection n'est active

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

bluetoothlink→get_linkState() YBluetoothLink
bluetoothlink→linkState()
bluetoothlink.get_linkState()
bluetoothlink.get_linkState()

Retourne l'état du lien bluetooth reçu.

function **get_linkState**( )

Retourne :

une valeur parmi `Y_LINKSTATE_DOWN`, `Y_LINKSTATE_FREE`, `Y_LINKSTATE_SEARCH`, `Y_LINKSTATE_EXISTS`, `Y_LINKSTATE_LINKED` et `Y_LINKSTATE_PLAY` représentant l'état du lien bluetooth reçu

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

bluetoothlink→get_logicalName() YBluetoothLink
bluetoothlink→logicalName()
bluetoothlink.get_logicalName()
bluetoothlink.get_logicalName()

Retourne le nom logique de l'interface cellulaire.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique de l'interface cellulaire.

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

bluetoothlink→get_module() YBluetoothLink
bluetoothlink→module()bluetoothlink.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

bluetoothlink→get_mute() YBluetoothLink
bluetoothlink→mute()bluetoothlink.get_mute()
bluetoothlink.get_mute()

Retourne l'état de la fonction silencieux (mute) de la sortie audio.

function **get_mute**( )

Retourne :

soit Y_MUTE_FALSE, soit Y_MUTE_TRUE, selon l'état de la fonction silencieux (mute) de la sortie audio

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

bluetoothlink→get_ownAddress() YBluetoothLink
bluetoothlink→ownAddress()
bluetoothlink.get_ownAddress()
bluetoothlink.get_ownAddress()

Retourne l'adresse MAC-48 de l'interface bluetooth, unique sur le réseau bluetooth.

function **get_ownAddress**( )

Retourne :
une chaîne de caractères représentant l'adresse MAC-48 de l'interface bluetooth, unique sur le réseau bluetooth

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

bluetoothlink→get_pairingPin() YBluetoothLink
bluetoothlink→pairingPin()
bluetoothlink.get_pairingPin()
bluetoothlink.get_pairingPin()

Retourne une string opaque si un code PIN a été configuré dans le module pour le pairing, ou une chaîne vide il n'a pas été configuré ou si la SIM a rejeté le code indiqué.

function **get_pairingPin**( )

Retourne :

une chaîne de caractères représentant une string opaque si un code PIN a été configuré dans le module pour le pairing, ou une chaîne vide il n'a pas été configuré ou si la SIM a rejeté le code indiqué

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

bluetoothlink→get_preAmplifier() YBluetoothLink
bluetoothlink→preAmplifier()
bluetoothlink.get_preAmplifier()
bluetoothlink.get_preAmplifier()

Retourne le volume du pré-amplificateur audio, en pour cents.

function **get_preAmplifier**( )

Retourne :
un entier représentant le volume du pré-amplificateur audio, en pour cents

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

bluetoothlink→get_remoteAddress() YBluetoothLink
bluetoothlink→remoteAddress()
bluetoothlink.get_remoteAddress()
bluetoothlink.get_remoteAddress()

Retourne l'adresse MAC-48 bluetooth de l'appareil distant à connecter.

function **get_remoteAddress**( )

Retourne :
une chaîne de caractères représentant l'adresse MAC-48 bluetooth de l'appareil distant à connecter

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

bluetoothlink→get_remoteName() YBluetoothLink
bluetoothlink→remoteName()
bluetoothlink.get_remoteName()
bluetoothlink.get_remoteName()

Retourne le nom bluetooth de l'appareil distant, si vu sur le réseau bluetooth.

function **get_remoteName**( )

Retourne :
une chaîne de caractères représentant le nom bluetooth de l'appareil distant, si vu sur le réseau bluetooth

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

bluetoothlink→get_userData() YBluetoothLink
bluetoothlink→userData()
bluetoothlink.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.

bluetoothlink→get_volume() YBluetoothLink
bluetoothlink→volume()bluetoothlink.get_volume()
bluetoothlink.get_volume()

Retourne le volume de l'écouteur connecté, en pour cents.

function **get_volume**( )

Retourne :
un entier représentant le volume de l'écouteur connecté, en pour cents

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

bluetoothlink→isOnline()bluetoothlink.isOnline() YBluetoothLink

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

function **isOnline**( )

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

bluetoothlink→load()bluetoothlink.load() YBluetoothLink

Met en cache les valeurs courantes de l'interface cellulaire, 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 occasionellement 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.

bluetoothlink→loadAttribute() YBluetoothLink
bluetoothlink.loadAttribute()
bluetoothlink.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

bluetoothlink→muteValueCallbacks() YBluetoothLink
bluetoothlink.muteValueCallbacks()
bluetoothlink.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

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

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

bluetoothlink→nextBluetoothLink() YBluetoothLink
bluetoothlink.nextBluetoothLink()
bluetoothlink.nextBluetoothLink()

Continue l'énumération des interfaces réseau cellulaire commencée à l'aide de `yFirstBluetoothLink()`.

function **nextBluetoothLink**( )

Retourne :

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

bluetoothlink→registerValueCallback() YBluetoothLink
bluetoothlink.registerValueCallback()
bluetoothlink.registerValueCallback()

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.

bluetoothlink→set_logicalName() YBluetoothLink
bluetoothlink→setLogicalName()
bluetoothlink.set_logicalName()
bluetoothlink.set_logicalName()

Modifie le nom logique de l'interface cellulaire.

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 cellulaire.

Retourne :

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

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

bluetoothlink→set_mute() YBluetoothLink
bluetoothlink→setMute()bluetoothlink.set_mute()
bluetoothlink.set_mute()

Modifie l'état de la fonction silencieux (mute) de la sortie audio.

function **set_mute**(**newval**)

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si vous désirez que le réglage soit préservé au prochain redémarrage.

Paramètres :

newval soit `Y_MUTE_FALSE`, soit `Y_MUTE_TRUE`, selon l'état de la fonction silencieux (mute) de la sortie audio

Retourne :

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

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

bluetoothlink→set_pairingPin() YBluetoothLink
bluetoothlink→setPairingPin()
bluetoothlink.set_pairingPin()
bluetoothlink.set_pairingPin()

Modifie le code PIN utilisé par le module pour le pairing bluetooth.

function **set_pairingPin**(**newval**)

N'oubliez pas d'appeler la méthode `saveToFlash()` du module pour que le paramètre soit sauvegardé dans la flash.

Paramètres :

newval une chaîne de caractères représentant le code PIN utilisé par le module pour le pairing bluetooth

Retourne :

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

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

bluetoothlink→set_preAmplifier() YBluetoothLink
bluetoothlink→setPreAmplifier()
bluetoothlink.set_preAmplifier()
bluetoothlink.set_preAmplifier()

Modifie le volume du pré-amplificateur audio, en pour cents.

function **set_preAmplifier**(**newval**)

Paramètres :

newval un entier représentant le volume du pré-amplificateur audio, 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.

bluetoothlink→set_remoteAddress() YBluetoothLink
bluetoothlink→setRemoteAddress()
bluetoothlink.set_remoteAddress()
bluetoothlink.set_remoteAddress()

Modifie l'adresse MAC-48 bluetooth définissant l'appareil distant à connecter.

function **set_remoteAddress**(**newval**)

Paramètres :

newval une chaîne de caractères représentant l'adresse MAC-48 bluetooth définissant l'appareil distant à connecter

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

bluetoothlink→set_userData() YBluetoothLink
bluetoothlink→setUserData()
bluetoothlink.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

bluetoothlink→set_volume() YBluetoothLink
bluetoothlink→setVolume()
bluetoothlink.set_volume()
bluetoothlink.set_volume()

Modifie le volume de l'écouteur connecté, en pour cents.

```
function set_volume( newval)
```

Paramètres :

newval un entier représentant le volume de l'écouteur connecté, 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.

bluetoothlink→unmuteValueCallbacks() YBluetoothLink
bluetoothlink.unmuteValueCallbacks()
bluetoothlink.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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

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

bluetoothlink→wait_async() YBluetoothLink
bluetoothlink.wait_async()

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 Buzzer

La librairie de programmation Yoctopuce permet de choisir la fréquence et le volume à laquelle le buzzer doit sonner. Il est aussi possible de pré-programmer une séquence à jouer.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_buzzer.js'></script>
cpp	#include "yocto_buzzer.h"
m	#import "yocto_buzzer.h"
pas	uses yocto_buzzer;
vb	yocto_buzzer.vb
cs	yocto_buzzer.cs
java	import com.yoctopuce.YoctoAPI.YBuzzer;
uwp	import com.yoctopuce.YoctoAPI.YBuzzer;
py	from yocto_buzzer import *
php	require_once('yocto_buzzer.php');
es	in HTML: <script src="../../lib/yocto_buzzer.js"></script> in node.js: require('yoctolib-es2017/yocto_buzzer.js');

Fonction globales

yFindBuzzer(**func**)

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

yFindBuzzerInContext(**yctx**, **func**)

Permet de retrouver un buzzer d'après un identifiant donné dans un Context YAPI.

yFirstBuzzer()

Commence l'énumération des buzzer accessibles par la librairie.

yFirstBuzzerInContext(**yctx**)

Commence l'énumération des buzzer accessibles par la librairie.

Méthodes des objets YBuzzer

buzzer→addFreqMoveToPlaySeq(**freq**, **msDelay**)

Ajoute une transition en fréquence à la séquence à jouer.

buzzer→addNotesToPlaySeq(**notes**)

Ajoute des notes à la séquence à jouer.

buzzer→addPulseToPlaySeq(**freq**, **msDuration**)

Ajoute une impulsion à la séquence à jouer.

buzzer→addVolMoveToPlaySeq(**volume**, **msDuration**)

Ajoute une transition en volume à la séquence à jouer.

buzzer→clearCache()

Invalide le cache.

buzzer→describe()

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

buzzer→freqMove(**frequency**, **duration**)

Fait varier la fréquence du buzzer pendant un temps donné.

buzzer→get_advertisedValue()

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

buzzer→get_errorMessage()

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

buzzer→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du buzzer.

buzzer→get_frequency()
Retourne la fréquence du signal envoyé sur le buzzer/haut parleur.

buzzer→get_friendlyName()
Retourne un identifiant global du buzzer au format `NOM_MODULE.NOM_FONCTION`.

buzzer→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

buzzer→get_functionId()
Retourne l'identifiant matériel du buzzer, sans référence au module.

buzzer→get_hardwareId()
Retourne l'identifiant matériel unique du buzzer au format `SERIAL.FUNCTIONID`.

buzzer→get_logicalName()
Retourne le nom logique du buzzer.

buzzer→get_module()
Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

buzzer→get_module_async(**callback**, **context**)
Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

buzzer→get_playSeqMaxSize()
Retourne la longueur maximum de la séquence à jouer.

buzzer→get_playSeqSignature()
Retourne la signature de la signature de la séquence à jouer.

buzzer→get_playSeqSize()
Retourne la longueur actuelle de la séquence à jouer.

buzzer→get_userData()
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

buzzer→get_volume()
Retourne le volume du signal envoyé sur le buzzer/haut parleur.

buzzer→isOnline()
Vérifie si le module hébergeant le buzzer est joignable, sans déclencher d'erreur.

buzzer→isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant le buzzer est joignable, sans déclencher d'erreur.

buzzer→load(**msValidity**)
Met en cache les valeurs courantes du buzzer, avec une durée de validité spécifiée.

buzzer→loadAttribute(**attrName**)
Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

buzzer→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes du buzzer, avec une durée de validité spécifiée.

buzzer→muteValueCallbacks()
Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

buzzer→nextBuzzer()
Continue l'énumération des buzzer commencée à l'aide de `yFirstBuzzer()`.

buzzer→oncePlaySeq()
Démarre l'exécution de la séquence à jouer préprogrammée, pour une seule exécution.

buzzer→playNotes(notes)

Joue immédiatement une séquence de notes.

buzzer→pulse(frequency, duration)

Active le buzzer pendant un temps donné.

buzzer→registerValueCallback(callback)

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

buzzer→resetPlaySeq()

efface le contenu de la sequence à jouer et mets la fréquence à zéro.

buzzer→set_frequency(newval)

Modifie la fréquence du signal envoyé sur le buzzer/speaker.

buzzer→set_logicalName(newval)

Modifie le nom logique du buzzer.

buzzer→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`.

buzzer→set_volume(newval)

Modifie le volume du signal envoyé sur le buzzer/haut parleur.

buzzer→startPlaySeq()

Démarre l'exécution de la séquence à jouer préprogrammée.

buzzer→stopPlaySeq()

Arrête l'exécution de la séquence à jouer préprogrammée et mets la fréquence à zéro.

buzzer→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

buzzer→volumeMove(volume, duration)

Fait varier le volume du buzzer pendant un temps donné, la fréquence reste inchangée.

buzzer→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.

YBuzzer.FindBuzzer() YBuzzer yFindBuzzer()YBuzzer.FindBuzzer() YBuzzer.FindBuzzer()

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

```
function FindBuzzer( 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 buzzer soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YBuzzer.isOnline()` pour tester si le buzzer 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le buzzer sans ambiguïté

Retourne :

un objet de classe `YBuzzer` qui permet ensuite de contrôler le buzzer.

YBuzzer.FindBuzzerInContext() YBuzzer
yFindBuzzerInContext()
YBuzzer.FindBuzzerInContext()
YBuzzer.FindBuzzerInContext()

Permet de retrouver un buzzer d'après un identifiant donné dans un Context YAPI.

function **FindBuzzerInContext**(**yctx**, **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 buzzer soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YBuzzer.isOnline()` pour tester si le buzzer 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le buzzer sans ambiguïté

Retourne :

un objet de classe `YBuzzer` qui permet ensuite de contrôler le buzzer.

YBuzzer.FirstBuzzer() YBuzzer yFirstBuzzer()YBuzzer.FirstBuzzer() YBuzzer.FirstBuzzer()

Commence l'énumération des buzzer accessibles par la librairie.

function **FirstBuzzer**( )

Utiliser la fonction `YBuzzer.nextBuzzer()` pour itérer sur les autres buzzer.

Retourne :

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

YBuzzer.FirstBuzzerInContext() YBuzzer
yFirstBuzzerInContext()
YBuzzer.FirstBuzzerInContext()
YBuzzer.FirstBuzzerInContext()

Commence l'énumération des buzzer accessibles par la librairie.

function **FirstBuzzerInContext**(**yctx**)

Utiliser la fonction `YBuzzer.nextBuzzer()` pour itérer sur les autres buzzer.

Paramètres :

yctx un contexte YAPI.

Retourne :

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

buzzer→addFreqMoveToPlaySeq() YBuzzer
buzzer.addFreqMoveToPlaySeq()
buzzer.addFreqMoveToPlaySeq()

Ajoute une transition en fréquence à la séquence à jouer.

function **addFreqMoveToPlaySeq**(**freq**, **msDelay**)

Paramètres :

freq	fréquence désirée à la fin de la transition, en Hz
msDelay	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.

buzzer→addNotesToPlaySeq() YBuzzer
buzzer.addNotesToPlaySeq()
buzzer.addNotesToPlaySeq()

Ajoute des notes à la séquence à jouer.

function **addNotesToPlaySeq**(**notes**)

Les notes sont spécifiées sous forme textuelle, séparées par des espaces. La hauteur est donnée par une lettre de A à G, selon la tradition anglo-saxonne. La durée est donnée par un chiffre, indiquant le diviseur par rapport à une ronde: 4 pour une noire, 8 pour une croche, etc. Quelques modifieurs sont supportés: le # et le b pour altérer la hauteur des notes, le `'` et la `,` pour octavier vers le haut ou vers le bas, le `.` pour allonger la durée.

Paramètres :

notes notes à ajouter, 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.

buzzer→addPulseToPlaySeq() YBuzzer
buzzer.addPulseToPlaySeq()
buzzer.addPulseToPlaySeq()

Ajoute une impulsion à la séquence à jouer.

function **addPulseToPlaySeq**(**freq**, **msDuration**)

Paramètres :

freq	fréquence de l'impulsion en Hz
msDuration	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.

buzzer→addVolMoveToPlaySeq() YBuzzer
buzzer.addVolMoveToPlaySeq()
buzzer.addVolMoveToPlaySeq()

Ajoute une transition en volume à la séquence à jouer.

function **addVolMoveToPlaySeq**(**volume**, **msDuration**)

La fréquence courante reste inchangée: si elle est à zéro, la transition n'aura aucun effet.

Paramètres :

volume	volume désiré à la fin de la transition, en pourcentage.
msDuration	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.

buzzer→clearCache()buzzer.clearCache() YBuzzer

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du buzzer. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

buzzer→describe()buzzer.describe() YBuzzer

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

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le buzzer (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

buzzer→freqMove()buzzer.freqMove() YBuzzer buzzer.freqMove()

Fait varier la fréquence du buzzer pendant un temps donné.

```
function freqMove( frequency, duration)
```

Paramètres :

frequency fréquence à atteindre en hertz, une fréquence inférieure à 25Hz arrêtera le buzzer.

duration durée de l'impulsion en milliseconds

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

buzzer→get_advertisedValue() YBuzzer
buzzer→advertisedValue()
buzzer.get_advertisedValue()
buzzer.get_advertisedValue()

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

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante du buzzer (pas plus de 6 caractères).

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

buzzer→get_errorMessage() YBuzzer
buzzer→errorMessage()buzzer.get_errorMessage()

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

```
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 buzzer.

buzzer→get_errorType() YBuzzer
buzzer→errorType()buzzer.get_errorType()

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

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 buzzer.

buzzer→get_frequency() YBuzzer
buzzer→frequency()buzzer.get_frequency()
buzzer.get_frequency()

Retourne la fréquence du signal envoyé sur le buzzer/haut parleur.

function **get_frequency**( )

Retourne :
une valeur numérique représentant la fréquence du signal envoyé sur le buzzer/haut parleur

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

buzzer→get_friendlyName() YBuzzer
buzzer→friendlyName()buzzer.get_friendlyName()

Retourne un identifiant global du buzzer au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

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

Retourne :
une chaîne de caractères identifiant le buzzer en utilisant les noms logiques (ex: MyCustomName.relay1)

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

buzzer→get_functionDescriptor() YBuzzer
buzzer→functionDescriptor()
buzzer.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

buzzer→get_functionId() YBuzzer
buzzer→functionId()buzzer.get_functionId()

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

function **get_functionId**( )

Par example `relay1`.

Retourne :

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

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

buzzer→get_hardwareId() YBuzzer
buzzer→hardwareId()buzzer.get_hardwareId()

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

function **get_hardwareId**( )

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

Retourne :

une chaîne de caractères identifiant le buzzer (ex: RELAYLO1-123456.relay1)

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

buzzer→get_logicalName() YBuzzer
buzzer→logicalName()buzzer.get_logicalName()
buzzer.get_logicalName()

Retourne le nom logique du buzzer.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du buzzer.

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

buzzer→get_module() YBuzzer
buzzer→module()buzzer.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`

buzzer→get_playSeqMaxSize() YBuzzer
buzzer→playSeqMaxSize()
buzzer.get_playSeqMaxSize()
buzzer.get_playSeqMaxSize()

Retourne la longueur maximum de la séquence à jouer.

function **get_playSeqMaxSize**( )

Retourne :
un entier représentant la longueur maximum de la séquence à jouer

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

buzzer→get_playSeqSignature() YBuzzer
buzzer→playSeqSignature()
buzzer.get_playSeqSignature()
buzzer.get_playSeqSignature()

Retourne la signature de la signature de la séquence à jouer.

function **get_playSeqSignature**( )

Les séquences à jouer ne pouvant pas être relues depuis le module, ce mécanisme peut être utilisé pour détecter si une séquence spécifique est déjà programmée.

Retourne :
un entier représentant la signature de la signature de la séquence à jouer

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

buzzer→get_playSeqSize() YBuzzer
buzzer→playSeqSize()buzzer.get_playSeqSize()
buzzer.get_playSeqSize()

Retourne la longueur actuelle de la séquence à jouer.

function **get_playSeqSize**( )

Retourne :
un entier représentant la longueur actuelle de la séquence à jouer

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

buzzer→get_userData() YBuzzer
buzzer→userData()buzzer.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.

buzzer→get_volume() YBuzzer
buzzer→volume()buzzer.get_volume()
buzzer.get_volume()

Retourne le volume du signal envoyé sur le buzzer/haut parleur.

function **get_volume**( )

Retourne :
un entier représentant le volume du signal envoyé sur le buzzer/haut parleur

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

buzzer→isOnline()buzzer.isOnline() YBuzzer

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

function **isOnline**( )

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

buzzer→load()buzzer.load() YBuzzer

Met en cache les valeurs courantes du buzzer, 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 occasionellement 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.

buzzer→loadAttribute()buzzer.loadAttribute() YBuzzer buzzer.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

buzzer→muteValueCallbacks() YBuzzer
buzzer.muteValueCallbacks()
buzzer.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

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

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

buzzer→nextBuzzer()buzzer.nextBuzzer() YBuzzer buzzer.nextBuzzer()

Continue l'énumération des buzzer commencée à l'aide de `yFirstBuzzer()`.

function **nextBuzzer**( )

Retourne :

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

buzzer→oncePlaySeq()buzzer.oncePlaySeq() YBuzzer buzzer.oncePlaySeq()

Démarre l'exécution de la séquence à jouer préprogrammée, pour une seule exécution.

function **oncePlaySeq**( )

Retourne :

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

buzzer→playNotes()buzzer.playNotes() YBuzzer buzzer.playNotes()

Joue immédiatement une séquence de notes.

function **playNotes**(**notes**)

Les notes sont spécifiées sous forme textuelle, séparées par des espaces. La hauteur est donnée par une lettre de A à G, selon la tradition anglo-saxonne. La durée est donnée par un chiffre, indiquant le diviseur par rapport à une ronde: 4 pour une noire, 8 pour une croche, etc. Quelques modifieurs sont supportés: le # et le b pour altérer la hauteur des notes, le `'` et la `,` pour octavier vers le haut ou vers le bas, le `.` pour allonger la durée.

Paramètres :

notes notes à ajouter, 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.

buzzer→pulse()buzzer.pulse()buzzer.pulse() YBuzzer

Active le buzzer pendant un temps donné.

function **pulse**(**frequency**, **duration**)

Paramètres :

frequency fréquence de l'impulsion, en hertz

duration durée de l'impulsion en milliseconds

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

buzzer→registerValueCallback() YBuzzer
buzzer.registerValueCallback()
buzzer.registerValueCallback()

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.

buzzer→resetPlaySeq()buzzer.resetPlaySeq() YBuzzer buzzer.resetPlaySeq()

efface le contenu de la sequence à jouer et mets la fréquence à zéro.

function **resetPlaySeq**( )

Retourne :

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

buzzer→set_frequency() YBuzzer
buzzer→setFrequency()buzzer.set_frequency()
buzzer.set_frequency()

Modifie la fréquence du signal envoyé sur le buzzer/speaker.

function **set_frequency**(**newval**)

Une fréquence nulle stoppe le buzzer.

Paramètres :

newval une valeur numérique représentant la fréquence du signal envoyé sur le buzzer/speaker

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

buzzer→set_logicalName() YBuzzer
buzzer→setLogicalName()buzzer.set_logicalName()
buzzer.set_logicalName()

Modifie le nom logique du buzzer.

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 buzzer.

Retourne :

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

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

buzzer→set_userData() YBuzzer
buzzer→setUserData()buzzer.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

buzzer→set_volume() YBuzzer
buzzer→setVolume()buzzer.set_volume()
buzzer.set_volume()

Modifie le volume du signal envoyé sur le buzzer/haut parleur.

function **set_volume**(**newval**)

Paramètres :

newval un entier représentant le volume du signal envoyé sur le buzzer/haut parleur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

buzzer→startPlaySeq()buzzer.startPlaySeq() YBuzzer buzzer.startPlaySeq()

Démarre l'exécution de la séquence à jouer préprogrammée.

function **startPlaySeq**( )

La séquence va tourner en boucle jusqu'à ce qu'elle soit stoppée par stopPlaySeq ou un changement explicite. Pour ne jouer qu'un seule fois la séquence, utiliser la méthode `oncePlaySeq()`.

Retourne :

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

buzzer→stopPlaySeq()buzzer.stopPlaySeq() YBuzzer buzzer.stopPlaySeq()

Arrête l'exécution de la séquence à jouer préprogrammée et mets la fréquence à zéro.

function **stopPlaySeq**( )

Retourne :

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

buzzer→unmuteValueCallbacks() YBuzzer
buzzer.unmuteValueCallbacks()
buzzer.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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

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

buzzer→volumeMove()buzzer.volumeMove() YBuzzer buzzer.volumeMove()

Fait varier le volume du buzzer pendant un temps donné, la fréquence reste inchangée.

function **volumeMove**(**volume**, **duration**)

Paramètres :

volume volume à atteindre en %.

duration durée de la transition en milliseconds

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

buzzer→wait_async()buzzer.wait_async() YBuzzer

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. Interface de la fonction CarbonDioxide

La classe YCarbonDioxide permet de lire et de configurer les capteurs de CO2 Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, elle permet d'effectuer des calibrations manuelles si nécessaire.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_carbondioxide.js'></script>
cpp     #include "yocto_carbondioxide.h"
m       #import "yocto_carbondioxide.h"
pas     uses yocto_carbondioxide;
vb      yocto_carbondioxide.vb
cs      yocto_carbondioxide.cs
java    import com.yoctopuce.YoctoAPI.YCarbonDioxide;
uwp     import com.yoctopuce.YoctoAPI.YCarbonDioxide;
py      from yocto_carbondioxide import *
php     require_once('yocto_carbondioxide.php');
es      in HTML: <script src="../../lib/yocto_carbondioxide.js"></script>
        in node.js: require('yoctolib-es2017/yocto_carbondioxide.js');
```

Fonction globales
yFindCarbonDioxide(**func**) Permet de retrouver un capteur de CO2 d'après un identifiant donné.
yFindCarbonDioxideInContext(**yctx**, **func**) Permet de retrouver un capteur de CO2 d'après un identifiant donné dans un Context YAPI.
yFirstCarbonDioxide() Commence l'énumération des capteurs de CO2 accessibles par la librairie.
yFirstCarbonDioxideInContext(**yctx**) 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→clearCache() Invalide le cache.
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_abcPeriod() Retourne la durée de la période pour la correction automatique de référence (auto-calibration) du capteur, exprimée en heures.
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), en ppm (val), sous forme de nombre à virgule.
carbondioxide→get_currentValue() Retourne la valeur actuelle du taux de CO2, en ppm (val), sous forme de nombre à virgule.

carbondioxide→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

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_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

carbondioxide→load(**msValidity**)

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

carbondioxide→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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_abcPeriod(**newval**)

Change la durée de la période pour la correction automatique de référence (auto-calibration) du capteur, exprimée en heures.

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.

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→startDataLogger()

Démarre l'enregistreur de données du module.

carbondioxide→stopDataLogger()

Arrête l'enregistreur de données du module.

carbondioxide→triggerBaselineCalibration()

Lance une calibration au niveau de CO2 ambiant standard (400ppm).

carbondioxide→triggerZeroCalibration()

Lance une calibration au niveau zéro (air exempt de CO2).

carbondioxide→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

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

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

function **FindCarbonDioxide**(**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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindCarbonDioxideInContext() YCarbonDioxide yFindCarbonDioxideInContext() YCarbonDioxide.FindCarbonDioxideInContext() YCarbonDioxide.FindCarbonDioxideInContext()

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

function **FindCarbonDioxideInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

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

Retourne :

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

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

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

function **FirstCarbonDioxide**( )

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.

YCarbonDioxide.FirstCarbonDioxideInContext() YCarbonDioxide yFirstCarbonDioxideInContext() YCarbonDioxide.FirstCarbonDioxideInContext() YCarbonDioxide.FirstCarbonDioxideInContext()

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

function **FirstCarbonDioxideInContext**(**yctx**)

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

Paramètres :

yctx un contexte YAPI.

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() YCarbonDioxide
carbondioxide.calibrateFromPoints()
carbondioxide.calibrateFromPoints()

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

function **calibrateFromPoints**(**rawValues**, **refValues**)

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

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

Paramètres :

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

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

Retourne :

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

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

carbondioxide→clearCache() YCarbonDioxide
carbondioxide.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur de CO2. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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

carbondioxide→get_abcPeriod() YCarbonDioxide
carbondioxide→abcPeriod()
carbondioxide.get_abcPeriod()
carbondioxide.get_abcPeriod()

Retourne la durée de la période pour la correction automatique de référence (auto-calibration) du capteur, exprimée en heures.

function **get_abcPeriod**( )

Une valeur négative signifie que la correction automatique de référence est désactivée.

Retourne :
un entier représentant la durée de la période pour la correction automatique de référence (auto-calibration) du capteur, exprimée en heures

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

carbondioxide→get_advertisedValue() YCarbonDioxide
carbondioxide→advertisedValue()
carbondioxide.get_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()
carbondioxide.get_currentRawValue()

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

function **get_currentRawValue**( )

Retourne :

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

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

carbondioxide→get_currentValue() YCarbonDioxide
carbondioxide→currentValue()
carbondioxide.get_currentValue()
carbondioxide.get_currentValue()

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

function **get_currentValue**( )

Retourne :

une valeur numérique représentant la valeur actuelle du taux de CO2, en ppm (val), sous forme de nombre à virgule

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

carbondioxide→get_dataLogger() YCarbonDioxide
carbondioxide→dataLogger()
carbondioxide.get_dataLogger()
carbondioxide.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

carbondioxide→get_errorMessage() YCarbonDioxide
carbondioxide→errorMessage()
carbondioxide.get_errorMessage()

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

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() YCarbonDioxide
carbondioxide→friendlyName()
carbondioxide.get_friendlyName()

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'identifant matériel du capteur de CO2 (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le capteur de CO2 en utilisant les noms logiques (ex: MyCustomName.relay1)

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

carbondioxide→get_functionDescriptor() YCarbonDioxide
carbondioxide→functionDescriptor()
carbondioxide.get_functionDescriptor()

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

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

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

function **get_functionId**( )

Par example `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() YCarbonDioxide
carbondioxide→hardwareId()
carbondioxide.get_hardwareId()

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 example 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() YCarbonDioxide
carbondioxide→highestValue()
carbondioxide.get_highestValue()
carbondioxide.get_highestValue()

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

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()
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() YCarbonDioxide
carbondioxide→module()
carbondioxide.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

carbondioxide→get_recordedData() YCarbonDioxide
carbondioxide→recordedData()
carbondioxide.get_recordedData()
carbondioxide.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 find de l'intercalle 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()
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() YCarbonDioxide
carbondioxide→resolution()
carbondioxide.get_resolution()
carbondioxide.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`.

carbondioxide→get_sensorState() YCarbonDioxide
carbondioxide→sensorState()
carbondioxide.get_sensorState()
carbondioxide.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :
un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

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

carbondioxide→get_unit() YCarbonDioxide
carbondioxide→unit()carbondioxide.get_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() YCarbonDioxide
carbondioxide→userData()
carbondioxide.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.

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→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 occasionellement 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→loadAttribute() YCarbonDioxide
carbondioxide.loadAttribute()
carbondioxide.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

carbondioxide→loadCalibrationPoints() YCarbonDioxide
carbondioxide.loadCalibrationPoints()
carbondioxide.loadCalibrationPoints()

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

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

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

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

Retourne :

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

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

carbondioxide→muteValueCallbacks() YCarbonDioxide
carbondioxide.muteValueCallbacks()
carbondioxide.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

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

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

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

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() YCarbonDioxide
carbondioxide.registerTimedReportCallback()
carbondioxide.registerTimedReportCallback()

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.

carbondioxide→registerValueCallback() YCarbonDioxide
carbondioxide.registerValueCallback()
carbondioxide.registerValueCallback()

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_abcPeriod() YCarbonDioxide
carbondioxide→setAbcPeriod()
carbondioxide.set_abcPeriod()
carbondioxide.set_abcPeriod()

Change la durée de la période pour la correction automatique de référence (auto-calibration) du capteur, exprimée en heures.

function **set_abcPeriod**(**newval**)

Pour désactivez la correction automatique de référence (par exemple lorsque le capteur est utilisé dans un environement constamment au dessus de 400ppm CO2), configurez la période à la valeur -1. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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.

carbondioxide→set_highestValue() YCarbonDioxide
carbondioxide→setHighestValue()
carbondioxide.set_highestValue()
carbondioxide.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.

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

carbondioxide→set_logicalName() YCarbonDioxide
carbondioxide→setLogicalName()
carbondioxide.set_logicalName()
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() YCarbonDioxide
carbondioxide→setLowestValue()
carbondioxide.set_lowestValue()
carbondioxide.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.

carbondioxide→set_reportFrequency() YCarbonDioxide
carbondioxide→setReportFrequency()
carbondioxide.set_reportFrequency()
carbondioxide.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.

carbondioxide→set_resolution() YCarbonDioxide
carbondioxide→setResolution()
carbondioxide.set_resolution()
carbondioxide.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.

carbondioxide→set_userData() YCarbonDioxide
carbondioxide→setUserData()
carbondioxide.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

carbondioxide→startDataLogger() YCarbonDioxide
carbondioxide.startDataLogger()
carbondioxide.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

carbondioxide→stopDataLogger() YCarbonDioxide
carbondioxide.stopDataLogger()
carbondioxide.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

carbondioxide→triggerBaselineCalibration() YCarbonDioxide
carbondioxide.triggerBaselineCalibration()
carbondioxide.triggerBaselineCalibration()

Lance une calibration au niveau de CO2 ambiant standard (400ppm).

function **triggerBaselineCalibration**( )

Il n'est normalement pas nécessaire de calibrer le capteur, car is est conçu pour compenser automatiquement toute dérive sur le long terme en se basant sur la plus faible valeur observée durant la période de calibration automatique. Toutefois, si vous désactivez la calibration automatique, vous pouvez lancer manuellement cette calibration ambiante en prenant soit de placer préalablement le capteur dans un environnement standard (par exemple à l'extérieur) à 400ppm.

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→triggerZeroCalibration() YCarbonDioxide
carbondioxide.triggerZeroCalibration()
carbondioxide.triggerZeroCalibration()

Lance une calibration au niveau zéro (air exempt de CO2).

function **triggerZeroCalibration**( )

Il n'est normalement pas nécessaire de calibrer le capteur, car is est conçu pour compenser automatiquement toute dérive sur le long terme en se basant sur la plus faible valeur observée durant la période de calibration automatique. Toutefois, si vous désactivez la calibration automatique, vous pouvez lancer manuellement cette calibration au niveau zéro après avoir fait circuler dans le capteur pendant une minute ou deux de l'air exempt de CO2, à l'aide d'un petit tube branché sur le capteur. Contactez support@yoctopuce.com pour plus de détail sur la procédure de calibration au niveau zéro.

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

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

carbondioxide→unmuteValueCallbacks() YCarbonDioxide
carbondioxide.unmuteValueCallbacks()
carbondioxide.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→wait_async() YCarbonDioxide
carbondioxide.wait_async()

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.

3.10. Interface de la fonction Cellular

La fonction YCellular permet de configurer et de contrôler la configuration du réseau cellulaire sur les modules Yoctopuce qui en sont dotés.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_cellular.js'></script>
cpp     #include "yocto_cellular.h"
m       #import "yocto_cellular.h"
pas     uses yocto_cellular;
vb      yocto_cellular.vb
cs      yocto_cellular.cs
java    import com.yoctopuce.YoctoAPI.YCellular;
uwp     import com.yoctopuce.YoctoAPI.YCellular;
py      from yocto_cellular import *
php     require_once('yocto_cellular.php');
es      in HTML: <script src="../../lib/yocto_cellular.js"></script>
        in node.js: require('yoctolib-es2017/yocto_cellular.js');
```

Fonction globales
yFindCellular(**func**) Permet de retrouver une interface cellulaire d'après un identifiant donné.
yFindCellularInContext(**yctx**, **func**) Permet de retrouver une interface cellulaire d'après un identifiant donné dans un Context YAPI.
yFirstCellular() Commence l'énumération des interfaces réseau cellulaire accessibles par la librairie.
yFirstCellularInContext(**yctx**) Commence l'énumération des interfaces réseau cellulaire accessibles par la librairie.
y_AT(**cmd**) Envoie une commande AT au module GSM, et retourne le résultat.
Méthodes des objets `YCellular`
cellular→clearCache() Invalide le cache.
cellular→clearDataCounters() Réinitialise les compteurs de données transmises et reçues.
cellular→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface cellulaire au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
cellular→get_advertisedValue() Retourne la valeur courante de l'interface cellulaire (pas plus de 6 caractères).
cellular→get_airplaneMode() Retourne vrai si le mode avion est activé (radio désactivée).
cellular→get_apn() Retourne le nom du point d'accès (APN) à utiliser, si nécessaire.
cellular→get_apnSecret() Retourne une string opaque si des paramètres d'identification sur l'APN ont été configurés dans le module, ou une chaîne vide autrement.
cellular→get_availableOperators()

Retourne la liste des opérateurs GSM disponibles à proximité.

cellular→get_cellIdentifier()

Retourne l'identifiant unique de la station de base utilisée: MCC, MNC, LAC et Cell ID.

cellular→get_cellOperator()

Retourne le nom de l'opérateur de réseau cellulaire actuellement utilisé.

cellular→get_cellType()

Type de connection cellulaire active.

cellular→get_dataReceived()

Retourne le nombre d'octets reçus jusqu'à présent.

cellular→get_dataSent()

Retourne le nombre d'octets envoyés jusqu'à présent.

cellular→get_enableData()

Retourne la condition dans laquelle le service de données IP (GRPS) doit être activé.

cellular→get_errorMessage()

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

cellular→get_errorType()

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

cellular→get_friendlyName()

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

cellular→get_functionDescriptor()

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

cellular→get_functionId()

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

cellular→get_hardwareId()

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

cellular→get_imsi()

Retourne une string opaque si un code PIN a été configuré dans le module pour accéder à la carte SIM, ou une chaîne vide il n'a pas été configuré ou si la SIM a rejeté le code indiqué.

cellular→get_linkQuality()

Retourne la qualité de la connection, exprimée en pourcents.

cellular→get_lockedOperator()

Retourne le nom de l'opérateur de réseau cellulaire à utiliser exclusivement, si le choix automatique est désactivé, ou une chaîne vide si la carte SIM sélectionne automatiquement l'opérateur selon ceux disponibles.

cellular→get_logicalName()

Retourne le nom logique de l'interface cellulaire.

cellular→get_message()

Retourne le dernier message de diagnostique de l'interface au réseau sans fil.

cellular→get_module()

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

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

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

cellular→get_pin()

Retourne une string opaque si un code PIN a été configuré dans le module pour accéder à la carte SIM, ou une chaîne vide il n'a pas été configuré ou si la SIM a rejeté le code indiqué.

cellular→get_pingInterval()

Retourne l'intervalle entre les tests de connectivité spontanés, en secondes.

cellular→get_userData()

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

cellular→isOnline()

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

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

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

cellular→load(**msValidity**)

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

cellular→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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

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

cellular→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

cellular→nextCellular()

Continue l'énumération des interfaces réseau cellulaire commencée à l'aide de `yFirstCellular()`.

cellular→quickCellSurvey()

Retourne la liste d'identifiants pour les antennes GSM à proximité, telle que requise pour géolocaliser rapidement le module.

cellular→registerValueCallback(**callback**)

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

cellular→sendPUK(**puk**, **newPin**)

Envoie le code PUK à la carte SIM pour la débloquer après trois échecs consécutifs de code PIN, et établit un nouveau code PIN dans la SIM.

cellular→set_airplaneMode(**newval**)

Modifie l'état du mode avion (radio désactivée).

cellular→set_apn(**newval**)

Retourne le nom du point d'accès (APN) à utiliser, si nécessaire.

cellular→set_apnAuth(**username**, **password**)

Configure les paramètres d'identification pour se connecter à l'APN.

cellular→set_dataReceived(**newval**)

Modifie la valeur du compteur d'octets reçus.

cellular→set_dataSent(**newval**)

Modifie la valeur du compteur d'octets envoyés.

cellular→set_enableData(**newval**)

Modifie la condition dans laquelle le service de données IP (GRPS) doit être activé.

cellular→set_lockedOperator(**newval**)

Modifie le nom de l'opérateur de réseau cellulaire à utiliser.

cellular→set_logicalName(**newval**)

Modifie le nom logique de l'interface cellulaire.

cellular→set_pin(**newval**)

Modifie le code PIN utilisé par le module pour accéder à la carte SIM.

cellular→set_pingInterval(**newval**)

Modifie l'intervalle entre les tests de connectivité spontanés, en secondes.

cellular→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`.

cellular→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

cellular→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.

YCellular.FindCellular() YCellular yFindCellular()YCellular.FindCellular() YCellular.FindCellular()

Permet de retrouver une interface cellulaire d'après un identifiant donné.

```
function FindCellular( 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 cellulaire soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YCellular.isOnline()` pour tester si l'interface cellulaire 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence l'interface cellulaire sans ambiguïté

Retourne :

un objet de classe `YCellular` qui permet ensuite de contrôler l'interface cellulaire.

YCellular.FindCellularInContext() YCellular
yFindCellularInContext()
YCellular.FindCellularInContext()
YCellular.FindCellularInContext()

Permet de retrouver une interface cellulaire d'après un identifiant donné dans un Context YAPI.

function **FindCellularInContext**(**yctx**, **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 cellulaire soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YCellular.isOnline()` pour tester si l'interface cellulaire 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'interface cellulaire sans ambiguïté

Retourne :

un objet de classe `YCellular` qui permet ensuite de contrôler l'interface cellulaire.

YCellular.FirstCellular() YCellular yFirstCellular()YCellular.FirstCellular() YCellular.FirstCellular()

Commence l'énumération des interfaces réseau cellulaire accessibles par la librairie.

function **FirstCellular**( )

Utiliser la fonction `YCellular.nextCellular()` pour itérer sur les autres interfaces réseau cellulaire.

Retourne :

un pointeur sur un objet `YCellular`, correspondant à la première interface cellulaire accessible en ligne, ou `null` si il n'y a pas de interfaces réseau cellulaire disponibles.

YCellular.FirstCellularInContext() YCellular
yFirstCellularInContext()
YCellular.FirstCellularInContext()
YCellular.FirstCellularInContext()

Commence l'énumération des interfaces réseau cellulaire accessibles par la librairie.

function **FirstCellularInContext**(**yctx**)

Utiliser la fonction `YCellular.nextCellular()` pour itérer sur les autres interfaces réseau cellulaire.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YCellular`, correspondant à la première interface cellulaire accessible en ligne, ou `null` si il n'y a pas de interfaces réseau cellulaire disponibles.

YCellular._AT() YCellular y_AT()cellular._AT()cellular._AT()

Envoie une commande AT au module GSM, et retourne le résultat.

```
function _AT( cmd)
```

La commande ne s'exécute que lorsque le module GSM dans un état standard, et doit le laisser exactement dans le même état. N'utilisez cette fonction qu'avec la plus grande prudence !

Paramètres :

cmd la commande AT à exécuter, comme par exemple: "+CCLK?"

Retourne :

une chaîne de caractères contenant le résultat de la commande. Les lignes vides sont automatiquement filtrées du résultat.

cellular→clearCache()cellular.clearCache() YCellular

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'interface cellulaire. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

cellular→clearDataCounters() YCellular
cellular.clearDataCounters()
cellular.clearDataCounters()

Réinitialise les compteurs de données transmises et reçues.

function **clearDataCounters**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

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

cellular→describe()cellular.describe() YCellular

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

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant l'interface cellulaire (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

cellular→get_advertisedValue() YCellular
cellular→advertisedValue()
cellular.get_advertisedValue()
cellular.get_advertisedValue()

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

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante de l'interface cellulaire (pas plus de 6 caractères).

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

cellular→get_airplaneMode() YCellular
cellular→airplaneMode()cellular.get_airplaneMode()
cellular.get_airplaneMode()

Retourne vrai si le mode avion est activé (radio désactivée).

function **get_airplaneMode**( )

Retourne :

soit `Y_AIRPLANEMODE_OFF`, soit `Y_AIRPLANEMODE_ON`, selon vrai si le mode avion est activé (radio désactivée)

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

cellular→get_apn() YCellular
cellular→apn()cellular.get_apn()cellular.get_apn()

Retourne le nom du point d'accès (APN) à utiliser, si nécessaire.

function **get_apn**( )

Lorsque l'APN est vide, celui proposé par l'opérateur cellulaire est utilisée.

Retourne :
une chaîne de caractères représentant le nom du point d'accès (APN) à utiliser, si nécessaire

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

cellular→get_apnSecret() YCellular
cellular→apnSecret()cellular.get_apnSecret()
cellular.get_apnSecret()

Retourne une string opaque si des paramètres d'identification sur l'APN ont été configurés dans le module, ou une chaîne vide autrement.

function **get_apnSecret**( )

Pour configurer ces paramètres, utilisez la méthode `set_apnAuth()`.

Retourne :

une chaîne de caractères représentant une string opaque si des paramètres d'identification sur l'APN ont été configurés dans le module, ou une chaîne vide autrement

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

cellular→get_availableOperators() YCellular
cellular→availableOperators()
cellular.get_availableOperators()
cellular.get_availableOperators()

Retourne la liste des opérateurs GSM disponibles à proximité.

function **get_availableOperators**( )

Cette fonction peut typiquement prendre 30 secondes à une minute pour rendre la main. Notez qu'en général une SIM ne permet de se connecter qu'à certains opérateur, et donc pas forcément à tous ceux listés par cette fonction.

Retourne :
une liste de noms d'opérateur.

cellular→get_cellIdentifier() YCellular
cellular→cellIdentifier()cellular.get_cellIdentifier()
cellular.get_cellIdentifier()

Retourne l'identifiant unique de la station de base utilisée: MCC, MNC, LAC et Cell ID.

function **get_cellIdentifier**( )

Retourne :
une chaîne de caractères représentant l'identifiant unique de la station de base utilisée: MCC, MNC, LAC et Cell ID

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

cellular→get_cellOperator() YCellular
cellular→cellOperator()cellular.get_cellOperator()
cellular.get_cellOperator()

Retourne le nom de l'opérateur de réseau cellulaire actuellement utilisé.

function **get_cellOperator**( )

Retourne :
une chaîne de caractères représentant le nom de l'opérateur de réseau cellulaire actuellement utilisé

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

cellular→get_cellType() YCellular
cellular→cellType()cellular.get_cellType()
cellular.get_cellType()

Type de connection cellulaire active.

function **get_cellType**( )

Retourne :

une valeur parmi `Y_CELLTYPE_GPRS`, `Y_CELLTYPE_EGPRS`, `Y_CELLTYPE_WCDMA`, `Y_CELLTYPE_HSDPA`, `Y_CELLTYPE_NONE` et `Y_CELLTYPE_CDMA`

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

cellular→get_dataReceived() YCellular
cellular→dataReceived()cellular.get_dataReceived()
cellular.get_dataReceived()

Retourne le nombre d'octets reçus jusqu'à présent.

function **get_dataReceived**( )

Retourne :
un entier représentant le nombre d'octets reçus jusqu'à présent

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

cellular→get_dataSent() YCellular
cellular→dataSent()cellular.get_dataSent()
cellular.get_dataSent()

Retourne le nombre d'octets envoyés jusqu'à présent.

function **get_dataSent**( )

Retourne :
un entier représentant le nombre d'octets envoyés jusqu'à présent

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

cellular→get_enableData() YCellular
cellular→enableData()cellular.get_enableData()
cellular.get_enableData()

Retourne la condition dans laquelle le service de données IP (GRPS) doit être activé.

function **get_enableData**( )

Lorsque le service de donnée n'est pas actif, seules les communications par SMS sont possibles.

Retourne :

une valeur parmi `Y_ENABLEDATA_HOMENETWORK`, `Y_ENABLEDATA_ROAMING`, `Y_ENABLEDATA_NEVER` et `Y_ENABLEDATA_NEUTRALITY` représentant la condition dans laquelle le service de données IP (GRPS) doit être activé

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

cellular→get_errorMessage() YCellular
cellular→errorMessage()cellular.get_errorMessage()

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

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 cellulaire.

cellular→get_errorType() YCellular
cellular→errorType()cellular.get_errorType()

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

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 cellulaire.

cellular→get_friendlyName() YCellular
cellular→friendlyName()cellular.get_friendlyName()

Retourne un identifiant global de l'interface cellulaire au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

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

Retourne :

une chaîne de caractères identifiant l'interface cellulaire en utilisant les noms logiques (ex: MyCustomName.relay1)

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

cellular→get_functionDescriptor() YCellular
cellular→functionDescriptor()
cellular.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

cellular→get_functionId() YCellular
cellular→functionId()cellular.get_functionId()

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

function **get_functionId**( )

Par example `relay1`.

Retourne :

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

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

cellular→get_hardwareId() YCellular
cellular→hardwareId()cellular.get_hardwareId()

Retourne l'identifiant matériel unique de l'interface cellulaire 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 cellulaire (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant l'interface cellulaire (ex: RELAYLO1-123456.relay1)

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

cellular→get_imsi() YCellular
cellular→imsi()cellular.get_imsi()cellular.get_imsi()

Retourne une string opaque si un code PIN a été configuré dans le module pour accéder à la carte SIM, ou une chaîne vide il n'a pas été configuré ou si la SIM a rejeté le code indiqué.

function **get_imsi**( )

Retourne :

une chaîne de caractères représentant une string opaque si un code PIN a été configuré dans le module pour accéder à la carte SIM, ou une chaîne vide il n'a pas été configuré ou si la SIM a rejeté le code indiqué

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

cellular→get_linkQuality() YCellular
cellular→linkQuality()cellular.get_linkQuality()
cellular.get_linkQuality()

Retourne la qualité de la connection, exprimée en pourcents.

function **get_linkQuality**( )

Retourne :
un entier représentant la qualité de la connection, exprimée en pourcents

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

cellular→get_lockedOperator() YCellular
cellular→lockedOperator()
cellular.get_lockedOperator()
cellular.get_lockedOperator()

Retourne le nom de l'opérateur de réseau cellulaire à utiliser exclusivement, si le choix automatique est désactivé, ou une chaîne vide si la carte SIM sélectionne automatiquement l'opérateur selon ceux disponibles.

function **get_lockedOperator**( )

Retourne :

une chaîne de caractères représentant le nom de l'opérateur de réseau cellulaire à utiliser exclusivement, si le choix automatique est désactivé, ou une chaîne vide si la carte SIM sélectionne automatiquement l'opérateur selon ceux disponibles

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

cellular→get_logicalName() YCellular
cellular→logicalName()cellular.get_logicalName()
cellular.get_logicalName()

Retourne le nom logique de l'interface cellulaire.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique de l'interface cellulaire.

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

cellular→get_message() YCellular
cellular→message()cellular.get_message()
cellular.get_message()

Retourne le dernier message de diagnostique de l'interface au réseau sans fil.

function **get_message**( )

Retourne :
une chaîne de caractères représentant le dernier message de diagnostique de l'interface au réseau sans fil

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

cellular→get_module() YCellular
cellular→module()cellular.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

cellular→get_pin() YCellular
cellular→pin()cellular.get_pin()cellular.get_pin()

Retourne une string opaque si un code PIN a été configuré dans le module pour accéder à la carte SIM, ou une chaîne vide il n'a pas été configuré ou si la SIM a rejeté le code indiqué.

```
function get_pin( )
```

Retourne :

une chaîne de caractères représentant une string opaque si un code PIN a été configuré dans le module pour accéder à la carte SIM, ou une chaîne vide il n'a pas été configuré ou si la SIM a rejeté le code indiqué

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

cellular→get_pingInterval() YCellular
cellular→pingInterval()cellular.get_pingInterval()
cellular.get_pingInterval()

Retourne l'intervalle entre les tests de connectivité spontanés, en secondes.

function **get_pingInterval**( )

Retourne :
un entier représentant l'intervalle entre les tests de connectivité spontanés, en secondes

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

cellular→get_userData() YCellular
cellular→userData()cellular.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.

cellular→isOnline()cellular.isOnline() YCellular

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

function **isOnline**( )

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

cellular→load()cellular.load() YCellular

Met en cache les valeurs courantes de l'interface cellulaire, 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 occasionellement 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.

cellular→loadAttribute()cellular.loadAttribute() YCellular cellular.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

cellular→muteValueCallbacks() YCellular
cellular.muteValueCallbacks()
cellular.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

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

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

cellular→nextCellular()cellular.nextCellular() YCellular cellular.nextCellular()

Continue l'énumération des interfaces réseau cellulaire commencée à l'aide de `yFirstCellular()`.

function **nextCellular**( )

Retourne :

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

cellular→quickCellSurvey()cellular.quickCellSurvey() YCellular cellular.quickCellSurvey()

Retourne la liste d'identifiants pour les antennes GSM à proximité, telle que requise pour géolocaliser rapidement le module.

function **quickCellSurvey**( )

La première antenne listée est la cellule active, et les suivantes sont les cellules voisines listée par la cellule active.

Retourne :
une liste de YCellRecord.

cellular→registerValueCallback() YCellular
cellular.registerValueCallback()
cellular.registerValueCallback()

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.

cellular→sendPUK()cellular.sendPUK() YCellular cellular.sendPUK()

Envoie le code PUK à la carte SIM pour la débloquer après trois échecs consécutifs de code PIN, et établit un nouveau code PIN dans la SIM.

function **sendPUK**(**puk**, **newPin**)

Seules dix tentatives consécutives de déblocage sont possibles: après dix tentatives infructueuses, la carte SIM sera définitivement inutilisable. Après avoir appelé cette fonction, vous devrez aussi appeler la méthode `set_pin()` pour indiquer au YoctoHub le nouveau PIN à utiliser dans le futur.

Paramètres :

puk code PUK de la carte SIM

newPin nouveau code PIN à configurer dans la carte SIM

Retourne :

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

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

cellular→set_airplaneMode() YCellular
cellular→setAirplaneMode()
cellular.set_airplaneMode()
cellular.set_airplaneMode()

Modifie l'état du mode avion (radio désactivée).

function **set_airplaneMode**(**newval**)

Paramètres :

newval soit `Y_AIRPLANEMODE_OFF`, soit `Y_AIRPLANEMODE_ON`, selon l'état du mode avion (radio désactivée)

Retourne :

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

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

cellular→set_apn() YCellular
cellular→setApn()cellular.set_apn()cellular.set_apn()

Retourne le nom du point d'accès (APN) à utiliser, si nécessaire.

function **set_apn**(**newval**)

Lorsque l'APN est vide, celui proposé par l'opérateur cellulaire est utilisée.

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.

cellular→set_apnAuth() YCellular
cellular→setApnAuth()cellular.set_apnAuth()
cellular.set_apnAuth()

Configure les paramètres d'identification pour se connecter à l'APN.

```
function set_apnAuth( username, password)
```

Les protocoles PAP et CHAP sont tous deux supportés.

Paramètres :

username nom d'utilisateur

password mot de passe

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

cellular→set_dataReceived() YCellular
cellular→setDataReceived()
cellular.set_dataReceived()
cellular.set_dataReceived()

Modifie la valeur du compteur d'octets reçus.

function **set_dataReceived**(**newval**)

Paramètres :

newval un entier représentant la valeur du compteur d'octets reçus

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

cellular→set_dataSent() YCellular
cellular→setDataSent()cellular.set_dataSent()
cellular.set_dataSent()

Modifie la valeur du compteur d'octets envoyés.

function **set_dataSent**(**newval**)

Paramètres :

newval un entier représentant la valeur du compteur d'octets envoyés

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

cellular→set_enableData() YCellular
cellular→setEnableData()cellular.set_enableData()
cellular.set_enableData()

Modifie la condition dans laquelle le service de données IP (GRPS) doit être activé.

function **set_enableData**(**newval**)

Le service peut être soit complètement désactivé, soit limité au réseau de de l'émetteur de la carte SIM, soit être activé pour tous les réseaux en partenariat avec la carte SIM (roaming). Attention, l'utilisation de données en roaming peut conduire à des coûts de télécommunication exhorbitants !

Lorsque le service de donnée n'est pas actif, seules les communications par SMS sont possibles.

Paramètres :

newval une valeur parmi `Y_ENABLEDATA_HOMENETWORK`, `Y_ENABLEDATA_ROAMING`, `Y_ENABLEDATA_NEVER` et `Y_ENABLEDATA_NEUTRALITY` représentant la condition dans laquelle le service de données IP (GRPS) doit être activé

Retourne :

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

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

cellular→set_lockedOperator() YCellular
cellular→setLockedOperator()
cellular.set_lockedOperator()
cellular.set_lockedOperator()

Modifie le nom de l'opérateur de réseau cellulaire à utiliser.

function **set_lockedOperator**(**newval**)

Si le nom est une chaîne vide, le choix sera fait automatiquement selon la carte SIM. Sinon, seul l'opérateur choisi sera utilisé.

Paramètres :

newval une chaîne de caractères représentant le nom de l'opérateur de réseau cellulaire à 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.

cellular→set_logicalName() YCellular
cellular→setLogicalName()cellular.set_logicalName()
cellular.set_logicalName()

Modifie le nom logique de l'interface cellulaire.

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 cellulaire.

Retourne :

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

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

cellular→set_pin() YCellular
cellular→setPin()cellular.set_pin()cellular.set_pin()

Modifie le code PIN utilisé par le module pour accéder à la carte SIM.

function **set_pin**(**newval**)

Cette fonction ne change pas le code sur la carte SIM elle-même, mais uniquement le paramètre utilisé par le module pour essayer d'en obtenir l'accès. Si le code SIM ne fonctionne pas dès le premier essai, il sera automatiquement oublié et un message "Enter SIM PIN" apparaître dans l'attribut 'message'. Il faudra alors appeler à nouveau cette méthode avec le bon code PIN. Après trois essais infructueux consécutifs le message devient "Enter SIM PUK" et il faut alors entrer le code PUK de la carte SIM avec la méthode `sendPUK`.

N'oubliez pas d'appeler la méthode `saveToFlash()` du module pour que le paramètre soit sauvegardé dans la flash.

Paramètres :

newval une chaîne de caractères représentant le code PIN utilisé par le module pour accéder à la carte SIM

Retourne :

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

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

cellular→set_pingInterval() YCellular
cellular→setPingInterval()cellular.set_pingInterval()
cellular.set_pingInterval()

Modifie l'intervalle entre les tests de connectivité spontanés, en secondes.

function **set_pingInterval**(**newval**)

Paramètres :

newval un entier représentant l'intervalle entre les tests de connectivité spontanés, 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.

cellular→set_userData() YCellular
cellular→setUserData()cellular.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

cellular→unmuteValueCallbacks() YCellular
cellular.unmuteValueCallbacks()
cellular.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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

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

cellular→wait_async()cellular.wait_async() YCellular

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.

3.11. Interface de la fonction ColorLed

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_colorled.js'></script>
cpp	#include "yocto_colorled.h"
m	#import "yocto_colorled.h"
pas	uses yocto_colorled;
vb	yocto_colorled.vb
cs	yocto_colorled.cs
java	import com.yoctopuce.YoctoAPI.YColorLed;
uwp	import com.yoctopuce.YoctoAPI.YColorLed;
py	from yocto_colorled import *
php	require_once('yocto_colorled.php');
es	in HTML: <script src="../../lib/yocto_colorled.js"></script> in node.js: require('yoctolib-es2017/yocto_colorled.js');

Fonction globales

yFindColorLed(**func**)

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

yFindColorLedInContext(**yctx**, **func**)

Permet de retrouver une LED RGB d'après un identifiant donné dans un Context YAPI.

yFirstColorLed()

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

yFirstColorLedInContext(**yctx**)

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

Méthodes des objets `YColorLed`

colorled→addHslMoveToBlinkSeq(**HSLcolor**, **msDelay**)

Ajoute une transition à la séquence de clignotement du module, la transition s'effectuera dans l'espace de couleur HSL.

colorled→addRgbMoveToBlinkSeq(**RGBcolor**, **msDelay**)

Ajoute une transition à la séquence de clignotement du module, la transition s'effectuera dans l'espace de couleur RGB

colorled→clearCache()

Invalide le cache.

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_blinkSeqMaxSize()

Retourne la longueur maximum de la sequence de clignotement.

colorled→get_blinkSeqSignature()

Retourne la signature de la signature de la séquence de clignotement.

colorled→get_blinkSeqSize()

Retourne la longueur actuelle de la sequence de clignotement.

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.

colorled→get_module()

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

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

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

colorled→get_rgbColor()

Retourne la couleur RGB courante de la LED.

colorled→get_rgbColorAtPowerOn()

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

colorled→get_userData()

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

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

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

colorled→isOnline()

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

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

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

colorled→load(**msValidity**)

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

colorled→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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

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

colorled→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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→resetBlinkSeq()

efface le contenu de la sequence de clignotement.

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 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→startBlinkSeq()

Démarre l'exécution de la séquence préprogrammée de clignotement.

colorled→stopBlinkSeq()

Arrête l'exécution de la séquence préprogrammée de clignotement.

colorled→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

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

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

```
function FindColorLed( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

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

Cette fonction n'exige pas que la LED RGB soit en ligne au moment ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindColorLedInContext() YColorLed
yFindColorLedInContext()
YColorLed.FindColorLedInContext()
YColorLed.FindColorLedInContext()

Permet de retrouver une LED RGB d'après un identifiant donné dans un Context YAPI.

function **FindColorLedInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

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

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

function **FirstColorLed**( )

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

Retourne :

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

YColorLed.FirstColorLedInContext() YColorLed
yFirstColorLedInContext()
YColorLed.FirstColorLedInContext()
YColorLed.FirstColorLedInContext()

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

function **FirstColorLedInContext**(**yctx**)

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

Paramètres :

yctx un contexte YAPI.

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→addHslMoveToBlinkSeq() YColorLed
colorled.addHslMoveToBlinkSeq()
colorled.addHslMoveToBlinkSeq()

Ajoute une transition à la séquence de clignotement du module, la transition s'effectuera dans l'espace de couleur HSL.

function **addHslMoveToBlinkSeq**(**HSLcolor**, **msDelay**)

Paramètres :

HSLcolor couleur HSL désirée à la fin de la transition

msDelay 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.

colorled→addRgbMoveToBlinkSeq() YColorLed
colorled.addRgbMoveToBlinkSeq()
colorled.addRgbMoveToBlinkSeq()

Ajoute une transition à la séquence de clignotement du module, la transition s'effectuera dans l'espace de couleur RGB

function **addRgbMoveToBlinkSeq**(**RGBcolor**, **msDelay**)

Paramètres :

RGBcolor couleur RGB désirée à la fin de la transition

msDelay 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.

colorled→clearCache()colorled.clearCache() YColorLed

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de la LED RGB. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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()
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_blinkSeqMaxSize() YColorLed
colorled→blinkSeqMaxSize()
colorled.get_blinkSeqMaxSize()
colorled.get_blinkSeqMaxSize()

Retourne la longueur maximum de la sequence de clignotement.

function **get_blinkSeqMaxSize**( )

Retourne :
un entier représentant la longueur maximum de la sequence de clignotement

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

colorled→get_blinkSeqSignature() YColorLed
colorled→blinkSeqSignature()
colorled.get_blinkSeqSignature()
colorled.get_blinkSeqSignature()

Retourne la signature de la signature de la séquence de clignotement.

function **get_blinkSeqSignature**( )

Les séquences de clignotement ne pouvant pas être relues du module, ce mécanisme peut être utilisé pour détecter si une séquence spécifique est déja programmée.

Retourne :
un entier représentant la signature de la signature de la séquence de clignotement

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

colorled→get_blinkSeqSize() YColorLed
colorled→blinkSeqSize()colorled.get_blinkSeqSize()
colorled.get_blinkSeqSize()

Retourne la longueur actuelle de la sequence de clignotement.

function **get_blinkSeqSize**( )

Retourne :
un entier représentant la longueur actuelle de la sequence de clignotement

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

colorled→get_errorMessage() YColorLed
colorled→errorMessage()
colorled.get_errorMessage()

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'identifant matériel de la LED RGB (par exemple: MyCustomName.relay1)

Retourne :
une chaîne de caractères identifiant la LED RGB en utilisant les noms logiques (ex: MyCustomName.relay1)

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

colorled→get_functionDescriptor() YColorLed
colorled→functionDescriptor()
colorled.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

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 example `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 example 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()
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()
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_rgbColor() YColorLed
colorled→rgbColor()colorled.get_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() YColorLed
colorled→rgbColorAtPowerOn()
colorled.get_rgbColorAtPowerOn()
colorled.get_rgbColorAtPowerOn()

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→hslMove()colorled.hslMove() YColorLed

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

function **hslMove**(**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→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 occasionellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→loadAttribute()colorled.loadAttribute() YColorLed colorled.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

colorled→muteValueCallbacks() YColorLed
colorled.muteValueCallbacks()
colorled.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

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

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

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

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() YColorLed
colorled.registerValueCallback()
colorled.registerValueCallback()

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→resetBlinkSeq()colorled.resetBlinkSeq() YColorLed colorled.resetBlinkSeq()

efface le contenu de la sequence de clignotement.

function **resetBlinkSeq**( )

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→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()
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() YColorLed
colorled→setLogicalName()
colorled.set_logicalName()
colorled.set_logicalName()

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()
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() YColorLed
colorled→setRgbColorAtPowerOn()
colorled.set_rgbColorAtPowerOn()
colorled.set_rgbColorAtPowerOn()

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

function **set_rgbColorAtPowerOn**(**newval**)

Paramètres :

newval un entier représentant la couleur que la LED va afficher 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→startBlinkSeq()colorled.startBlinkSeq() YColorLed colorled.startBlinkSeq()

Démarre l'exécution de la séquence préprogrammée de clignotement.

function **startBlinkSeq**( )

La séquence va tourner en boucle jusqu'à ce qu'elle soit stoppée par stopBlinkSeq ou un changement explicite.

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→stopBlinkSeq()colorled.stopBlinkSeq() YColorLed colorled.stopBlinkSeq()

Arrête l'exécution de la séquence préprogrammée de clignotement.

function **stopBlinkSeq**( )

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→unmuteValueCallbacks() YColorLed
colorled.unmuteValueCallbacks()
colorled.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.12. Interface de la fonction ColorLedCluster

La librairie de programmation Yoctopuce permet de piloter un cluster de LED. Contrairement a la classe ColorLed, la classe ColorLedCluster permet de changer modifier plusieurs LEDs à la fois. Les changements de couleur peuvent être fait 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 les LEDs avec une certaine teinte et d'en faire progressivement varier la saturation ou la luminosité.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_colorledcluster.js'></script>
cpp	#include "yocto_colorledcluster.h"
m	#import "yocto_colorledcluster.h"
pas	uses yocto_colorledcluster;
vb	yocto_colorledcluster.vb
cs	yocto_colorledcluster.cs
java	import com.yoctopuce.YoctoAPI.YColorLedCluster;
uwp	import com.yoctopuce.YoctoAPI.YColorLedCluster;
py	from yocto_colorledcluster import *
php	require_once('yocto_colorledcluster.php');
es	in HTML: <script src="../../lib/yocto_colorledcluster.js"></script> in node.js: require('yoctolib-es2017/yocto_colorledcluster.js');

Fonction globales

yFindColorLedCluster(**func**)

Permet de retrouver un cluster de LEDs RGB d'après un identifiant donné.

yFindColorLedClusterInContext(**yctx**, **func**)

Permet de retrouver un cluster de LEDs RGB d'après un identifiant donné dans un Context YAPI.

yFirstColorLedCluster()

Commence l'énumération des clusters de LEDs RGB accessibles par la librairie.

yFirstColorLedClusterInContext(**yctx**)

Commence l'énumération des clusters de LEDs RGB accessibles par la librairie.

Méthodes des objets `YColorLedCluster`

colorledcluster→addHslMoveToBlinkSeq(**seqIndex**, **hslValue**, **delay**)

Ajoute à une séquence une transition dans l'espace HSL.

colorledcluster→addMirrorToBlinkSeq(**seqIndex**)

Ajoute à une séquence une fin en mirroir.

colorledcluster→addRgbMoveToBlinkSeq(**seqIndex**, **rgbValue**, **delay**)

Ajoute à une séquence une transition dans l'espace RGB.

colorledcluster→clearCache()

Invalide le cache.

colorledcluster→describe()

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

colorledcluster→get_activeLedCount()

Retourne le nombre de LED actuellement gérées par le module.

colorledcluster→get_advertisedValue()

Retourne la valeur courante du cluster de LEDs RGB (pas plus de 6 caractères).

colorledcluster→get_blinkSeqMaxCount()

Retourne le nombre maximum de séquences mis à disposition par le module.

colorledcluster→get_blinkSeqMaxSize()

Retourne la longueur maximum d'une sequence.

colorledcluster→get_blinkSeqSignatures(**seqIndex**, **count**)

Retourne une liste de signatures 32 bits pour les séquences de clignotement.

colorledcluster→get_blinkSeqState(**seqIndex**, **count**)

Retourne une liste d'entiers indiquant si les séquences sont démarrées ou pas.

colorledcluster→get_blinkSeqStateAtPowerOn(**seqIndex**, **count**)

Retourne une liste d'entiers indiquant l'état du flag pilotant le démarrage de la séquence à la mise sous tension du module.

colorledcluster→get_blinkSeqStateSpeed(**seqIndex**, **count**)

Retourne une liste d'entiers donnant la vitesse de chaque séquence.

colorledcluster→get_errorMessage()

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

colorledcluster→get_errorType()

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

colorledcluster→get_friendlyName()

Retourne un identifiant global du cluster de LEDs RGB au format `NOM_MODULE.NOM_FONCTION`.

colorledcluster→get_functionDescriptor()

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

colorledcluster→get_functionId()

Retourne l'identifiant matériel du cluster de LEDs RGB, sans référence au module.

colorledcluster→get_hardwareId()

Retourne l'identifiant matériel unique du cluster de LEDs RGB au format `SERIAL.FUNCTIONID`.

colorledcluster→get_linkedSeqArray(**ledIndex**, **count**)

Retourne une liste d'index de séquence for chaque LED RGB.

colorledcluster→get_logicalName()

Retourne le nom logique du cluster de LEDs RGB.

colorledcluster→get_maxLedCount()

Retourne le nombre maximum de LEDs gérables par le module.

colorledcluster→get_module()

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

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

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

colorledcluster→get_rgbColorArray(**ledIndex**, **count**)

Retourne une liste de couleurs RGB 24 bits contenant l'état des LEDs RGB, tel quel.

colorledcluster→get_rgbColorArrayAtPowerOn(**ledIndex**, **count**)

Retourne une liste de couleurs RGB 24 bits contenant la couleur des LEDs RGB au démarrage.

colorledcluster→get_rgbColorBuffer(**ledIndex**, **count**)

Retourne un objet binaire contenant l'état des LEDs RGB, tel quel.

colorledcluster→get_userData()

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

colorledcluster→hslArray_move(**hslList**, **delay**)

Configure une transition HSL vers une liste de couleurs HSL, pixel par pixel.

colorledcluster→hsl_move(ledIndex, count, hslValue, delay)

Permet de faire passer un groupe de LED adjacentes de la couleur courante à une autre, de manière continue et indépendante.

colorledcluster→isOnline()

Vérifie si le module hébergeant le cluster de LEDs RGB est joignable, sans déclencher d'erreur.

colorledcluster→isOnline_async(callback, context)

Vérifie si le module hébergeant le cluster de LEDs RGB est joignable, sans déclencher d'erreur.

colorledcluster→linkLedToBlinkSeq(ledIndex, count, seqIndex, offset)

Lie un groupe de LEDs adjacentes à une séquence.

colorledcluster→linkLedToBlinkSeqAtPowerOn(ledIndex, count, seqIndex, offset)

Lie un groupe de LEDs adjacentes à une séquence, au démarrage du module.

colorledcluster→linkLedToPeriodicBlinkSeq(ledIndex, count, seqIndex, periods)

Lie un groupe de LEDs adjacentes à une séquence.

colorledcluster→load(msValidity)

Met en cache les valeurs courantes du cluster de LEDs RGB, avec une durée de validité spécifiée.

colorledcluster→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

colorledcluster→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du cluster de LEDs RGB, avec une durée de validité spécifiée.

colorledcluster→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

colorledcluster→nextColorLedCluster()

Continue l'énumération des clusters de LEDs RGB commencée à l'aide de `yFirstColorLedCluster()`.

colorledcluster→registerValueCallback(callback)

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

colorledcluster→resetBlinkSeq(seqIndex)

Stoppe l'exécution et efface le contenu d'une séquence.

colorledcluster→rgbArray_move(rgbList, delay)

Configure une transition RGB vers une liste de couleurs RGB, pixel par pixel.

colorledcluster→rgb_move(ledIndex, count, rgbValue, delay)

Permet de faire passer un groupe de LED adjacentes de la couleur courante à une autre, de manière continue et indépendante.

colorledcluster→saveBlinkSeq(seqIndex)

Sauve la configuration d'une séquence.

colorledcluster→saveLedsConfigAtPowerOn()

Sauve la configuration de démarrage des LEDs.

colorledcluster→set_activeLedCount(newval)

Modifie le nombre de LED actuellement gérées par le module.

colorledcluster→set_blinkSeqSpeed(seqIndex, speed)

Change la vitesse d'exécution d'une séquence, en pour mille.

colorledcluster→set_blinkSeqStateAtPowerOn(seqIndex, autostart)

Configure une séquence pour qu'elle démarre automatiquement au démarrage du module.

colorledcluster→set_hslColor(ledIndex, count, hslValue)

Modifie la couleur courante d'un groupe de LED consécutives en utilisant une couleur HSL .

colorledcluster→set_hslColorArray(ledIndex, hslList)

Envoie des couleurs HSL 24 bits (fournie sous forme d'une liste d'entiers) sur l'affichage LED HSL.

colorledcluster→set_hslColorBuffer(ledIndex, buff)

Envoie un objet binaire tel quel sur l'affichage LED HSL.

colorledcluster→set_logicalName(newval)

Modifie le nom logique du cluster de LEDs RGB.

colorledcluster→set_rgbColor(ledIndex, count, rgbValue)

Modifie la couleur courante d'un groupe de LED consécutives en utilisant une couleur RGB (Rouge Vert Bleu).

colorledcluster→set_rgbColorArray(ledIndex, rgbList)

Envoie des couleurs RGB 24 bits (fournie sous forme d'une liste d'entiers) sur l'affichage LED RGB.

colorledcluster→set_rgbColorAtPowerOn(ledIndex, count, rgbValue)

Modifie la couleur au démarrage d'un groupe de LED consécutives en utilisant une couleur RGB (Rouge Vert Bleu).

colorledcluster→set_rgbColorBuffer(ledIndex, buff)

Envoie un objet binaire tel quel sur l'affichage LED RGB.

colorledcluster→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`.

colorledcluster→startBlinkSeq(seqIndex)

Démarre l'exécution d'une séquence : toutes les LED liées à cette séquence vont commencer à l'exécuter en boucle.

colorledcluster→stopBlinkSeq(seqIndex)

Stoppe l'exécution d'une séquence.

colorledcluster→unlinkLedFromBlinkSeq(ledIndex, count)

Délie un groupe de LEDs adjacentes d'une séquence.

colorledcluster→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

colorledcluster→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.

YColorLedCluster.FindColorLedCluster() YColorLedCluster yFindColorLedCluster() YColorLedCluster.FindColorLedCluster() YColorLedCluster.FindColorLedCluster()

Permet de retrouver un cluster de LEDs RGB d'après un identifiant donné.

```
function FindColorLedCluster( 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 cluster de LEDs RGB soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YColorLedCluster.isOnline()` pour tester si le cluster de LEDs 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le cluster de LEDs RGB sans ambiguïté

Retourne :

un objet de classe `YColorLedCluster` qui permet ensuite de contrôler le cluster de LEDs RGB.

YColorLedCluster.FindColorLedClusterInContext() YColorLedCluster yFindColorLedClusterInContext() YColorLedCluster.FindColorLedClusterInContext() YColorLedCluster.FindColorLedClusterInContext()

Permet de retrouver un cluster de LEDs RGB d'après un identifiant donné dans un Context YAPI.

function **FindColorLedClusterInContext**(**yctx**, **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 cluster de LEDs RGB soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YColorLedCluster.isOnline()` pour tester si le cluster de LEDs 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le cluster de LEDs RGB sans ambiguïté

Retourne :

un objet de classe `YColorLedCluster` qui permet ensuite de contrôler le cluster de LEDs RGB.

YColorLedCluster.FirstColorLedCluster() YColorLedCluster yFirstColorLedCluster() YColorLedCluster.FirstColorLedCluster() YColorLedCluster.FirstColorLedCluster()

Commence l'énumération des clusters de LEDs RGB accessibles par la librairie.

```
function FirstColorLedCluster( )
```

Utiliser la fonction `YColorLedCluster.nextColorLedCluster()` pour itérer sur les autres clusters de LEDs RGB.

Retourne :

un pointeur sur un objet `YColorLedCluster`, correspondant au premier cluster de LEDs RGB accessible en ligne, ou `null` si il n'y a pas de clusters de LEDs RGB disponibles.

YColorLedCluster.FirstColorLedClusterInContext() YColorLedCluster yFirstColorLedClusterInContext() YColorLedCluster.FirstColorLedClusterInContext() YColorLedCluster.FirstColorLedClusterInContext()

Commence l'énumération des clusters de LEDs RGB accessibles par la librairie.

function **FirstColorLedClusterInContext**(**yctx**)

Utiliser la fonction `YColorLedCluster.nextColorLedCluster()` pour itérer sur les autres clusters de LEDs RGB.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YColorLedCluster`, correspondant au premier cluster de LEDs RGB accessible en ligne, ou `null` si il n'y a pas de clusters de LEDs RGB disponibles.

colorledcluster→addHslMoveToBlinkSeq() YColorLedCluster
colorledcluster.addHslMoveToBlinkSeq()
colorledcluster.addHslMoveToBlinkSeq()

Ajoute à une séquence une transition dans l'espace HSL.

function **addHslMoveToBlinkSeq**(**seqIndex**, **hslValue**, **delay**)

Une séquence est une liste de transitions qui pourra être exécutée en boucle par un groupe arbitraire de LEDs. Les séquences sont persistenteset sont sauvées dans le mémoire flash du module quand la méthode `saveBlinkSeq()` est appelée.

Paramètres :

seqIndex index de la séquence.

hslValue couleur visée (0xHHSSLL)

delay durée de la transistion en ms.

Retourne :

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

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

colorledcluster→addMirrorToBlinkSeq() YColorLedCluster
colorledcluster.addMirrorToBlinkSeq()
colorledcluster.addMirrorToBlinkSeq()

Ajoute à une séquence une fin en mirroir.

function **addMirrorToBlinkSeq**(**seqIndex**)

Lorsque la séquence arrivera à la fin de la dernière transition, sa vitesse d'exécution sera automatiquement inversée de sorte à ce que la séquence se rejoue à l'envers, en mirroir. Lorsque la première transition sera complétée à la fin de l'exécution à l'envers, la séquence repartira à nouveau dans le sens initial.

Paramètres :

seqIndex index de la séquence.

Retourne :

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

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

colorledcluster→addRgbMoveToBlinkSeq() YColorLedCluster
colorledcluster.addRgbMoveToBlinkSeq()
colorledcluster.addRgbMoveToBlinkSeq()

Ajoute à une séquence une transition dans l'espace RGB.

function **addRgbMoveToBlinkSeq**(**seqIndex**, **rgbValue**, **delay**)

Une séquence est une liste de transitions qui pourra être exécutée en boucle par un groupe arbitraire de LEDs. Les séquences sont persistantes et sont sauvées dans le mémoire flash du module quand la méthode `saveBlinkSeq()` est applée.

Paramètres :

seqIndex index de la séquence.

rgbValue couleur visée (0xRRGGBB)

delay durée de la transistion en ms.

Retourne :

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

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

colorledcluster→clearCache() YColorLedCluster
colorledcluster.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du cluster de LEDs RGB. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

colorledcluster→describe()colorledcluster.describe() YColorLedCluster

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

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le cluster de LEDs RGB (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

colorledcluster→get_activeLedCount() YColorLedCluster
colorledcluster→activeLedCount()
colorledcluster.get_activeLedCount()
colorledcluster.get_activeLedCount()

Retourne le nombre de LED actuellement gérées par le module.

function **get_activeLedCount**( )

Retourne :
un entier représentant le nombre de LED actuellement gérées par le module

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

colorledcluster→get_advertisedValue() YColorLedCluster
colorledcluster→advertisedValue()
colorledcluster.get_advertisedValue()
colorledcluster.get_advertisedValue()

Retourne la valeur courante du cluster de LEDs RGB (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :

une chaîne de caractères représentant la valeur courante du cluster de LEDs RGB (pas plus de 6 caractères).

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

colorledcluster→get_blinkSeqMaxCount() YColorLedCluster
colorledcluster→blinkSeqMaxCount()
colorledcluster.get_blinkSeqMaxCount()
colorledcluster.get_blinkSeqMaxCount()

Retourne le nombre maximum de séquences mis à disposition par le module.

function **get_blinkSeqMaxCount**( )

Retourne :
un entier représentant le nombre maximum de séquences mis à disposition par le module

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

colorledcluster→get_blinkSeqMaxSize() YColorLedCluster
colorledcluster→blinkSeqMaxSize()
colorledcluster.get_blinkSeqMaxSize()
colorledcluster.get_blinkSeqMaxSize()

Retourne la longueur maximum d'une sequence.

function **get_blinkSeqMaxSize**( )

Retourne :
un entier représentant la longueur maximum d'une sequence

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

colorledcluster→get_blinkSeqSignatures() YColorLedCluster
colorledcluster→blinkSeqSignatures()
colorledcluster.get_blinkSeqSignatures()
colorledcluster.get_blinkSeqSignatures()

Retourne une liste de signatures 32 bits pour les séquences de clignotement.

function **get_blinkSeqSignatures**(**seqIndex**, **count**)

Les séquences de clignotement ne pouvant pas être relues du module, ce mécanisme peut être utilisé pour détecter si une séquence spécifique est déjà programmée.

Paramètres :

seqIndex index de la première séquence qui doit être retournée

count nombre de séquences qui doivent être retournées

Retourne :

une liste d'entiers 32 bits

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

colorledcluster→get_blinkSeqState() YColorLedCluster
colorledcluster→blinkSeqState()
colorledcluster.get_blinkSeqState()
colorledcluster.get_blinkSeqState()

Retourne une liste d'entiers indiquant si les séquences sont démarrées ou pas.

function **get_blinkSeqState**(**seqIndex**, **count**)

Paramètres :

seqIndex index de la première séquence qui doit être retournée

count nombre de séquences qui doivent être retournées

Retourne :

une liste d'entiers, 0 pour les séquences arrêtées et 1 pour les séquences démarrées.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

colorledcluster→get_blinkSeqStateAtPowerOn() YColorLedCluster
colorledcluster→blinkSeqStateAtPowerOn()
colorledcluster.get_blinkSeqStateAtPowerOn()
colorledcluster.get_blinkSeqStateAtPowerOn()

Retourne une liste d'entiers indiquant l'état du flag pilotant le démarrage de la séquence à la mise sous tension du module.

function **get_blinkSeqStateAtPowerOn**(**seqIndex**, **count**)

Paramètres :

seqIndex index de la première séquence qui doit être retournée

count nombre de séquences qui doivent être retournées

Retourne :

une liste d'entiers, 0 pour les séquences arrêtées et 1 pour les séquences démarrées.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

colorledcluster→get_blinkSeqStateSpeed() YColorLedCluster
colorledcluster→blinkSeqStateSpeed()
colorledcluster.get_blinkSeqStateSpeed()
colorledcluster.get_blinkSeqStateSpeed()

Retourne une liste d'entiers donnant la vitesse de chaque séquence.

function **get_blinkSeqStateSpeed**(**seqIndex**, **count**)

Paramètres :

seqIndex index de la première séquence dont le vitesse doit être retournée

count nombre de séquences à traiter

Retourne :

une liste d'entiers, 0 pour les séquences arrêtées et 1 pour les séquences démarrées.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

colorledcluster→get_errorMessage() YColorLedCluster
colorledcluster→errorMessage()
colorledcluster.get_errorMessage()

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

colorledcluster→get_errorType() YColorLedCluster
colorledcluster→errorType()
colorledcluster.get_errorType()

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

colorledcluster→get_friendlyName() YColorLedCluster
colorledcluster→friendlyName()
colorledcluster.get_friendlyName()

Retourne un identifiant global du cluster de LEDs RGB au format `NOM_MODULE.NOM_FONCTION`.

function **get_friendlyName**( )

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

Retourne :

une chaîne de caractères identifiant le cluster de LEDs RGB en utilisant les noms logiques (ex: `MyCustomName.relay1`)

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

colorledcluster→get_functionDescriptor() YColorLedCluster
colorledcluster→functionDescriptor()
colorledcluster.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

colorledcluster→get_functionId() YColorLedCluster
colorledcluster→functionId()
colorledcluster.get_functionId()

Retourne l'identifiant matériel du cluster de LEDs RGB, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le cluster de LEDs RGB (ex: `relay1`)

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

colorledcluster→get_hardwareId() YColorLedCluster
colorledcluster→hardwareId()
colorledcluster.get_hardwareId()

Retourne l'identifiant matériel unique du cluster de LEDs RGB au format SERIAL.FUNCTIONID.

function **get_hardwareId**( )

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du cluster de LEDs RGB (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le cluster de LEDs RGB (ex: RELAYLO1-123456.relay1)

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

colorledcluster→get_linkedSeqArray() YColorLedCluster
colorledcluster→linkedSeqArray()
colorledcluster.get_linkedSeqArray()
colorledcluster.get_linkedSeqArray()

Retourne une liste d'index de séquence for chaque LED RGB.

function **get_linkedSeqArray**(**ledIndex**, **count**)

Le premier entier correspond à l'index de la première LED, l'entier suivant à la LED suivante, etc.

Paramètres :

ledIndex index de la première LED qui doit être retournée

count nombre de LEDs qui doivent être retournées

Retourne :

une liste d'entiers correspondant à des index de séquence.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

colorledcluster→get_logicalName() YColorLedCluster
colorledcluster→logicalName()
colorledcluster.get_logicalName()
colorledcluster.get_logicalName()

Retourne le nom logique du cluster de LEDs RGB.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du cluster de LEDs RGB.

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

colorledcluster→get_maxLedCount() YColorLedCluster
colorledcluster→maxLedCount()
colorledcluster.get_maxLedCount()
colorledcluster.get_maxLedCount()

Retourne le nombre maximum de LEDs gérables par le module.

function **get_maxLedCount**( )

Retourne :
un entier représentant le nombre maximum de LEDs gérables par le module

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

colorledcluster→get_module() YColorLedCluster
colorledcluster→module()
colorledcluster.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`

colorledcluster→get_rgbColorArray() YColorLedCluster
colorledcluster→rgbColorArray()
colorledcluster.get_rgbColorArray()
colorledcluster.get_rgbColorArray()

Retourne une liste de couleurs RGB 24 bits contenant l'état des LEDs RGB, tel quel.

function **get_rgbColorArray**(**ledIndex**, **count**)

Le premier entier correspond à la couleur RGB de la première LED, l'entier suivant à la LED suivante, etc.

Paramètres :

ledIndex index de la première LED qui doit être retournée

count nombre de LEDs qui doivent être retournées

Retourne :

une liste de couleurs 24bit avec les composantes RGB des LEDs choisies, au format 0xRRGGBB.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

colorledcluster→get_rgbColorArrayAtPowerOn() YColorLedCluster
colorledcluster→rgbColorArrayAtPowerOn()
colorledcluster.get_rgbColorArrayAtPowerOn()
colorledcluster.get_rgbColorArrayAtPowerOn()

Retourne une liste de couleurs RGB 24 bits contenant la couleur des LEDs RGB au démarrage.

function **get_rgbColorArrayAtPowerOn**(**ledIndex**, **count**)

Le premier entier correspond à la couleur RGB de démarrage de la première LED, l'entier suivant à la LED suivante, etc.

Paramètres :

ledIndex index de la première LED qui doit être retournée

count nombre de LEDs qui doivent être retournées

Retourne :

une liste de couleurs 24bit avec les composantes RGB des LEDs choisies, au format 0xRRGGBB.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

colorledcluster→get_rgbColorBuffer() YColorLedCluster
colorledcluster→rgbColorBuffer()
colorledcluster.get_rgbColorBuffer()
colorledcluster.get_rgbColorBuffer()

Retourne un objet binaire contenant l'état des LEDs RGB, tel quel.

function **get_rgbColorBuffer**(**ledIndex**, **count**)

Les trois premiers octets correspondent aux composantes RGB de la première LED choisie, les trois octets suivants à la LED suivante, etc.

Paramètres :

ledIndex index de la première LED qui doit être retournée

count nombre de LEDs qui doivent être retournées

Retourne :

un objet binaire avec les composantes RGB des LEDs choisies.

En cas d'erreur, déclenche une exception ou retourne un objet binaire vide.

colorledcluster→get_userData() YColorLedCluster
colorledcluster→userData()
colorledcluster.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.

colorledcluster→hslArray_move() YColorLedCluster
colorledcluster.hslArray_move()
colorledcluster.hslArray_move()

Configure une transition HSL vers une liste de couleurs HSL, pixel par pixel.

function **hslArray_move**(**hslList**, **delay**)

Le premier entier correspond à la couleur RGB finale pour la première LED, l'entier suivant à la LED suivante, etc.

Paramètres :

hslList la liste de valeurs HSL (24 bits) finales désirées, au format 0xHHSSLL

delay durée de la transition en ms.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorledcluster→hsl_move() YColorLedCluster
colorledcluster.hsl_move()
colorledcluster.hsl_move()

Permet de faire passer un groupe de LED adjacentes de la couleur courante à une autre, de manière continue et indépendante.

function **hsl_move**(**ledIndex**, **count**, **hslValue**, **delay**)

La transition se fait dans l'espace HSL. En HSL, la teinte étant une valeur circulaire (0..360°) il y a toujours deux manières d'opérer la transition: en augmentant ou en diminuant le teinte. Le module optera pour la transition passant par le chemin le plus court. dans le cas d'une différence d'exactement 180°, il optera pour la transition qui augmente la valeur de la teinte.

Paramètres :

ledIndex	index de la première LED affectée.
count	nombre de LED consécutives affectés.
hslValue	nouvelle couleur (0xHHSSLL)
delay	durée de la transistion enms.

Retourne :

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

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

colorledcluster→isOnline()colorledcluster.isOnline() YColorLedCluster

Vérifie si le module hébergeant le cluster de LEDs RGB est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du cluster de LEDs 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 le cluster de LEDs RGB est joignable, `false` sinon

colorledcluster→linkLedToBlinkSeq() YColorLedCluster
colorledcluster.linkLedToBlinkSeq()
colorledcluster.linkLedToBlinkSeq()

Lie un groupe de LEDs adjacentes à une séquence.

function **linkLedToBlinkSeq**(**ledIndex**, **count**, **seqIndex**, **offset**)

Ces LEDs commenceront à exécuter la séquence dès que son exécution sera lancé à l'aide de startBlinkSeq. Il est possible d'induire un décalage dans l'exécution à l'aide du paramètre offset. On peut ainsi créer plusieurs groupes de LEDs qui exécutent la même séquence mais de manière décalée. Une LED ne peut être affectée qu'à une seule séquence à la fois.

Paramètres :

ledIndex	index de la première LED affectée.
count	nombre de LED consécutives affectés.
seqIndex	index de la séquence.
offset	décalage dans l'exécution de la séquence en ms

Retourne :

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

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

colorledcluster→linkLedToBlinkSeqAtPowerOn() YColorLedCluster
colorledcluster.linkLedToBlinkSeqAtPowerOn()
colorledcluster.linkLedToBlinkSeqAtPowerOn()

Lie un groupe de LEDs adjacentes à une séquence, au démarrage du module.

function **linkLedToBlinkSeqAtPowerOn**(**ledIndex**, **count**, **seqIndex**, **offset**)

Ne pas oublier de configurer le démarrage automatique de la séquence et d'appeler `saveLedsConfigAtPowerOn()`. Il est possible d'induire un décalage dans l'exécution à l'aide du paramètre offset. On peut ainsi créer plusieurs groupes de LEDs qui exécutent la même séquence mais de manière décalée. Une LED ne peut être affectée qu'à une seule séquence à la fois.

Paramètres :

ledIndex	index de la première LED affectée.
count	nombre de LED consécutives affectés.
seqIndex	index de la séquence.
offset	décalage dans l'exécution de la séquence en ms

Retourne :

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

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

colorledcluster→linkLedToPeriodicBlinkSeq() YColorLedCluster
colorledcluster.linkLedToPeriodicBlinkSeq()
colorledcluster.linkLedToPeriodicBlinkSeq()

Lie un groupe de LEDs adjacentes à une séquence.

function **linkLedToPeriodicBlinkSeq**(**ledIndex**, **count**, **seqIndex**, **periods**)

Ces LEDs commenceront à exécuter la séquence dès que son exécution sera lancé à l'aide de startBlinkSeq. Cette fonction précalcule un décalage entre les LEDs de sorte à ce que le nombre choisi de périodes de la séquence soit visible sur le groupe de LEDs (effet d'onde).

Paramètres :

ledIndex	index de la première LED affectée.
count	nombre de LED consécutives affectés.
seqIndex	index de la séquence.
periods	nombre de périodes à répartir entre les LEDs.

Retourne :

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

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

colorledcluster→load()colorledcluster.load() YColorLedCluster

Met en cache les valeurs courantes du cluster de LEDs 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 occasionellement 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.

colorledcluster→loadAttribute() YColorLedCluster
colorledcluster.loadAttribute()
colorledcluster.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

colorledcluster→muteValueCallbacks() YColorLedCluster
colorledcluster.muteValueCallbacks()
colorledcluster.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

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

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

colorledcluster→nextColorLedCluster() YColorLedCluster
colorledcluster.nextColorLedCluster()
colorledcluster.nextColorLedCluster()

Continue l'énumération des clusters de LEDs RGB commencée à l'aide de `yFirstColorLedCluster()`.

function **nextColorLedCluster**( )

Retourne :

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

colorledcluster→registerValueCallback() YColorLedCluster
colorledcluster.registerValueCallback()
colorledcluster.registerValueCallback()

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.

colorledcluster→resetBlinkSeq() YColorLedCluster
colorledcluster.resetBlinkSeq()
colorledcluster.resetBlinkSeq()

Stoppe l'exécution et efface le contenu d'une séquence.

function **resetBlinkSeq**(**seqIndex**)

Les LEDs liées à cette séquence ne seront plus mises à jour.

Paramètres :

seqIndex index de la séquence à réinitialiser.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorledcluster→rgbArray_move() YColorLedCluster
colorledcluster.rgbArray_move()
colorledcluster.rgbArray_move()

Configure une transition RGB vers une liste de couleurs RGB, pixel par pixel.

function **rgbArray_move**(**rgbList**, **delay**)

Le premier entier correspond à la couleur RGB finale pour la première LED, l'entier suivant à la LED suivante, etc.

Paramètres :

rgbList la liste de valeurs RGB (24 bits) finales désirées, au format 0xRRGGBB

delay durée de la transition en ms.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorledcluster→rgb_move() YColorLedCluster
colorledcluster.rgb_move()
colorledcluster.rgb_move()

Permet de faire passer un groupe de LED adjacentes de la couleur courante à une autre, de manière continue et indépendante.

function **rgb_move**(**ledIndex**, **count**, **rgbValue**, **delay**)

La transition se fait dans l'espace RGB.

Paramètres :

ledIndex	index de la première LED affectée.
count	nombre de LED consécutives affectés.
rgbValue	nouvelle couleur (0xRRGGBB)
delay	durée de la transistion enms.

Retourne :

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

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

colorledcluster→saveBlinkSeq() YColorLedCluster
colorledcluster.saveBlinkSeq()
colorledcluster.saveBlinkSeq()

Sauve la configuration d'une séquence.

function **saveBlinkSeq**(**seqIndex**)

Attention, seul les paramètres de la séquence sont sauvés. Pour sauver le choix de séquence des LEDs il faut appeler la méthode `saveLedsConfigAtPowerOn()`.

Paramètres :

seqIndex index de la séquence à lancer.

Retourne :

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

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

colorledcluster→saveLedsConfigAtPowerOn() YColorLedCluster
colorledcluster.saveLedsConfigAtPowerOn()
colorledcluster.saveLedsConfigAtPowerOn()

Sauve la configuration de démarrage des LEDs.

function **saveLedsConfigAtPowerOn**( )

Cela inclut la couleur de démarrage ou le choix de séquence de démarrage pour toutes les LEDs. Attention, si des LEDs sont liées à une séquence il faut appeler la méthode `saveBlinkSeq()` en plus pour sauver la définition de la séquence.

Retourne :

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

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

colorledcluster→set_activeLedCount() YColorLedCluster
colorledcluster→setActiveLedCount()
colorledcluster.set_activeLedCount()
colorledcluster.set_activeLedCount()

Modifie le nombre de LED actuellement gérées par le module.

function **set_activeLedCount**(**newval**)

Paramètres :

newval un entier représentant le nombre de LED actuellement gérées 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.

colorledcluster→set_blinkSeqSpeed() YColorLedCluster
colorledcluster→setBlinkSeqSpeed()
colorledcluster.set_blinkSeqSpeed()
colorledcluster.set_blinkSeqSpeed()

Change la vitesse d'exécution d'une séquence, en pour mille.

function **set_blinkSeqSpeed**(**seqIndex**, **speed**)

La vitesse d'exécution naturelle est de 1000 pour mille. En configurant une vitesse inférieure, on peut jouer la séquence au ralenti. Une vitesse négative permet même de jouer la séquence à l'envers.

Paramètres :

seqIndex index de la séquence à lancer.

speed vitesse d'exécution de la séquence (-1000...1000).

Retourne :

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

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

colorledcluster→set_blinkSeqStateAtPowerOn() YColorLedCluster
colorledcluster→setBlinkSeqStateAtPowerOn()
colorledcluster.set_blinkSeqStateAtPowerOn()
colorledcluster.set_blinkSeqStateAtPowerOn()

Configure une séquence pour qu'elle démarre automatiquement au démarrage du module.

function **set_blinkSeqStateAtPowerOn**(**seqIndex**, **autostart**)

N'oubliez pas d'appeler `saveBlinkSeq()` pour sauvegarder la modification dans la mémoire flash du module.

Paramètres :

seqIndex index de la séquence concernée.

autostart 0 pour que la séquence soit arrêtée et 1 pour qu'elle démarre.

Retourne :

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

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

colorledcluster→set_hslColor() YColorLedCluster
colorledcluster→setHslColor()
colorledcluster.set_hslColor()
colorledcluster.set_hslColor()

Modifie la couleur courante d'un groupe de LED consécutives en utilisant une couleur HSL .

function **set_hslColor**(**ledIndex**, **count**, **hslValue**)

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

Paramètres :

ledIndex index de la première LED affectée.

count nombre de LED consécutives affectés.

hslValue nouvelle couleur.

Retourne :

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

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

colorledcluster→set_hslColorArray() YColorLedCluster
colorledcluster→setHslColorArray()
colorledcluster.set_hslColorArray()
colorledcluster.set_hslColorArray()

Envoie des couleurs HSL 24 bits (fournie sous forme d'une liste d'entiers) sur l'affichage LED HSL.

function **set_hslColorArray**(**ledIndex**, **hslList**)

Le premier entier correspond à la couleur HSL de la LED indiquée en argument, l'entier suivant à la LED suivante, etc.

Paramètres :

ledIndex index de la première LED qui doit être modifiée

hslList la liste de valeurs HSL (24 bits) à envoyer, au format 0xHHSSLL

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorledcluster→set_hslColorBuffer() YColorLedCluster
colorledcluster→setHslColorBuffer()
colorledcluster.set_hslColorBuffer()
colorledcluster.set_hslColorBuffer()

Envoie un objet binaire tel quel sur l'affichage LED HSL.

function **set_hslColorBuffer**(**ledIndex**, **buff**)

Les trois premiers octets correspondent aux composantes HSL de la LED indiquée en argument, les trois octets suivants à la LED suivante, etc.

Paramètres :

ledIndex index de la première LED qui doit être modifiée

buff l'objet binaire à envoyer

Retourne :

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

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

colorledcluster→set_logicalName() YColorLedCluster
colorledcluster→setLogicalName()
colorledcluster.set_logicalName()
colorledcluster.set_logicalName()

Modifie le nom logique du cluster de LEDs 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 du cluster de LEDs 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.

colorledcluster→set_rgbColor() YColorLedCluster
colorledcluster→setRgbColor()
colorledcluster.set_rgbColor()
colorledcluster.set_rgbColor()

Modifie la couleur courante d'un groupe de LED consécutives en utilisant une couleur RGB (Rouge Vert Bleu).

function **set_rgbColor**(**ledIndex**, **count**, **rgbValue**)

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

Paramètres :

ledIndex index de la première LED affectée.

count nombre de LED consécutives affectés.

rgbValue nouvelle couleur.

Retourne :

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

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

colorledcluster→set_rgbColorArray() YColorLedCluster
colorledcluster→setRgbColorArray()
colorledcluster.set_rgbColorArray()
colorledcluster.set_rgbColorArray()

Envoie des couleurs RGB 24 bits (fournie sous forme d'une liste d'entiers) sur l'affichage LED RGB.

function **set_rgbColorArray**(**ledIndex**, **rgbList**)

Le premier entier correspond à la couleur RGB de la LED indiquée en argument, l'entier suivant à la LED suivante, etc.

Paramètres :

ledIndex index de la première LED qui doit être modifiée

rgbList la liste de valeurs RGB (24 bits) à envoyer, au format 0xRRGGBB

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorledcluster→set_rgbColorAtPowerOn() YColorLedCluster
colorledcluster→setRgbColorAtPowerOn()
colorledcluster.set_rgbColorAtPowerOn()
colorledcluster.set_rgbColorAtPowerOn()

Modifie la couleur au démarrage d'un groupe de LED consécutives en utilisant une couleur RGB (Rouge Vert Bleu).

function **set_rgbColorAtPowerOn**(**ledIndex**, **count**, **rgbValue**)

L'encodage est réalisé de la manière suivante: 0xRRGGBB. N'oubliez pas d'appeler `saveLedsConfigAtPowerOn()` pour sauvegarder la modification dans la mémoire flash du module.

Paramètres :

ledIndex	index de la première LED affectée.
count	nombre de LED consécutives affectés.
rgbValue	nouvelle couleur.

Retourne :

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

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

colorledcluster→set_rgbColorBuffer() YColorLedCluster
colorledcluster→setRgbColorBuffer()
colorledcluster.set_rgbColorBuffer()
colorledcluster.set_rgbColorBuffer()

Envoie un objet binaire tel quel sur l'affichage LED RGB.

function **set_rgbColorBuffer**(**ledIndex**, **buff**)

Les trois premiers octets correspondent aux composantes RGB de la LED indiquée en argument, les trois octets suivants à la LED suivante, etc.

Paramètres :

ledIndex index de la première LED qui doit être modifiée

buff l'objet binaire à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorledcluster→set_userData() YColorLedCluster
colorledcluster→setUserData()
colorledcluster.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

colorledcluster→startBlinkSeq() YColorLedCluster
colorledcluster.startBlinkSeq()
colorledcluster.startBlinkSeq()

Démarre l'exécution d'une séquence : toutes les LED liées à cette séquence vont commencer à l'exécuter en boucle.

function **startBlinkSeq**(**seqIndex**)

Paramètres :

seqIndex index de la séquence à lancer.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorledcluster→stopBlinkSeq() YColorLedCluster
colorledcluster.stopBlinkSeq()
colorledcluster.stopBlinkSeq()

Stoppe l'exécution d'une séquence.

function **stopBlinkSeq**(**seqIndex**)

Si la séquence est redémarrée l'exécution repartira du début.

Paramètres :

seqIndex index de la séquence à stopper.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorledcluster→unlinkLedFromBlinkSeq() YColorLedCluster
colorledcluster.unlinkLedFromBlinkSeq()
colorledcluster.unlinkLedFromBlinkSeq()

Délie un groupe de LEDs adjacentes d'une séquence.

function **unlinkLedFromBlinkSeq**(**ledIndex**, **count**)

Paramètres :

ledIndex index de la première LED affectée.

count nombre de LEDs consécutives affectées.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorledcluster→unmuteValueCallbacks() YColorLedCluster
colorledcluster.unmuteValueCallbacks()
colorledcluster.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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

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

colorledcluster→wait_async() YColorLedCluster
colorledcluster.wait_async()

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 Compass

La classe YSensor est la classe parente de tous les senseurs Yoctopuce. Elle permet de lire la valeur courante et l'unité de n'importe quel capteur, de lire les valeurs min/max, de configurer la fréquence d'enregistrement autonome des données et de récupérer les mesures enregistrées. Elle permet aussi d'enregistrer un callback appelé lorsque la valeur mesurée change ou à intervalle prédéfini. L'utilisation de cette classe plutôt qu'une de ces sous-classes permet de créer des application génériques, compatibles même avec les capteurs Yoctopuce futurs. Note: la classe YAnButton est le seul type d'entrée analogique qui n'hérite pas de YSensor.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_compass.js'></script>
cpp	#include "yocto_compass.h"
m	#import "yocto_compass.h"
pas	uses yocto_compass;
vb	yocto_compass.vb
cs	yocto_compass.cs
java	import com.yoctopuce.YoctoAPI.YCompass;
uwp	import com.yoctopuce.YoctoAPI.YCompass;
py	from yocto_compass import *
php	require_once('yocto_compass.php');
es	in HTML: <script src="../../lib/yocto_compass.js"></script> in node.js: require('yoctolib-es2017/yocto_compass.js');

Fonction globales

yFindCompass(**func**)

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

yFindCompassInContext(**yctx**, **func**)

Permet de retrouver un compas d'après un identifiant donné dans un Context YAPI.

yFirstCompass()

Commence l'énumération des compas accessibles par la librairie.

yFirstCompassInContext(**yctx**)

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→clearCache()

Invalide le cache.

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_bandwidth()

Retourne la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

compass→get_currentRawValue()

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

compass→get_currentValue()

Retourne la valeur actuelle du cap relatif, en degrés, sous forme de nombre à virgule.

compass→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

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, indépendemment 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_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

compass→load(msValidity)

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

compass→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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_bandwidth(newval)

Modifie la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

compass→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

compass→set_logFrequency(newval)

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

compass→set_logicalName(newval)

Modifie le nom logique du compas.

compass→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

compass→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

compass→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

compass→set_userData(data)

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

compass→startDataLogger()

Démarre l'enregistreur de données du module.

compass→stopDataLogger()

Arrête l'enregistreur de données du module.

compass→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

compass→wait_async(callback, context)

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

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

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

```
function FindCompass( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindCompassInContext() YCompass
yFindCompassInContext()
YCompass.FindCompassInContext()
YCompass.FindCompassInContext()

Permet de retrouver un compas d'après un identifiant donné dans un Context YAPI.

function **FindCompassInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le compas sans ambiguïté

Retourne :

un objet de classe `YCompass` qui permet ensuite de contrôler le compas.

YCompass.FirstCompass() YCompass yFirstCompass()YCompass.FirstCompass() YCompass.FirstCompass()

Commence l'énumération des compas accessibles par la librairie.

function **FirstCompass**( )

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.

YCompass.FirstCompassInContext() YCompass
yFirstCompassInContext()
YCompass.FirstCompassInContext()
YCompass.FirstCompassInContext()

Commence l'énumération des compas accessibles par la librairie.

function **FirstCompassInContext**(**yctx**)

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

Paramètres :

yctx un contexte YAPI.

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() YCompass
compass.calibrateFromPoints()
compass.calibrateFromPoints()

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

function **calibrateFromPoints**(**rawValues**, **refValues**)

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

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

Paramètres :

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

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

Retourne :

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

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

compass→clearCache()compass.clearCache() YCompass

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du compas. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le compas (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

compass→get_advertisedValue() YCompass
compass→advertisedValue()
compass.get_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_bandwidth() YCompass
compass→bandwidth()compass.get_bandwidth()
compass.get_bandwidth()

Retourne la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

function **get_bandwidth**( )

Retourne :
un entier représentant la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement)

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

compass→get_currentRawValue() YCompass
compass→currentRawValue()
compass.get_currentRawValue()
compass.get_currentRawValue()

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

function **get_currentRawValue**( )

Retourne :
une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés, sous forme de nombre à virgule

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

compass→get_currentValue() YCompass
compass→currentValue()
compass.get_currentValue()
compass.get_currentValue()

Retourne la valeur actuelle du cap relatif, en degrés, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :

une valeur numérique représentant la valeur actuelle du cap relatif, en degrés, sous forme de nombre à virgule

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

compass→get_dataLogger() YCompass
compass→dataLogger()compass.get_dataLogger()
compass.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

compass→get_errorMessage() YCompass
compass→errorMessage()
compass.get_errorMessage()

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

```
function get_errorMessage( )
```

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

Retourne :

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

compass→get_errorType() YCompass
compass→errorType()compass.get_errorType()

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

function **get_errorType**( )

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

Retourne :
un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du compas.

compass→get_friendlyName() 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'identifant 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 example `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 example 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()
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()
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()
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()
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()
compass.get_magneticHeading()

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

function **get_magneticHeading**( )

Retourne :
une valeur numérique représentant la direction du nord magnétique, indépendemment du cap configuré

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

compass→get_module() YCompass
compass→module()compass.get_module()

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

function **get_module**( )

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

Retourne :
une instance de YModule

compass→get_recordedData() YCompass
compass→recordedData()
compass.get_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)
```

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 find de l'intercalle 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()
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()
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_sensorState() YCompass
compass→sensorState()compass.get_sensorState()
compass.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :
un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

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

compass→get_unit() YCompass
compass→unit()compass.get_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.

compass→isOnline()compass.isOnline() YCompass

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

function **isOnline**( )

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

Retourne :
`true` si le compas est joignable, `false` sinon

compass→load()compass.load() YCompass

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

function **load**(**msValidity**)

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

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

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

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

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

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

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

Retourne :

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

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

compass→muteValueCallbacks() YCompass
compass.muteValueCallbacks()
compass.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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

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

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

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() YCompass
compass.registerTimedReportCallback()
compass.registerTimedReportCallback()

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() YCompass
compass.registerValueCallback()
compass.registerValueCallback()

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_bandwidth() YCompass
compass→setBandwidth()compass.set_bandwidth()
compass.set_bandwidth()

Modifie la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

function **set_bandwidth**(**newval**)

Lorsque la fréquence est plus basse, un moyennage est effectué.

Paramètres :

newval un entier représentant la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement)

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_highestValue() YCompass
compass→setHighestValue()
compass.set_highestValue()
compass.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.

compass→set_logFrequency() YCompass
compass→setLogFrequency()
compass.set_logFrequency()
compass.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.

compass→set_logicalName() YCompass
compass→setLogicalName()
compass.set_logicalName()
compass.set_logicalName()

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() YCompass
compass→setLowestValue()
compass.set_lowestValue()
compass.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.

compass→set_reportFrequency() YCompass
compass→setReportFrequency()
compass.set_reportFrequency()
compass.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.

compass→set_resolution() YCompass
compass→setResolution()compass.set_resolution()
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→startDataLogger() YCompass
compass.startDataLogger()
compass.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

compass→stopDataLogger() YCompass
compass.stopDataLogger()
compass.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

compass→unmuteValueCallbacks() YCompass
compass.unmuteValueCallbacks()
compass.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.14. Interface de la fonction Current

La classe YCurrent permet de lire et de configurer les capteurs de courant Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_current.js'></script>
cpp     #include "yocto_current.h"
m       #import "yocto_current.h"
pas     uses yocto_current;
vb      yocto_current.vb
cs      yocto_current.cs
java    import com.yoctopuce.YoctoAPI.YCurrent;
uwp     import com.yoctopuce.YoctoAPI.YCurrent;
py      from yocto_current import *
php     require_once('yocto_current.php');
es      in HTML: <script src="../../lib/yocto_current.js"></script>
        in node.js: require('yoctolib-es2017/yocto_current.js');
```

Fonction globales
yFindCurrent(**func**) Permet de retrouver un capteur de courant d'après un identifiant donné.
yFindCurrentInContext(**yctx**, **func**) Permet de retrouver un capteur de courant d'après un identifiant donné dans un Context YAPI.
yFirstCurrent() Commence l'énumération des capteurs de courant accessibles par la librairie.
yFirstCurrentInContext(**yctx**) 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→clearCache() Invalide le cache.
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), en mA, sous forme de nombre à virgule.
current→get_currentValue() Retourne la valeur actuelle du courant, en mA, sous forme de nombre à virgule.
current→get_dataLogger() Retourne l'objet YDataLogger du module qui héberge le senseur.
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 depuis le démarrage du module.

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 depuis le démarrage du module.

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_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

current→load(**msValidity**)

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

current→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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.

current→set_logFrequency(newval)

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

current→set_logicalName(newval)

Modifie le nom logique du capteur de courant.

current→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

current→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

current→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

current→set_userData(data)

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

current→startDataLogger()

Démarre l'enregistreur de données du module.

current→stopDataLogger()

Arrête l'enregistreur de données du module.

current→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

current→wait_async(callback, context)

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

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

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

```
function FindCurrent( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindCurrentInContext() YCurrent
yFindCurrentInContext()
YCurrent.FindCurrentInContext()
YCurrent.FindCurrentInContext()

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

function **FindCurrentInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

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

Retourne :

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

YCurrent.FirstCurrent() YCurrent yFirstCurrent()YCurrent.FirstCurrent() YCurrent.FirstCurrent()

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

function **FirstCurrent**( )

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.

YCurrent.FirstCurrentInContext() YCurrent
yFirstCurrentInContext()
YCurrent.FirstCurrentInContext()
YCurrent.FirstCurrentInContext()

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

function **FirstCurrentInContext**(**yctx**)

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

Paramètres :

yctx un contexte YAPI.

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() YCurrent
current.calibrateFromPoints()
current.calibrateFromPoints()

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

function **calibrateFromPoints**(**rawValues**, **refValues**)

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

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

Paramètres :

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

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

Retourne :

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

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

current→clearCache()current.clearCache() YCurrent

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur de courant. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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()
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()
current.get_currentRawValue()

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

function **get_currentRawValue**( )

Retourne :

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

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

current→get_currentValue() YCurrent
current→currentValue()current.get_currentValue()
current.get_currentValue()

Retourne la valeur actuelle du courant, en mA, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle du courant, en mA, sous forme de nombre à virgule

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

current→get_dataLogger() YCurrent
current→dataLogger()current.get_dataLogger()
current.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

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'identifant 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 example `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 example 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()
current.get_highestValue()

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

function **get_highestValue**( )

Retourne :

une valeur numérique représentant la valeur maximale observée pour le courant depuis le démarrage du module

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

current→get_logFrequency() YCurrent
current→logFrequency()current.get_logFrequency()
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()
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()
current.get_lowestValue()

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

function **get_lowestValue**( )

Retourne :
une valeur numérique représentant la valeur minimale observée pour le courant depuis le démarrage du module

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

current→get_module() YCurrent
current→module()current.get_module()

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

function **get_module**( )

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

Retourne :
une instance de YModule

current→get_recordedData() YCurrent
current→recordedData()current.get_recordedData()
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**)

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 find de l'intercalle 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()
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()
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_sensorState() YCurrent
current→sensorState()current.get_sensorState()
current.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :

un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

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

current→get_unit() YCurrent
current→unit()current.get_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→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 occasionellement 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→loadAttribute()current.loadAttribute() YCurrent current.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

current→loadCalibrationPoints() YCurrent
current.loadCalibrationPoints()
current.loadCalibrationPoints()

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

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

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

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

Retourne :

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

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

current→muteValueCallbacks() YCurrent
current.muteValueCallbacks()
current.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

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

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

current→nextCurrent()current.nextCurrent() YCurrent current.nextCurrent()

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() YCurrent
current.registerTimedReportCallback()
current.registerTimedReportCallback()

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

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

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

Paramètres :

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

current→registerValueCallback() YCurrent
current.registerValueCallback()
current.registerValueCallback()

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() YCurrent
current→setHighestValue()
current.set_highestValue()current.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.

current→set_logFrequency() YCurrent
current→setLogFrequency()
current.set_logFrequency()
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()
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()
current.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.

current→set_reportFrequency() YCurrent
current→setReportFrequency()
current.set_reportFrequency()
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()
current.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.

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→startDataLogger()current.startDataLogger() YCurrent current.startDataLogger()

Démarre l'enregistreur de données du module.

```
function startDataLogger( )
```

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :

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

current→stopDataLogger()current.stopDataLogger() YCurrent current.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

current→unmuteValueCallbacks() YCurrent
current.unmuteValueCallbacks()
current.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.15. Interface de la fonction CurrentLoopOutput

La librairie de programmation Yoctopuce permet de changer la valeur de la sortie 4-20mA et de connaitre l'état de la boucle de courant.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_currentloopoutput.js'></script>
cpp     #include "yocto_currentloopoutput.h"
m       #import "yocto_currentloopoutput.h"
pas     uses yocto_currentloopoutput;
vb      yocto_currentloopoutput.vb
cs      yocto_currentloopoutput.cs
java    import com.yoctopuce.YoctoAPI.YCurrentLoopOutput;
uwp     import com.yoctopuce.YoctoAPI.YCurrentLoopOutput;
py      from yocto_currentloopoutput import *
php     require_once('yocto_currentloopoutput.php');
es      in HTML: <script src="../../lib/yocto_currentloopoutput.js"></script>
        in node.js: require('yoctolib-es2017/yocto_currentloopoutput.js');
```

Fonction globales
yFindCurrentLoopOutput(**func**) Permet de retrouver une sortie 4-20mA d'après un identifiant donné.
yFindCurrentLoopOutputInContext(**yctx**, **func**) Permet de retrouver une sortie 4-20mA d'après un identifiant donné dans un Context YAPI.
yFirstCurrentLoopOutput() Commence l'énumération des sortie 4-20mA accessibles par la librairie.
yFirstCurrentLoopOutputInContext(**yctx**) Commence l'énumération des sortie 4-20mA accessibles par la librairie.
Méthodes des objets `YCurrentLoopOutput`
currentloopoutput→clearCache() Invalide le cache.
currentloopoutput→currentMove(**mA_target**, **ms_duration**) Déclenche une transition progressive du courant dans la boucle.
currentloopoutput→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de la sortie sortie 4-20mA au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
currentloopoutput→get_advertisedValue() Retourne la valeur courante de la sortie sortie 4-20mA (pas plus de 6 caractères).
currentloopoutput→get_current() Retourne la valeur de consigne pour le courant de la boucle, en mA.
currentloopoutput→get_currentAtStartUp() Retourne le courant dans le boucle au démarrage du module, en mA.
currentloopoutput→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la sortie sortie 4-20mA.
currentloopoutput→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la sortie sortie 4-20mA.
currentloopoutput→get_friendlyName()

Retourne un identifiant global de la sortie sortie 4-20mA au format NOM_MODULE.NOM_FONCTION.

currentloopoutput→get_functionDescriptor()

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

currentloopoutput→get_functionId()

Retourne l'identifiant matériel de la sortie sortie 4-20mA, sans référence au module.

currentloopoutput→get_hardwareId()

Retourne l'identifiant matériel unique de la sortie sortie 4-20mA au format SERIAL.FUNCTIONID.

currentloopoutput→get_logicalName()

Retourne le nom logique de la sortie sortie 4-20mA.

currentloopoutput→get_loopPower()

Retourne l'état de l'alimentation de la boucle.

currentloopoutput→get_module()

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

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

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

currentloopoutput→get_userData()

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

currentloopoutput→isOnline()

Vérifie si le module hébergeant la sortie sortie 4-20mA est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant la sortie sortie 4-20mA est joignable, sans déclencher d'erreur.

currentloopoutput→load(**msValidity**)

Met en cache les valeurs courantes de la sortie sortie 4-20mA, avec une durée de validité spécifiée.

currentloopoutput→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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

Met en cache les valeurs courantes de la sortie sortie 4-20mA, avec une durée de validité spécifiée.

currentloopoutput→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

currentloopoutput→nextCurrentLoopOutput()

Continue l'énumération des sortie 4-20mA commencée à l'aide de yFirstCurrentLoopOutput().

currentloopoutput→registerValueCallback(**callback**)

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

currentloopoutput→set_current(**newval**)

Modifie le courant dans la boucle, les valeurs admises sont de 3 à 21mA.

currentloopoutput→set_currentAtStartUp(**newval**)

Modifie la valeur de courant dans la boucle au démarrage du module.

currentloopoutput→set_logicalName(**newval**)

Modifie le nom logique de la sortie sortie 4-20mA.

currentloopoutput→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.

currentloopoutput→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

currentloopoutput→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.

YCurrentLoopOutput.FindCurrentLoopOutput() YCurrentLoopOutput yFindCurrentLoopOutput() YCurrentLoopOutput.FindCurrentLoopOutput() YCurrentLoopOutput.FindCurrentLoopOutput()

Permet de retrouver une sortie 4-20mA d'après un identifiant donné.

function **FindCurrentLoopOutput**(**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 sortie sortie 4-20mA soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YCurrentLoopOutput.isOnline()` pour tester si la sortie sortie 4-20mA 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence la sortie sortie 4-20mA sans ambiguïté

Retourne :

un objet de classe `YCurrentLoopOutput` qui permet ensuite de contrôler la sortie sortie 4-20mA.

YCurrentLoopOutput.FindCurrentLoopOutputInContext() YCurrentLoopOutput yFindCurrentLoopOutputInContext() YCurrentLoopOutput.FindCurrentLoopOutputInContext() YCurrentLoopOutput.FindCurrentLoopOutputInContext()

Permet de retrouver une sortie 4-20mA d'après un identifiant donné dans un Context YAPI.

function **FindCurrentLoopOutputInContext**(**yctx**, **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 sortie sortie 4-20mA soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YCurrentLoopOutput.isOnline()` pour tester si la sortie sortie 4-20mA 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence la sortie sortie 4-20mA sans ambiguïté

Retourne :

un objet de classe `YCurrentLoopOutput` qui permet ensuite de contrôler la sortie sortie 4-20mA.

YCurrentLoopOutput.FirstCurrentLoopOutput() YCurrentLoopOutput yFirstCurrentLoopOutput()
YCurrentLoopOutput.FirstCurrentLoopOutput()
YCurrentLoopOutput.FirstCurrentLoopOutput()

Commence l'énumération des sortie 4-20mA accessibles par la librairie.

function **FirstCurrentLoopOutput**( )

Utiliser la fonction `YCurrentLoopOutput.nextCurrentLoopOutput()` pour itérer sur les autres sortie 4-20mA.

Retourne :

un pointeur sur un objet `YCurrentLoopOutput`, correspondant à la première sortie 4-20mA accessible en ligne, ou `null` si il n'y a pas de sortie 4-20mA disponibles.

YCurrentLoopOutput.FirstCurrentLoopOutputInContext() YCurrentLoopOutput yFirstCurrentLoopOutputInContext() YCurrentLoopOutput.FirstCurrentLoopOutputInContext() YCurrentLoopOutput.FirstCurrentLoopOutputInContext()

Commence l'énumération des sortie 4-20mA accessibles par la librairie.

function **FirstCurrentLoopOutputInContext**(**yctx**)

Utiliser la fonction `YCurrentLoopOutput.nextCurrentLoopOutput()` pour itérer sur les autres sortie 4-20mA.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YCurrentLoopOutput`, correspondant à la première sortie 4-20mA accessible en ligne, ou `null` si il n'y a pas de sortie 4-20mA disponibles.

currentloopoutput→clearCache() YCurrentLoopOutput
currentloopoutput.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de la sortie sortie 4-20mA. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

currentloopoutput→currentMove() YCurrentLoopOutput
currentloopoutput.currentMove()
currentloopoutput.currentMove()

Déclenche une transition progressive du courant dans la boucle.

function **currentMove**(**mA_target**, **ms_duration**)

N'importe quel changement explicite de courant annulera tout processus de transition en cours.

Paramètres :

mA_target	nouvelle valeur du courant à la fin de la transition (nombre flottant, représentant le courant en mA)
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.

currentloopoutput→describe() YCurrentLoopOutput
currentloopoutput.describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de la sortie sortie 4-20mA au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant la sortie sortie 4-20mA (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

currentloopoutput→get_advertisedValue() YCurrentLoopOutput
currentloopoutput→advertisedValue()
currentloopoutput.get_advertisedValue()
currentloopoutput.get_advertisedValue()

Retourne la valeur courante de la sortie sortie 4-20mA (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :

une chaîne de caractères représentant la valeur courante de la sortie sortie 4-20mA (pas plus de 6 caractères).

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

currentloopoutput→get_current() YCurrentLoopOutput
currentloopoutput→current()
currentloopoutput.get_current()
currentloopoutput.get_current()

Retourne la valeur de consigne pour le courant de la boucle, en mA.

function **get_current**( )

Retourne :
une valeur numérique représentant la valeur de consigne pour le courant de la boucle, en mA

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

currentloopoutput→get_currentAtStartUp() YCurrentLoopOutput
currentloopoutput→currentAtStartUp()
currentloopoutput.get_currentAtStartUp()
currentloopoutput.get_currentAtStartUp()

Retourne le courant dans le boucle au démarrage du module, en mA.

function **get_currentAtStartUp**( )

Retourne :
une valeur numérique représentant le courant dans le boucle au démarrage du module, en mA

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

currentloopoutput→get_errorMessage() YCurrentLoopOutput
currentloopoutput→errorMessage()
currentloopoutput.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la sortie sortie 4-20mA.

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 sortie sortie 4-20mA.

currentloopoutput→get_errorType() YCurrentLoopOutput
currentloopoutput→errorType()
currentloopoutput.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la sortie sortie 4-20mA.

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 sortie sortie 4-20mA.

currentloopoutput→get_friendlyName() YCurrentLoopOutput
currentloopoutput→friendlyName()
currentloopoutput.get_friendlyName()

Retourne un identifiant global de la sortie sortie 4-20mA au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et de la sortie sortie 4-20mA si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel de la sortie sortie 4-20mA (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant la sortie sortie 4-20mA en utilisant les noms logiques (ex: `MyCustomName.relay1`)

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

currentloopoutput→get_functionDescriptor() YCurrentLoopOutput
currentloopoutput→functionDescriptor()
currentloopoutput.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

currentloopoutput→get_functionId() YCurrentLoopOutput
currentloopoutput→functionId()
currentloopoutput.get_functionId()

Retourne l'identifiant matériel de la sortie sortie 4-20mA, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant la sortie sortie 4-20mA (ex: `relay1`)

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

currentloopoutput→get_hardwareId() YCurrentLoopOutput
currentloopoutput→hardwareId()
currentloopoutput.get_hardwareId()

Retourne l'identifiant matériel unique de la sortie sortie 4-20mA 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 sortie sortie 4-20mA (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant la sortie sortie 4-20mA (ex: RELAYLO1-123456.relay1)

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

currentloopoutput→get_logicalName() YCurrentLoopOutput
currentloopoutput→logicalName()
currentloopoutput.get_logicalName()
currentloopoutput.get_logicalName()

Retourne le nom logique de la sortie sortie 4-20mA.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique de la sortie sortie 4-20mA.

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

currentloopoutput→get_loopPower() YCurrentLoopOutput
currentloopoutput→loopPower()
currentloopoutput.get_loopPower()
currentloopoutput.get_loopPower()

Retourne l'état de l'alimentation de la boucle.

function **get_loopPower**( )

POWEROK: la boucle est alimentée. NOPWR: la boucle n'est pas alimenté. LOWPWR: la boucle n'est pas alimenté suffisamment pour pouvoir maintenir le courant demandé (tension insuffisante)

Retourne :
une valeur parmi `Y_LOOPPOWER_NOPWR`, `Y_LOOPPOWER_LOWPWR` et `Y_LOOPPOWER_POWEROK` représentant l'état de l'alimentation de la boucle

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

currentloopoutput→get_module() YCurrentLoopOutput
currentloopoutput→module()
currentloopoutput.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

currentloopoutput→get_userData() YCurrentLoopOutput
currentloopoutput→userData()
currentloopoutput.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.

currentloopoutput→isOnline() YCurrentLoopOutput
currentloopoutput.isOnline()

Vérifie si le module hébergeant la sortie sortie 4-20mA est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache de la sortie sortie 4-20mA sont valides au moment de l'appel, le module est considéré joignable. 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 sortie sortie 4-20mA est joignable, `false` sinon

currentloopoutput→load()currentloopoutput.load() YCurrentLoopOutput

Met en cache les valeurs courantes de la sortie sortie 4-20mA, 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 occasionellement 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.

currentloopoutput→loadAttribute() YCurrentLoopOutput
currentloopoutput.loadAttribute()
currentloopoutput.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

currentloopoutput→muteValueCallbacks() YCurrentLoopOutput
currentloopoutput.muteValueCallbacks()
currentloopoutput.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

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

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

currentloopoutput→nextCurrentLoopOutput() YCurrentLoopOutput
currentloopoutput.nextCurrentLoopOutput()
currentloopoutput.nextCurrentLoopOutput()

Continue l'énumération des sortie 4-20mA commencée à l'aide de `yFirstCurrentLoopOutput()`.

function **nextCurrentLoopOutput**( )

Retourne :

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

currentloopoutput→registerValueCallback() YCurrentLoopOutput
currentloopoutput.registerValueCallback()
currentloopoutput.registerValueCallback()

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.

currentloopoutput→set_current() YCurrentLoopOutput
currentloopoutput→setCurrent()
currentloopoutput.set_current()
currentloopoutput.set_current()

Modifie le courant dans la boucle, les valeurs admises sont de 3 à 21mA.

function **set_current**(**newval**)

Attention, si la boucle n'est pas suffisamment alimentée, le courant ne pourra pas être maintenu et loopPower passera a LOWPWR.

Paramètres :

newval une valeur numérique représentant le courant dans la boucle, les valeurs admises sont de 3 à 21mA

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

currentloopoutput→set_currentAtStartUp() YCurrentLoopOutput
currentloopoutput→setCurrentAtStartUp()
currentloopoutput.set_currentAtStartUp()
currentloopoutput.set_currentAtStartUp()

Modifie la valeur de courant dans la boucle au démarrage du module.

function **set_currentAtStartUp**(**newval**)

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval une valeur numérique représentant la valeur de courant dans la boucle 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.

currentloopoutput→set_logicalName() YCurrentLoopOutput
currentloopoutput→setLogicalName()
currentloopoutput.set_logicalName()
currentloopoutput.set_logicalName()

Modifie le nom logique de la sortie sortie 4-20mA.

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 sortie sortie 4-20mA.

Retourne :

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

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

currentloopoutput→set_userData() YCurrentLoopOutput
currentloopoutput→setUserData()
currentloopoutput.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

currentloopoutput→unmuteValueCallbacks() YCurrentLoopOutput
currentloopoutput.unmuteValueCallbacks()
currentloopoutput.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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

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

currentloopoutput→wait_async() YCurrentLoopOutput
currentloopoutput.wait_async()

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 DaisyChain

L'interface YDaisyChain permet de contrôler le bon fonctionnement du lien de chaînage direct entre modules, sans passer par un hub.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_daisychain.js'></script>
cpp	#include "yocto_daisychain.h"
m	#import "yocto_daisychain.h"
pas	uses yocto_daisychain;
vb	yocto_daisychain.vb
cs	yocto_daisychain.cs
java	import com.yoctopuce.YoctoAPI.YDaisyChain;
uwp	import com.yoctopuce.YoctoAPI.YDaisyChain;
py	from yocto_daisychain import *
php	require_once('yocto_daisychain.php');
es	in HTML: <script src="../../lib/yocto_daisychain.js"></script> in node.js: require('yoctolib-es2017/yocto_daisychain.js');

Fonction globales

yFindDaisyChain(**func**)

Permet de retrouver une chaîne de modules d'après un identifiant donné.

yFindDaisyChainInContext(**yctx**, **func**)

Permet de retrouver une chaîne de modules d'après un identifiant donné dans un Context YAPI.

yFirstDaisyChain()

Commence l'énumération des chaînes de module accessibles par la librairie.

yFirstDaisyChainInContext(**yctx**)

Commence l'énumération des chaînes de module accessibles par la librairie.

Méthodes des objets YDaisyChain

daisychain→clearCache()

Invalide le cache.

daisychain→describe()

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

daisychain→get_advertisedValue()

Retourne la valeur courante de la chaîne de modules (pas plus de 6 caractères).

daisychain→get_childCount()

Retourne le nombre de sous-modules actuellement détectés.

daisychain→get_daisyState()

Retourne l'état du lien de chaînage.

daisychain→get_errorMessage()

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

daisychain→get_errorType()

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

daisychain→get_friendlyName()

Retourne un identifiant global de la chaîne de modules au format `NOM_MODULE.NOM_FONCTION`.

daisychain→get_functionDescriptor()

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

daisychain→get_functionId()

Retourne l'identifiant matériel de la chaîne de modules, sans référence au module.

daisychain→get_hardwareId()

Retourne l'identifiant matériel unique de la chaîne de modules au format SERIAL.FUNCTIONID.

daisychain→get_logicalName()

Retourne le nom logique de la chaîne de modules.

daisychain→get_module()

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

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

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

daisychain→get_requiredChildCount()

Retourne le nombre de sous-modules attendus en fonctionnement normal.

daisychain→get_userData()

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

daisychain→isOnline()

Vérifie si le module hébergeant la chaîne de modules est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant la chaîne de modules est joignable, sans déclencher d'erreur.

daisychain→load(**msValidity**)

Met en cache les valeurs courantes de la chaîne de modules, avec une durée de validité spécifiée.

daisychain→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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

Met en cache les valeurs courantes de la chaîne de modules, avec une durée de validité spécifiée.

daisychain→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

daisychain→nextDaisyChain()

Continue l'énumération des chaînes de module commencée à l'aide de yFirstDaisyChain().

daisychain→registerValueCallback(**callback**)

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

daisychain→set_logicalName(**newval**)

Modifie le nom logique de la chaîne de modules.

daisychain→set_requiredChildCount(**newval**)

Modifie le nombre de sous-modules attendus en fonctionnement normal.

daisychain→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.

daisychain→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

daisychain→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.

YDaisyChain.FindDaisyChain() YDaisyChain yFindDaisyChain()YDaisyChain.FindDaisyChain() YDaisyChain.FindDaisyChain()

Permet de retrouver une chaîne de modules d'après un identifiant donné.

```
function FindDaisyChain( 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 chaîne de modules soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDaisyChain.isOnline()` pour tester si la chaîne de modules 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence la chaîne de modules sans ambiguïté

Retourne :

un objet de classe `YDaisyChain` qui permet ensuite de contrôler la chaîne de modules.

YDaisyChain.FindDaisyChainInContext() YDaisyChain
yFindDaisyChainInContext()
YDaisyChain.FindDaisyChainInContext()
YDaisyChain.FindDaisyChainInContext()

Permet de retrouver une chaîne de modules d'après un identifiant donné dans un Context YAPI.

function **FindDaisyChainInContext**(**yctx**, **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 chaîne de modules soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDaisyChain.isOnline()` pour tester si la chaîne de modules 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence la chaîne de modules sans ambiguïté

Retourne :

un objet de classe `YDaisyChain` qui permet ensuite de contrôler la chaîne de modules.

YDaisyChain.FirstDaisyChain() YDaisyChain yFirstDaisyChain()YDaisyChain.FirstDaisyChain() YDaisyChain.FirstDaisyChain()

Commence l'énumération des chaînes de module accessibles par la librairie.

function **FirstDaisyChain**( )

Utiliser la fonction `YDaisyChain.nextDaisyChain()` pour itérer sur les autres chaînes de module.

Retourne :

un pointeur sur un objet `YDaisyChain`, correspondant à la première chaîne de modules accessible en ligne, ou `null` si il n'y a pas de chaînes de module disponibles.

YDaisyChain.FirstDaisyChainInContext() YDaisyChain yFirstDaisyChainInContext() YDaisyChain.FirstDaisyChainInContext() YDaisyChain.FirstDaisyChainInContext()

Commence l'énumération des chaînes de module accessibles par la librairie.

function **FirstDaisyChainInContext**(**yctx**)

Utiliser la fonction `YDaisyChain.nextDaisyChain()` pour itérer sur les autres chaînes de module.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YDaisyChain`, correspondant à la première chaîne de modules accessible en ligne, ou `null` si il n'y a pas de chaînes de module disponibles.

daisychain→clearCache()daisychain.clearCache() YDaisyChain

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de la chaîne de modules. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

daisychain→describe()daisychain.describe() YDaisyChain

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

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant la chaîne de modules (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

daisychain→get_advertisedValue() YDaisyChain
daisychain→advertisedValue()
daisychain.get_advertisedValue()
daisychain.get_advertisedValue()

Retourne la valeur courante de la chaîne de modules (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :

une chaîne de caractères représentant la valeur courante de la chaîne de modules (pas plus de 6 caractères).

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

daisychain→get_childCount() YDaisyChain
daisychain→childCount()
daisychain.get_childCount()
daisychain.get_childCount()

Retourne le nombre de sous-modules actuellement détectés.

function **get_childCount**( )

Retourne :
un entier représentant le nombre de sous-modules actuellement détectés

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

daisychain→get_daisyState() YDaisyChain
daisychain→daisyState()daisychain.get_daisyState()
daisychain.get_daisyState()

Retourne l'état du lien de chaînage.

function **get_daisyState**( )

Retourne :

une valeur parmi Y_DAISYSTATE_READY, Y_DAISYSTATE_IS_CHILD, Y_DAISYSTATE_FIRMWARE_MISMATCH, Y_DAISYSTATE_CHILD_MISSING et Y_DAISYSTATE_CHILD_LOST représentant l'état du lien de chaînage

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

daisychain→get_errorMessage() YDaisyChain
daisychain→errorMessage()
daisychain.get_errorMessage()

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

```
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 chaîne de modules.

daisychain→get_errorType() YDaisyChain
daisychain→errorType()daisychain.get_errorType()

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

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 chaîne de modules.

daisychain→get_friendlyName() YDaisyChain
daisychain→friendlyName()
daisychain.get_friendlyName()

Retourne un identifiant global de la chaîne de modules au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

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

Retourne :

une chaîne de caractères identifiant la chaîne de modules en utilisant les noms logiques (ex: MyCustomName.relay1)

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

daisychain→get_functionDescriptor() YDaisyChain
daisychain→functionDescriptor()
daisychain.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

daisychain→get_functionId() YDaisyChain
daisychain→functionId()daisychain.get_functionId()

Retourne l'identifiant matériel de la chaîne de modules, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant la chaîne de modules (ex: `relay1`)

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

daisychain→get_hardwareId() YDaisyChain
daisychain→hardwareId()
daisychain.get_hardwareId()

Retourne l'identifiant matériel unique de la chaîne de modules 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 chaîne de modules (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant la chaîne de modules (ex: RELAYLO1-123456.relay1)

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

daisychain→get_logicalName() YDaisyChain
daisychain→logicalName()
daisychain.get_logicalName()
daisychain.get_logicalName()

Retourne le nom logique de la chaîne de modules.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique de la chaîne de modules.

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

daisychain→get_module() YDaisyChain
daisychain→module()daisychain.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

daisychain→get_requiredChildCount() YDaisyChain
daisychain→requiredChildCount()
daisychain.get_requiredChildCount()
daisychain.get_requiredChildCount()

Retourne le nombre de sous-modules attendus en fonctionnement normal.

function **get_requiredChildCount**( )

Retourne :
un entier représentant le nombre de sous-modules attendus en fonctionnement normal

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

daisychain→get_userData() YDaisyChain
daisychain→userData()daisychain.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.

daisychain→isOnline()daisychain.isOnline() YDaisyChain

Vérifie si le module hébergeant la chaîne de modules est joignable, sans déclencher d'erreur.

function **isOnline**( )

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

daisychain→load()daisychain.load() YDaisyChain

Met en cache les valeurs courantes de la chaîne de modules, 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 occasionellement 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.

daisychain→loadAttribute()daisychain.loadAttribute() YDaisyChain daisychain.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

daisychain→muteValueCallbacks() YDaisyChain
daisychain.muteValueCallbacks()
daisychain.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

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

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

daisychain→nextDaisyChain() YDaisyChain
daisychain.nextDaisyChain()
daisychain.nextDaisyChain()

Continue l'énumération des chaînes de module commencée à l'aide de `yFirstDaisyChain()`.

function **nextDaisyChain**( )

Retourne :

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

daisychain→registerValueCallback() YDaisyChain
daisychain.registerValueCallback()
daisychain.registerValueCallback()

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.

daisychain→set_logicalName() YDaisyChain
daisychain→setLogicalName()
daisychain.set_logicalName()
daisychain.set_logicalName()

Modifie le nom logique de la chaîne de modules.

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 chaîne de modules.

Retourne :

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

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

daisychain→set_requiredChildCount() YDaisyChain
daisychain→setRequiredChildCount()
daisychain.set_requiredChildCount()
daisychain.set_requiredChildCount()

Modifie le nombre de sous-modules attendus en fonctionnement normal.

function **set_requiredChildCount**(**newval**)

Si la valeur est nulle, aucune exigence n'est posée. Si elle est non-nulle, le nombre de sous-modules détecté est vérifié au démarrage. Le status passe alors en erreur si le nombre de sous-modules ne correspond pas.

Paramètres :

newval un entier représentant le nombre de sous-modules attendus en fonctionnement normal

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

daisychain→set_userData() YDaisyChain
daisychain→setUserData()daisychain.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

daisychain→unmuteValueCallbacks() YDaisyChain
daisychain.unmuteValueCallbacks()
daisychain.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

daisychain→wait_async()daisychain.wait_async() YDaisyChain

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 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_api.js'></script>
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;
uwp	import com.yoctopuce.YoctoAPI.YModule;
py	from yocto_api import *
php	require_once('yocto_api.php');
es	in HTML: <script src="../../lib/yocto_api.js"></script> in node.js: require('yoctolib-es2017/yocto_api.js');

Fonction globales
yFindDataLogger(**func**) Permet de retrouver un enregistreur de données d'après un identifiant donné.
yFindDataLoggerInContext(**yctx**, **func**) Permet de retrouver un enregistreur de données d'après un identifiant donné dans un Context YAPI.
yFirstDataLogger() Commence l'énumération des enregistreurs de données accessibles par la librairie.
yFirstDataLoggerInContext(**yctx**) Commence l'énumération des enregistreurs de données accessibles par la librairie.
Méthodes des objets `YDataLogger`
datalogger→clearCache() Invalide le cache.
datalogger→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de l'enregistreur de données au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
datalogger→forgetAllDataStreams() Efface tout l'historique des mesures de l'enregistreur de données.
datalogger→get_advertisedValue() Retourne la valeur courante de l'enregistreur de données (pas plus de 6 caractères).
datalogger→get_autoStart() Retourne le mode d'activation automatique de l'enregistreur de données à la mise sous tension.
datalogger→get_beaconDriven() Retourne vrais si l'enregistreur de données est synchronisé avec la balise de localisation.
datalogger→get_currentRunIndex() Retourne le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active.
datalogger→get_dataSets() Retourne une liste d'objets YDataSet permettant de récupérer toutes les mesures stockées par l'enregistreur de données.

datalogger→get_dataStreams(v)

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur (ancienne méthode).

datalogger→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

datalogger→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

datalogger→get_friendlyName()

Retourne un identifiant global de l'enregistreur de données au format `NOM_MODULE.NOM_FONCTION`.

datalogger→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

datalogger→get_functionId()

Retourne l'identifiant matériel de l'enregistreur de données, sans référence au module.

datalogger→get_hardwareId()

Retourne l'identifiant matériel unique de l'enregistreur de données au format `SERIAL.FUNCTIONID`.

datalogger→get_logicalName()

Retourne le nom logique de l'enregistreur de données.

datalogger→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

datalogger→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

datalogger→get_recording()

Retourne l'état d'activation de l'enregistreur de données.

datalogger→get_timeUTC()

Retourne le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue.

datalogger→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

datalogger→isOnline()

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

datalogger→isOnline_async(callback, context)

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

datalogger→load(msValidity)

Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.

datalogger→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

datalogger→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.

datalogger→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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_beaconDriven(newval)

Modifie le mode de synchronisation de l'enregistreur de données .

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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

datalogger→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YDataLogger.FindDataLogger() YDataLogger yFindDataLogger()YDataLogger.FindDataLogger() YDataLogger.FindDataLogger()

Permet de retrouver un enregistreur de données d'après un identifiant donné.

```
function FindDataLogger( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindDataLoggerInContext() YDataLogger
yFindDataLoggerInContext()
YDataLogger.FindDataLoggerInContext()
YDataLogger.FindDataLoggerInContext()

Permet de retrouver un enregistreur de données d'après un identifiant donné dans un Context YAPI.

function **FindDataLoggerInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'enregistreur de données sans ambiguïté

Retourne :

un objet de classe `YDataLogger` qui permet ensuite de contrôler l'enregistreur de données.

YDataLogger.FirstDataLogger() YDataLogger yFirstDataLogger()YDataLogger.FirstDataLogger() YDataLogger.FirstDataLogger()

Commence l'énumération des enregistreurs de données accessibles par la librairie.

function **FirstDataLogger**( )

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.

YDataLogger.FirstDataLoggerInContext() YDataLogger
yFirstDataLoggerInContext()
YDataLogger.FirstDataLoggerInContext()
YDataLogger.FirstDataLoggerInContext()

Commence l'énumération des enregistreurs de données accessibles par la librairie.

function **FirstDataLoggerInContext**(**yctx**)

Utiliser la fonction `YDataLogger.nextDataLogger()` pour itérer sur les autres enregistreurs de données.

Paramètres :

yctx un contexte YAPI.

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→clearCache()datalogger.clearCache() YDataLogger

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'enregistreur de données. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant l'enregistreur de données (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

datalogger→forgetAllDataStreams() YDataLogger
datalogger.forgetAllDataStreams()
datalogger.forgetAllDataStreams()

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()
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()
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_beaconDriven() YDataLogger
datalogger→beaconDriven()
datalogger.get_beaconDriven()
datalogger.get_beaconDriven()

Retourne vrais si l'enregistreur de données est synchronisé avec la balise de localisation.

function **get_beaconDriven**( )

Retourne :

soit `Y_BEACONDRIVEN_OFF`, soit `Y_BEACONDRIVEN_ON`, selon vrais si l'enregistreur de données est synchronisé avec la balise de localisation

En cas d'erreur, déclenche une exception ou retourne `Y_BEACONDRIVEN_INVALID`.

datalogger→get_currentRunIndex() YDataLogger
datalogger→currentRunIndex()
datalogger.get_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()
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 pout stocker les objets YDataStream, et la méthode va les remplire 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'identifant 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() YDataLogger
datalogger→functionDescriptor()
datalogger.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

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 example `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 example 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()
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_recording() YDataLogger
datalogger→recording()datalogger.get_recording()
datalogger.get_recording()

Retourne l'état d'activation de l'enregistreur de données.

function **get_recording**( )

Retourne :

une valeur parmi `Y_RECORDING_OFF`, `Y_RECORDING_ON` et `Y_RECORDING_PENDING` représentant 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()
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.

datalogger→isOnline()datalogger.isOnline() YDataLogger

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache de l'enregistreur de données sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'enregistreur de données est joignable, `false` sinon

datalogger→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 occasionellement 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→loadAttribute()datalogger.loadAttribute() YDataLogger datalogger.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

datalogger→muteValueCallbacks() YDataLogger
datalogger.muteValueCallbacks()
datalogger.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→nextDataLogger() YDataLogger
datalogger.nextDataLogger()
datalogger.nextDataLogger()

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() YDataLogger
datalogger.registerValueCallback()
datalogger.registerValueCallback()

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()
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_beaconDriven() YDataLogger
datalogger→setBeaconDriven()
datalogger.set_beaconDriven()
datalogger.set_beaconDriven()

Modifie le mode de synchronisation de l'enregistreur de données .

function **set_beaconDriven**(**newval**)

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval soit `Y_BEACONDRIVEN_OFF`, soit `Y_BEACONDRIVEN_ON`, selon le mode de synchronisation de l'enregistreur de données

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→set_logicalName() YDataLogger
datalogger→setLogicalName()
datalogger.set_logicalName()
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()datalogger.set_recording()

Modifie l'état d'activation de l'enregistreur de données.

function **set_recording**(**newval**)

Paramètres :

newval une valeur parmi `Y_RECORDING_OFF`, `Y_RECORDING_ON` et `Y_RECORDING_PENDING` représentant 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()
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'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

datalogger→unmuteValueCallbacks() YDataLogger
datalogger.unmuteValueCallbacks()
datalogger.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.18. Séquence de données mise en forme

Un Run est un intervalle de temps pendant lequel un module est sous tension. Les objets YDataRun fournissent un accès facilité à toutes les mesures collectées durant un Run donné, y compris en permettant la lecture par mesure distantes d'un intervalle spécifié.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_datalogger.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YDataLogger = yoctolib.YDataLogger;
php     require_once('yocto_datalogger.php');
cpp     #include "yocto_datalogger.h"
m       #import "yocto_datalogger.h"
pas     uses yocto_datalogger;
vb      yocto_datalogger.vb
cs      yocto_datalogger.cs
java    import com.yoctopuce.YoctoAPI.YDataLogger;
py      from yocto_datalogger import *
```

Méthodes des objets YDataRun
datarun→get_averageValue(**measureName**, **pos**) Retourne la valeur moyenne des mesures observées au moment choisi.
datarun→get_duration() Retourne la durée (en secondes) du Run.
datarun→get_maxValue(**measureName**, **pos**) Retourne la valeur maximale des mesures observées au moment choisi.
datarun→get_measureNames() Retourne les noms des valeurs mesurées par l'enregistreur de données.
datarun→get_minValue(**measureName**, **pos**) Retourne la valeur minimale des mesures observées au moment choisi.
datarun→get_startTimeUTC() Retourne l'heure absolue du début du Run, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).
datarun→get_valueCount() Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.
datarun→get_valueInterval() Retourne l'intervalle de temps représenté par chaque valeur de ce run.
datarun→set_valueInterval(**valueInterval**) Change l'intervalle de temps représenté par chaque valeur de ce run.

datarun→get_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.

3.19. Séquence de données enregistrées

Les objets YDataSet permettent de récupérer un ensemble de mesures enregistrées correspondant à un capteur donné, pour une période choisie. Ils permettent le chargement progressif des données. Lorsque l'objet YDataSet est instancié par la fonction `get_recordedData()`, aucune donnée n'est encore chargée du module. Ce sont les appels successifs à la méthode `loadMore()` qui procèdent au chargement effectif des données depuis l'enregistreur de données.

Un résumé des mesures disponibles est disponible via la fonction `get_preview()` dès le premier appel à `loadMore()`. Les mesures elles-même sont disponibles via la fonction `get_measures()` au fur et à mesure de leur chargement.

Cette classe ne fonctionne que si le module utilise un firmware récent, car les objets YDataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_api.js'></script>
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;
uwp	import com.yoctopuce.YoctoAPI.YModule;
py	from yocto_api import *
php	require_once('yocto_api.php');
es	in HTML: <script src="../../lib/yocto_api.js"></script> in node.js: require('yoctolib-es2017/yocto_api.js');

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_measuresAt(**measure**) Retourne les mesures détaillées pour une mesure résumée précédemment retournée par `get_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.
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 disponibels, 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()
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()
dataset.get_functionId()

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

function **get_functionId**( )

Par example `temperature1`.

Retourne :

une chaîne de caractères identifiant la fonction (ex: `temperature1`)

dataset→get_hardwareId() YDataSet
dataset→hardwareId()dataset.get_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 example THRMCPL1-123456.temperature1).

Retourne :

une chaîne de caractères identifiant la fonction (ex: THRMCPL1-123456.temperature1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

dataset→get_measures() YDataSet
dataset→measures()dataset.get_measures()
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 ou la mesure a débuté - le moment ou 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_measuresAt() YDataSet
dataset→measuresAt()dataset.get_measuresAt()
dataset.get_measuresAt()

Retourne les mesures détaillées pour une mesure résumée précédemment retournée par `get_preview()`.

function **get_measuresAt**(**measure**)

Le résultat est fourni sous forme d'une liste d'objets YMeasure.

Paramètres :

measure mesure résumée extraite de la liste précédemment retournée par `get_preview()`.

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_preview() YDataSet
dataset→preview()dataset.get_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()
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()
dataset.get_startTimeUTC()

Retourne l'heure absolue du début des mesures disponibels, 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()
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()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 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.

```
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.

3.20. Séquence de données enregistrées brute

Les objets YDataStream correspondent aux séquences de mesures enregistrées brutes, directement telles qu'obtenues par l'enregistreur de données présent dans les senseurs de Yoctopuce.

Dans la plupart des cas, il n'est pas nécessaire d'utiliser les objets DataStream, car les objets YDataSet (retournés par la méthode `get_recordedData()` des senseurs et la méthode `get_dataSets()` du DataLogger) fournissent une interface plus pratique.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_api.js'></script>
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;
uwp	import com.yoctopuce.YoctoAPI.YModule;
py	from yocto_api import *
php	require_once('yocto_api.php');
es	in HTML: <script src="../../lib/yocto_api.js"></script> in node.js: require('yoctolib-es2017/yocto_api.js');

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 contenues 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()

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()
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()
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 déclanche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

Retourne :
un entier positif correspondant au nombre de colonnes.

En cas d'erreur, déclenche une exception ou retourne zéro.

datastream→get_columnNames() YDataStream
datastream→columnNames()
datastream.get_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 férquence, 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 déclanche 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()
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 sémentique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode get_columnNames().

Cette méthode déclanche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

Paramètres :

row index de l'enregistrement (ligne)

col index de la mesure (colonne)

Retourne :

un nombre décimal

En cas d'erreur, déclenche une exception ou retourne Y_DATA_INVALID.

datastream→get_dataRows() YDataStream
datastream→dataRows()datastream.get_dataRows()
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 sémentique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Cette méthode déclanche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

Retourne :
une liste d'enregistrements, chaque enregistrement étant lui-même une liste de nombres décimaux.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

datastream→get_dataSamplesIntervalMs() YDataStream
datastream→dataSamplesIntervalMs()
datastream.get_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()
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()
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()
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()
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 déclanche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

Retourne :
un entier positif correspondant au nombre d'enregistrements.

En cas d'erreur, déclenche une exception ou retourne zéro.

datastream→get_runIndex() YDataStream
datastream→runIndex()datastream.get_runIndex()
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()
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 courantegener (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 (valeure 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()
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.

3.21. 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 chanque 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>
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;
uwp	import com.yoctopuce.YoctoAPI.YDigitalIO;
py	from yocto_digitalio import *
php	require_once('yocto_digitalio.php');
es	in HTML: <script src="../../lib/yocto_digitalio.js"></script> in node.js: require('yoctolib-es2017/yocto_digitalio.js');

Fonction globales

yFindDigitalIO(**func**)

Permet de retrouver un port d'E/S digital d'après un identifiant donné.

yFindDigitalIOInContext(**yctx**, **func**)

Permet de retrouver un port d'E/S digital d'après un identifiant donné dans un Context YAPI.

yFirstDigitalIO()

Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.

yFirstDigitalIOInContext(**yctx**)

Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.

Méthodes des objets `YDigitalIO`

digitalio→clearCache()

Invalide le cache.

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_portDiags()

Retourne le diagnostique de l'état du port (Yocto-IO et Yocto-MaxiIO-V2 seulement).

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→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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.

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 polaritée 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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

digitalio→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YDigitalIO.FindDigitalIO() YDigitalIO yFindDigitalIO()YDigitalIO.FindDigitalIO() YDigitalIO.FindDigitalIO()

Permet de retrouver un port d'E/S digital d'après un identifiant donné.

function **FindDigitalIO**(**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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindDigitalIOInContext() YDigitalIO
yFindDigitalIOInContext()
YDigitalIO.FindDigitalIOInContext()
YDigitalIO.FindDigitalIOInContext()

Permet de retrouver un port d'E/S digital d'après un identifiant donné dans un Context YAPI.

function **FindDigitalIOInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

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() YDigitalIO yFirstDigitalIO()YDigitalIO.FirstDigitalIO() YDigitalIO.FirstDigitalIO()

Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.

function **FirstDigitalIO**( )

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.

YDigitalIO.FirstDigitalIOInContext() YDigitalIO
yFirstDigitalIOInContext()
YDigitalIO.FirstDigitalIOInContext()
YDigitalIO.FirstDigitalIOInContext()

Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.

function **FirstDigitalIOInContext**(**yctx**)

Utiliser la fonction `YDigitalIO.nextDigitalIO()` pour itérer sur les autres ports d'E/S digitaux.

Paramètres :

yctx un contexte YAPI.

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→clearCache()digitalio.clearCache() YDigitalIO

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du port d'E/S digital. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

digitalio→delayedPulse()digitalio.delayedPulse() YDigitalIO digitalio.delayedPulse()

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 poid 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 utilsé 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 debuggeur.

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()
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()
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 poid 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()
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 poid 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()
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 poid 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()
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 poid 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'identifant 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 example `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 example 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()
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_outputVoltage() YDigitalIO
digitalio→outputVoltage()
digitalio.get_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_portDiags() YDigitalIO
digitalio→portDiags()digitalio.get_portDiags()
digitalio.get_portDiags()

Retourne le diagnostique de l'état du port (Yocto-IO et Yocto-MaxiIO-V2 seulement).

function **get_portDiags**( )

Le bit 0 signale un court-circuit sur la sortie 0, etc. Le bit 8 indique un défaut d'alimentation, et le bit 9 indique une surchauffe (courant excessif). En fonctionnement normal, le diagnostique devrait être à zéro.

Retourne :
un entier représentant le diagnostique de l'état du port (Yocto-IO et Yocto-MaxiIO-V2 seulement)

En cas d'erreur, déclenche une exception ou retourne `Y_PORTDIAGS_INVALID`.

digitalio→get_portDirection() YDigitalIO
digitalio→portDirection()digitalio.get_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()
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()
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()
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()
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.

digitalio→isOnline()digitalio.isOnline() YDigitalIO

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du port d'E/S digital sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si le port d'E/S digital est joignable, `false` sinon

digitalio→load()digitalio.load() YDigitalIO

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

function **load**(**msValidity**)

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionellement 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→loadAttribute()digitalio.loadAttribute() YDigitalIO digitalio.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

digitalio→muteValueCallbacks() YDigitalIO
digitalio.muteValueCallbacks()
digitalio.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→nextDigitalIO()digitalio.nextDigitalIO() YDigitalIO digitalio.nextDigitalIO()

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()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 poid 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() YDigitalIO
digitalio.registerValueCallback()
digitalio.registerValueCallback()

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()
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 poid 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() YDigitalIO
digitalio→setBitOpenDrain()
digitalio.set_bitOpenDrain()
digitalio.set_bitOpenDrain()

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 poid 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()
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 poid faible est à l'index 0

bitpolarity nouvelle valeur de la plarité. 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()
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 poid 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()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() YDigitalIO
digitalio→setOutputVoltage()
digitalio.set_outputVoltage()
digitalio.set_outputVoltage()

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() YDigitalIO
digitalio→setPortDirection()
digitalio.set_portDirection()
digitalio.set_portDirection()

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() YDigitalIO
digitalio→setPortOpenDrain()
digitalio.set_portOpenDrain()
digitalio.set_portOpenDrain()

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()
digitalio.set_portPolarity()

Modifie la polaritée 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 polaritée 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()
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'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

digitalio→toggle_bitState()digitalio.toggle_bitState() YDigitalIO digitalio.toggle_bitState()

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 poid 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→unmuteValueCallbacks() YDigitalIO
digitalio.unmuteValueCallbacks()
digitalio.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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.

3.22. Interface de la fonction Display

L'interface de contrôle des écrans Yoctopuce est conçue pour afficher facilement des informations et des images. Le module est capable de gérer seul la superposition de plusieurs couches graphiques, qui peuvent être dessinées individuellement, sans affichage immédiat, puis librement positionnées sur l'écran. Il est aussi capable de rejouer des séquences de commandes pré-enregistrées (animations).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_display.js'></script>
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;
uwp	import com.yoctopuce.YoctoAPI.YDisplay;
py	from yocto_display import *
php	require_once('yocto_display.php');
es	in HTML: <script src="../../lib/yocto_display.js"></script> in node.js: require('yoctolib-es2017/yocto_display.js');

Fonction globales

yFindDisplay(**func**)

Permet de retrouver un ecran d'après un identifiant donné.

yFindDisplayInContext(**yctx**, **func**)

Permet de retrouver un ecran d'après un identifiant donné dans un Context YAPI.

yFirstDisplay()

Commence l'énumération des écran accessibles par la librairie.

yFirstDisplayInContext(**yctx**)

Commence l'énumération des écran accessibles par la librairie.

Méthodes des objets `YDisplay`

display→clearCache()

Invalide le cache.

display→copyLayerContent(**srcLayerId**, **dstLayerId**)

Copie le contentu d'un couche d'affichage vers une autre couche.

display→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'ecran 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'ecran (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'ecran est alimenté, faux sinon.

display→get_errorMessage()
Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'ecran.

display→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'ecran.

display→get_friendlyName()
Retourne un identifiant global de l'ecran 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'ecran, sans référence au module.

display→get_hardwareId()
Retourne l'identifiant matériel unique de l'ecran 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'ecran.

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'ecran est joignable, sans déclencher d'erreur.

display→isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant l'ecran est joignable, sans déclencher d'erreur.

display→load(**msValidity**)
Met en cache les valeurs courantes de l'ecran, avec une durée de validité spécifiée.

display→loadAttribute(**attrName**)
Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

display→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes de l'ecran, avec une durée de validité spécifiée.

display→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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 luminositéde l'écran.

display→set_enabled(**newval**)

Modifie l'état d'activité de l'écran.

display→set_logicalName(**newval**)

Modifie le nom logique de l'ecran.

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 contentu de deux couches d'affichage.

display→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

display→upload(**pathname**, **content**)

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

display→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YDisplay.FindDisplay() YDisplay yFindDisplay()YDisplay.FindDisplay() YDisplay.FindDisplay()

Permet de retrouver un ecran d'après un identifiant donné.

```
function FindDisplay( 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindDisplayInContext() YDisplay
yFindDisplayInContext()
YDisplay.FindDisplayInContext()
YDisplay.FindDisplayInContext()

Permet de retrouver un ecran d'après un identifiant donné dans un Context YAPI.

function **FindDisplayInContext**(**yctx**, **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 :

yctx un contexte YAPI

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() YDisplay yFirstDisplay()YDisplay.FirstDisplay() YDisplay.FirstDisplay()

Commence l'énumération des écran accessibles par la librairie.

function **FirstDisplay**( )

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.

YDisplay.FirstDisplayInContext() YDisplay
yFirstDisplayInContext()
YDisplay.FirstDisplayInContext()
YDisplay.FirstDisplayInContext()

Commence l'énumération des écran accessibles par la librairie.

function **FirstDisplayInContext**(**yctx**)

Utiliser la fonction `YDisplay.nextDisplay()` pour itérer sur les autres écran.

Paramètres :

yctx un contexte YAPI.

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→clearCache()display.clearCache() YDisplay

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'ecran. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

display→copyLayerContent() YDisplay
display.copyLayerContent()
display.copyLayerContent()

Copie le contentu d'un couche d'affichage vers une autre couche.

function **copyLayerContent**(**srcLayerId**, **dstLayerId**)

La couleur et la transparence de tous les pixels de la couche de destination sont changés pour correspondre à la couche source. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

Paramètres :

srcLayerId l'identifiant de la couche d'origine (un chiffre parmi 0..layerCount-1)

dstLayerId l'identifiant de la couche de destination (un chiffre parmi 0..layerCount-1)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→describe()display.describe() YDisplay

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'ecran au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :

une chaîne de caractères décrivant l'ecran (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

display→fade()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()
display.get_advertisedValue()

Retourne la valeur courante de l'ecran (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante de l'ecran (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()
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()
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()
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()
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()
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'ecran.

```
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'ecran.

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'ecran.

```
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'ecran.

display→get_friendlyName() YDisplay
display→friendlyName()display.get_friendlyName()

Retourne un identifiant global de l'ecran au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et de l'ecran si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel de l'ecran (par exemple: MyCustomName.relay1)

Retourne :
une chaîne de caractères identifiant l'ecran en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

display→get_functionDescriptor() YDisplay
display→functionDescriptor()
display.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

display→get_functionId() YDisplay
display→functionId()display.get_functionId()

Retourne l'identifiant matériel de l'ecran, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant l'ecran (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 example 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()
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()
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()
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()
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_orientation() YDisplay
display→orientation()display.get_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()
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'ecran est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache de l'ecran sont valides au moment de l'appel, le module est considéré joignable. 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'ecran est joignable, `false` sinon

display→load()display.load() YDisplay

Met en cache les valeurs courantes de l'ecran, 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 occasionellement 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→loadAttribute()display.loadAttribute() YDisplay display.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

display→muteValueCallbacks() YDisplay
display.muteValueCallbacks()
display.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→newSequence()display.newSequence() YDisplay display.newSequence()

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
display.nextDisplay()

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 display.pauseSequence()

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 display.playSequence()

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() YDisplay
display.registerValueCallback()
display.registerValueCallback()

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()display.resetAll() YDisplay

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

function **resetAll**( )

Utiliser cette fonction dans une sequence va tuer stopper l'affichage de la sequence: ne pas utiliser cette fonction pour réinitialiser l'écran au début d'une séquence.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→saveSequence()display.saveSequence() YDisplay display.saveSequence()

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()
display.set_brightness()

Modifie la luminositéde l'écran.

function **set_brightness**(**newval**)

Le paramêtre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la luminositéde l'écran

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→set_enabled() YDisplay
display→setEnabled()display.set_enabled()
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()
display.set_logicalName()

Modifie le nom logique de l'ecran.

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'ecran.

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()
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()
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 display.stopSequence()

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() YDisplay
display.swapLayerContent()
display.swapLayerContent()

Permute le contentu 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→unmuteValueCallbacks() YDisplay
display.unmuteValueCallbacks()
display.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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()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.

3.23. 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>
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;
uwp	import com.yoctopuce.YoctoAPI.YDisplay;
py	from yocto_display import *
php	require_once('yocto_display.php');
es	in HTML: <script src="../../lib/yocto_display.js"></script> in node.js: require('yoctolib-es2017/yocto_display.js');

Méthodes des objets YDisplayLayer
displaylayer→clear()
Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.
displaylayer→clearConsole()
Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.
displaylayer→consoleOut(**text**)
Affiche un message dans la zone de console, et déplace le curseur de la console à la fin du texte.
displaylayer→drawBar(**x1**, **y1**, **x2**, **y2**)
Dessine un rectangle plein à une position spécifiée.
displaylayer→drawBitmap(**x**, **y**, **w**, **bitmap**, **bgcol**)
Dessine un bitmap à la position spécifiée de la couche.
displaylayer→drawCircle(**x**, **y**, **r**)
Dessine un cercle vide à une position spécifiée.
displaylayer→drawDisc(**x**, **y**, **r**)
Dessine un disque plein à une position spécifiée.
displaylayer→drawImage(**x**, **y**, **imagename**)
Dessine une image GIF à la position spécifiée de la couche.
displaylayer→drawPixel(**x**, **y**)
Dessine un pixel unique à une position spécifiée.
displaylayer→drawRect(**x1**, **y1**, **x2**, **y2**)
Dessine un rectangle vide à une position spécifiée.
displaylayer→drawText(**x**, **y**, **anchor**, **text**)
Affiche un texte à la position spécifiée de la couche.
displaylayer→get_display()
Retourne l'YDisplay parent.
displaylayer→get_displayHeight()
Retourne la hauteur de l'écran, en pixels.
displaylayer→get_displayWidth()

Retourne la largeur de l'écran, en pixels.

displaylayer→get_layerHeight()

Retourne la hauteur des couches affichables, en pixels.

displaylayer→get_layerWidth()

Retourne la largeur des couches affichables, en pixels.

displaylayer→hide()

Cache la couche de dessin.

displaylayer→lineTo(**x**, **y**)

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

displaylayer→moveTo(**x**, **y**)

Déplace le point de dessin courant de cette couche à la position spécifiée.

displaylayer→reset()

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

displaylayer→selectColorPen(**color**)

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

displaylayer→selectEraser()

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de copie d'images bitmaps.

displaylayer→selectFont(**fontname**)

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

displaylayer→selectGrayPen(**graylevel**)

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

displaylayer→setAntialiasingMode(**mode**)

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

displaylayer→setConsoleBackground(**bgcol**)

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

displaylayer→setConsoleMargins(**x1**, **y1**, **x2**, **y2**)

Configure les marges d'affichage pour la fonction `consoleOut`.

displaylayer→setConsoleWordWrap(**wordwrap**)

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

displaylayer→setLayerPosition(**x**, **y**, **scrollTime**)

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

displaylayer→unhide()

Affiche la couche.

displaylayer→clear()displaylayer.clear() YDisplayLayer displaylayer.clear()

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 le 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() YDisplayLayer
displaylayer.clearConsole()
displaylayer.clearConsole()

Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.

function **clearConsole**( )

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 displaylayer.consoleOut()

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() 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() YDisplayLayer
displaylayer.drawBitmap()displaylayer.drawBitmap()

Dessine un bitmap à la position spécifiée de la couche.

function **drawBitmap**(**x**, **y**, **w**, **bitmap**, **bgcol**)

Le bitmap est passé sous forme d'un objet binaire, où chaque bit correspond à un pixel, de gauche à droite et de haut en bas. Le bit de poids fort de chaque octet correspond au pixel de gauche, et le bit de poids faible au pixel le plus à droite. Les bits à 1 sont dessinés avec la couleur active de la couche. Les bits à 0 avec la couleur de fond spécifiée, sauf si la valeur -1 a été choisie, auquel cas ils ne sont pas dessinés (ils sont considérés comme transparents). Chaque ligne commence sur un nouvel octet. La hauteur du bitmap est donnée implicitement par la taille de l'objet binaire.

Paramètres :

x	la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du bitmap
y	la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du bitmap
w	la largeur du bitmap, en pixels
bitmap	l'objet binaire contenant le bitmap
bgcol	le niveau de gris à utiliser pour les bits à zéro (0 = noir, 255 = blanc), ou -1 pour lasser les pixels inchangés

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→drawCircle()displaylayer.drawCircle() YDisplayLayer displaylayer.drawCircle()

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 displaylayer.drawDisc()

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 displaylayer.drawImage()

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 displaylayer.drawPixel()

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 displaylayer.drawRect()

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 displaylayer.drawText()

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()
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()
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()
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()
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()
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 displaylayer.hide()

Cache la couche de dessin.

```
function hide( )
```

L'etat 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 displaylayer.lineTo()

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 displaylayer.moveTo()

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 displaylayer.reset()

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() YDisplayLayer
displaylayer.selectColorPen()
displaylayer.selectColorPen()

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() YDisplayLayer
displaylayer.selectEraser()displaylayer.selectEraser()

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de copie d'images bitmaps.

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 displaylayer.selectFont()

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() YDisplayLayer
displaylayer.selectGrayPen()
displaylayer.selectGrayPen()

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

function **selectGrayPen**(**graylevel**)

Le niveau de gris est fourni sous forme d'un chiffre allant de 0 (noir) à 255 (blanc, ou la couleur la plus claire de l'écran, quelle qu'elle soit). Pour les écrans monochromes (sans niveaux de gris), tout valeur inférieure à 128 conduit à un point noir, et toue 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() YDisplayLayer
displaylayer.setAntialiasingMode()
displaylayer.setAntialiasingMode()

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

function **setAntialiasingMode**(**mode**)

L'anti-aliasing est atténue la pixelisation des images lorsqu'on regarde l'écran depuis une distance suffisante, mais peut aussi donner parfois une impression de flou lorsque l'écran est regardé de très près. Au final, c'est un choix esthétique qui vous revient. L'anti-aliasing est activé par défaut pour les écrans en niveaux de gris et les écrans couleurs, mais vous pouvez le désactiver si vous préférez. Ce réglage n'a pas d'effet sur les écrans monochromes.

Paramètres :

mode `true` pour activer l'antialiasing, `false` pour le désactiver.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→setConsoleBackground() YDisplayLayer
displaylayer.setConsoleBackground()
displaylayer.setConsoleBackground()

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() YDisplayLayer
displaylayer.setConsoleMargins()
displaylayer.setConsoleMargins()

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() YDisplayLayer
displaylayer.setConsoleWordWrap()
displaylayer.setConsoleWordWrap()

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() YDisplayLayer
displaylayer.setLayerPosition()
displaylayer.setLayerPosition()

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 displaylayer.unhide()

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.

3.24. Interface de contrôle de l'alimentation

La librairie de programmation Yoctopuce permet de contrôler la source d'alimentation qui doit être utilisée pour les fonctions du module consommant beaucoup de courant. Le module est par ailleurs capable de couper automatiquement l'alimentation externe lorsqu'il détecte que la tension a trop chuté (batterie épuisée).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_dualpower.js'></script>
cpp	#include "yocto_dualpower.h"
m	#import "yocto_dualpower.h"
pas	uses yocto_dualpower;
vb	yocto_dualpower.vb
cs	yocto_dualpower.cs
java	import com.yoctopuce.YoctoAPI.YDualPower;
uwp	import com.yoctopuce.YoctoAPI.YDualPower;
py	from yocto_dualpower import *
php	require_once('yocto_dualpower.php');
es	in HTML: <script src="../../lib/yocto_dualpower.js"></script> in node.js: require('yoctolib-es2017/yocto_dualpower.js');

Fonction globales

yFindDualPower(**func**)

Permet de retrouver un contrôle d'alimentation d'après un identifiant donné.

yFindDualPowerInContext(**yctx**, **func**)

Permet de retrouver un contrôle d'alimentation d'après un identifiant donné dans un Context YAPI.

yFirstDualPower()

Commence l'énumération des contrôles d'alimentation accessibles par la librairie.

yFirstDualPowerInContext(**yctx**)

Commence l'énumération des contrôles d'alimentation accessibles par la librairie.

Méthodes des objets `YDualPower`

dualpower→clearCache()

Invalide le cache.

dualpower→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'alimentation au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

dualpower→get_advertisedValue()

Retourne la valeur courante du contrôle d'alimentation (pas plus de 6 caractères).

dualpower→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

dualpower→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

dualpower→get_extVoltage()

Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.

dualpower→get_friendlyName()

Retourne un identifiant global du contrôle d'alimentation au format `NOM_MODULE.NOM_FONCTION`.

dualpower→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

dualpower→get_functionId()

Retourne l'identifiant matériel du contrôle d'alimentation, sans référence au module.

dualpower→get_hardwareId()

Retourne l'identifiant matériel unique du contrôle d'alimentation au format SERIAL.FUNCTIONID.

dualpower→get_logicalName()

Retourne le nom logique du contrôle d'alimentation.

dualpower→get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

dualpower→get_module_async(**callback**, **context**)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

dualpower→get_powerControl()

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

dualpower→get_powerState()

Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

dualpower→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

dualpower→isOnline()

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

dualpower→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

dualpower→load(**msValidity**)

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

dualpower→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

dualpower→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

dualpower→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

dualpower→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YDualPower.FindDualPower() YDualPower yFindDualPower()YDualPower.FindDualPower() YDualPower.FindDualPower()

Permet de retrouver un contrôle d'alimentation d'après un identifiant donné.

```
function FindDualPower( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindDualPowerInContext() YDualPower
yFindDualPowerInContext()
YDualPower.FindDualPowerInContext()
YDualPower.FindDualPowerInContext()

Permet de retrouver un contrôle d'alimentation d'après un identifiant donné dans un Context YAPI.

function **FindDualPowerInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le contrôle d'alimentation sans ambiguïté

Retourne :

un objet de classe `YDualPower` qui permet ensuite de contrôler le contrôle d'alimentation.

YDualPower.FirstDualPower() YDualPower yFirstDualPower()YDualPower.FirstDualPower() YDualPower.FirstDualPower()

Commence l'énumération des contrôles d'alimentation accessibles par la librairie.

function **FirstDualPower**( )

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.

YDualPower.FirstDualPowerInContext() YDualPower
yFirstDualPowerInContext()
YDualPower.FirstDualPowerInContext()
YDualPower.FirstDualPowerInContext()

Commence l'énumération des contrôles d'alimentation accessibles par la librairie.

function **FirstDualPowerInContext**(**yctx**)

Utiliser la fonction `YDualPower.nextDualPower()` pour itérer sur les autres contrôles d'alimentation.

Paramètres :

yctx un contexte YAPI.

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→clearCache()dualpower.clearCache() YDualPower

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du contrôle d'alimentation. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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()
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()
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'identifant 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() YDualPower
dualpower→functionDescriptor()
dualpower.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

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 example `relay1`.

Retourne :

une chaîne de caractères identifiant le contrôle d'alimentation (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

dualpower→get_hardwareId() YDualPower
dualpower→hardwareId()dualpower.get_hardwareId()

Retourne l'identifiant matériel unique du contrôle d'alimentation au format SERIAL.FUNCTIONID.

function **get_hardwareId**( )

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du contrôle d'alimentation (par example 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()
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_powerControl() YDualPower
dualpower→powerControl()
dualpower.get_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()
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.

dualpower→isOnline()dualpower.isOnline() YDualPower

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du contrôle d'alimentation sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le contrôle d'alimentation est joignable, `false` sinon

dualpower→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 occasionellement 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→loadAttribute()dualpower.loadAttribute() YDualPower dualpower.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

dualpower→muteValueCallbacks() YDualPower
dualpower.muteValueCallbacks()
dualpower.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

dualpower→nextDualPower() YDualPower
dualpower.nextDualPower()
dualpower.nextDualPower()

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() YDualPower
dualpower.registerValueCallback()
dualpower.registerValueCallback()

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() YDualPower
dualpower→setLogicalName()
dualpower.set_logicalName()
dualpower.set_logicalName()

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() YDualPower
dualpower→setPowerControl()
dualpower.set_powerControl()
dualpower.set_powerControl()

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→unmuteValueCallbacks() YDualPower
dualpower.unmuteValueCallbacks()
dualpower.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.25. Interface de la fonction Files

L'interface de stockage de fichiers permet de stocker des fichiers sur certains modules, par exemple pour personnaliser un service web (dans le cas d'un module connecté au réseau) ou pour ajouter un police de caractères (dans le cas d'un module d'affichage).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_files.js'></script>
cpp	#include "yocto_files.h"
m	#import "yocto_files.h"
pas	uses yocto_files;
vb	yocto_files.vb
cs	yocto_files.cs
java	import com.yoctopuce.YoctoAPI.YFiles;
uwp	import com.yoctopuce.YoctoAPI.YFiles;
py	from yocto_files import *
php	require_once('yocto_files.php');
es	in HTML: <script src="../../lib/yocto_files.js"></script> in node.js: require('yoctolib-es2017/yocto_files.js');

Fonction globales

yFindFiles(**func**)

Permet de retrouver un système de fichier d'après un identifiant donné.

yFindFilesInContext(**yctx**, **func**)

Permet de retrouver un système de fichier d'après un identifiant donné dans un Context YAPI.

yFirstFiles()

Commence l'énumération des système de fichier accessibles par la librairie.

yFirstFilesInContext(**yctx**)

Commence l'énumération des système de fichier accessibles par la librairie.

Méthodes des objets `YFiles`

files→clearCache()

Invalide le cache.

files→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

files→download(**pathname**)

Télécharge le fichier choisi du filesystème et retourne son contenu.

files→download_async(**pathname**, **callback**, **context**)

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

files→fileExist(**filename**)

Test si un fichier esit dans le système de fichier du module.

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→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

files→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

files→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

files→upload(pathname, content)

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

files→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YFiles.FindFiles() YFiles yFindFiles()YFiles.FindFiles()YFiles.FindFiles()

Permet de retrouver un système de fichier d'après un identifiant donné.

```
function FindFiles( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindFilesInContext() YFiles yFindFilesInContext()YFiles.FindFilesInContext() YFiles.FindFilesInContext()

Permet de retrouver un système de fichier d'après un identifiant donné dans un Context YAPI.

function **FindFilesInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le système de fichier sans ambiguïté

Retourne :

un objet de classe `YFiles` qui permet ensuite de contrôler le système de fichier.

YFiles.FirstFiles() YFiles yFirstFiles()YFiles.FirstFiles()YFiles.FirstFiles()

Commence l'énumération des système de fichier accessibles par la librairie.

```
function FirstFiles( )
```

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.

YFiles.FirstFilesInContext() YFiles yFirstFilesInContext()YFiles.FirstFilesInContext() YFiles.FirstFilesInContext()

Commence l'énumération des système de fichier accessibles par la librairie.

function **FirstFilesInContext**(**yctx**)

Utiliser la fonction `YFiles.nextFiles()` pour itérer sur les autres système de fichier.

Paramètres :

yctx un contexte YAPI.

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→clearCache()files.clearCache() YFiles

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du système de fichier. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le système de fichier (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

files→download()files.download()files.download() YFiles

Télécharge le fichier choisi du filesystème 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→fileExist()files.fileExist()files.fileExist() YFiles

Test si un fichier esit dans le système de fichier du module.

function **fileExist**(**filename**)

Paramètres :

filename le nom de fichier.

Retourne :

vrais si le fichier existe, et faux is le fichier n'existe pas.

En cas d'erreur, déclenche une exception.

files→format_fs()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()
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()
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()
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'identifant 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 example `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 example 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()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()
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_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→load()files.load() YFiles

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

function **load**(**msValidity**)

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionellement 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→loadAttribute()files.loadAttribute() YFiles files.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

files→muteValueCallbacks() YFiles
files.muteValueCallbacks()files.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→nextFiles()files.nextFiles()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() YFiles
files.registerValueCallback()
files.registerValueCallback()

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()files.remove() YFiles

Efface un fichier, spécifié par son path complet, du système de fichier.

function **remove**(**pathname**)

A cause de la fragmentation, l'effacement d'un fichier ne libère pas toujours la totalité de l'espace qu'il occuppe. 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()
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'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

files→unmuteValueCallbacks() YFiles
files.unmuteValueCallbacks()
files.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→upload()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.

3.26. Interface de contrôle pour la mise à jour de firmware

La classe YFirmwareUpdate permet de contrôler la mise à jour du de firmware d'un module Yoctopuce. Cette classe ne doit pas être instancier directement, mais est retournée par la méthode `updateFirmware` de l'un objet YModule.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_api.js'></script>
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;
uwp	import com.yoctopuce.YoctoAPI.YModule;
py	from yocto_api import *
php	require_once('yocto_api.php');
es	in HTML: <script src="../../lib/yocto_api.js"></script> in node.js: require('yoctolib-es2017/yocto_api.js');

Fonction globales
yCheckFirmware(**serial**, **path**, **minrelease**) Teste si le fichier byn est valide pour le module.
yGetAllBootLoaders() Retourne la liste des modules en mode "mise à jour".
yGetAllBootLoadersInContext(**yctx**) Retourne la liste des modules en mode "mise à jour".
Méthodes des objets `YFirmwareUpdate`
firmwareupdate→get_progress() Retourne l'état d'avancement de la mise à jour de firmware, sur une échelle de 0 à 100.
firmwareupdate→get_progressMessage() le dernier message de la mise à jour de firmware.
firmwareupdate→startUpdate() Démarre la mise à jour de firmware.

YFirmwareUpdate.CheckFirmware() YFirmwareUpdate yCheckFirmware() YFirmwareUpdate.CheckFirmware()

Teste si le fichier byn est valide pour le module.

function **CheckFirmware**(**serial**, **path**, **minrelease**)

Il est possible de passer un répertoire qui contient plusieurs fichiers byn. Dans ce cas cette méthode retourne le path du fichier byn compatible le plus récent. Cette fonction ignore les firmwares qui sont plus anciens que minrelease.

Paramètres :

serial	le numéro de série du module à mettre à jour
path	le path sur un fichier byn ou un répertoire contenant plusieurs fichiers byn
minrelease	un entier positif

Retourne :

le path du fichier byn à utiliser, ou une chaîne vide si aucun firmware plus récent n'est disponible En cas d'erreur, retourne une chaîne de caractère qui comment par "error:".

YFirmwareUpdate.GetAllBootLoaders() YFirmwareUpdate yGetAllBootLoaders() YFirmwareUpdate.GetAllBootLoaders()

Retourne la liste des modules en mode "mise à jour".

function **GetAllBootLoaders**( )

Seuls les modules connectés en USB sont listés. Pour les modules connectés à un YoctoHub, il faut se connecter à l'interface web du YoctoHub.

Retourne :
un tableau de chaînes de caractères contenant les numéros de série des modules en attente de "mise à jour"

YFirmwareUpdate.GetAllBootLoadersInContext() YFirmwareUpdate yGetAllBootLoadersInContext() YFirmwareUpdate.GetAllBootLoadersInContext()

Retourne la liste des modules en mode "mise à jour".

```
function GetAllBootLoadersInContext( yctx)
```

Seuls les modules connectés en USB sont listés. Pour les modules connectés à un YoctoHub, il faut se connecter à l'interface web du YoctoHub.

Paramètres :

yctx un contexte YAPI.

Retourne :

un tableau de chaînes de caractères contenant les numéros de série des modules en attente de "mise à jour"

firmwareupdate→get_progress() YFirmwareUpdate
firmwareupdate→progress()
firmwareupdate.get_progress()
firmwareupdate.get_progress()

Retourne l'état d'avancement de la mise à jour de firmware, sur une échelle de 0 à 100.

function **get_progress**( )

A l'instanciation de l'objet l'avancement est nul. Au fur et à mesure l'avancement progresse pour atteindre la valeur 100. Quand la valeur de 100 est retourné la mise à jour s'est terminée avec succès. En cas d'erreur pendant la mise à jour une valeur négative est retournée et la description de l'erreur peu être obtenu avec la méthode `get_progressMessage`.

Retourne :

un nombre entier entre 0 et 100 représentant l'avancement de la mise à jour du firmware, ou un code d'erreur négatif en cas de problème.

firmwareupdate→get_progressMessage() YFirmwareUpdate
firmwareupdate→progressMessage()
firmwareupdate.get_progressMessage()
firmwareupdate.get_progressMessage()

le dernier message de la mise à jour de firmware.

function **get_progressMessage**( )

En cas d'erreur durant la mise à jour le message d'erreur est retourné.

Retourne :
un chaîne de caractère avec le dernier message, ou le message d'erreur si la mise à jour n'a pas réussi

firmwareupdate→startUpdate() YFirmwareUpdate
firmwareupdate.startUpdate()
firmwareupdate.startUpdate()

Démarre la mise à jour de firmware.

function **startUpdate**( )

La méthode démarre en arrière plan le processus de mise à jour de firmware. Cet méthode rend la main immédiatement. l'état d'avancement de la mise à jour peut être suivit à l'aide des méthodes `get_progress()` et `get_progressMessage()`.

Retourne :

un nombre entier entre 0 et 100 représentant l'avancement de la mise à jour du firmware, ou un code d'erreur négatif en cas de problème.

En cas d'erreur, un code d'erreur négatif.

3.27. Interface de la fonction GenericSensor

La classe YGenericSensor permet de lire et de configurer les transducteurs de signaux Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, elle permet de configurer une conversion automatique entre le signal mesuré et la grandeur physique représentée.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_genericsensor.js'></script>
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;
uwp     import com.yoctopuce.YoctoAPI.YGenericSensor;
py      from yocto_genericsensor import *
php     require_once('yocto_genericsensor.php');
es      in HTML: <script src="../../lib/yocto_genericsensor.js"></script>
        in node.js: require('yoctolib-es2017/yocto_genericsensor.js');
```

Fonction globales
yFindGenericSensor(**func**) Permet de retrouver un capteur générique d'après un identifiant donné.
yFindGenericSensorInContext(**yctx**, **func**) Permet de retrouver un capteur générique d'après un identifiant donné dans un Context YAPI.
yFirstGenericSensor() Commence l'énumération des capteurs génériques accessibles par la librairie.
yFirstGenericSensorInContext(**yctx**) 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→clearCache() Invalide le cache.
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_dataLogger() Retourne l'objet YDataLogger du module qui héberge le senseur.
genericsensor→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

genericsensor→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

genericsensor→get_friendlyName()

Retourne un identifiant global du capteur générique au format NOM_MODULE.NOM_FONCTION.

genericsensor→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

genericsensor→get_functionId()

Retourne l'identifiant matériel du capteur générique, sans référence au module.

genericsensor→get_hardwareId()

Retourne l'identifiant matériel unique du capteur générique au format SERIAL.FUNCTIONID.

genericsensor→get_highestValue()

Retourne la valeur maximale observée pour la measure 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 measure 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_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

genericsensor→get_signalBias()

Retourne le biais du signal électrique pour la correction du point zéro.

genericsensor→get_signalRange()

Retourne la plage de signal électrique utilisé par le capteur.

genericsensor→get_signalSampling()

Retourne la méthode d'échantillonnage du signal utilisée.

genericsensor→get_signalUnit()

Retourne l'unité du signal électrique utilisé par le capteur.

genericsensor→get_signalValue()

Retourne la valeur actuelle du signal électrique mesuré par le capteur.

genericsensor→get_unit()

Retourne l'unité dans laquelle la measure 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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

genericsensor→load(msValidity)

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

genericsensor→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

genericsensor→nextGenericSensor()

Continue l'énumération des capteurs génériques commencée à l'aide de `yFirstGenericSensor()`.

genericsensor→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

genericsensor→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

genericsensor→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

genericsensor→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

genericsensor→set_logicalName(newval)

Modifie le nom logique du capteur générique.

genericsensor→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

genericsensor→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

genericsensor→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

genericsensor→set_signalBias(newval)

Modifie le biais du signal électrique pour la correction du point zéro.

genericsensor→set_signalRange(newval)

Modifie la plage de signal électrique utilisé par le capteur.

genericsensor→set_signalSampling(newval)

Modifie la méthode d'échantillonnage du signal à utiliser.

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→startDataLogger()

Démarre l'enregistreur de données du module.

genericsensor→stopDataLogger()

Arrête l'enregistreur de données du module.

genericsensor→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

genericsensor→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

genericsensor→zeroAdjust()

Ajuste le biais du signal de sorte à ce que la valeur actuelle du signal soit interprétée comme zéro (tare).

YGenericSensor.FindGenericSensor() YGenericSensor
yFindGenericSensor()
YGenericSensor.FindGenericSensor()
YGenericSensor.FindGenericSensor()

Permet de retrouver un capteur générique d'après un identifiant donné.

```
function FindGenericSensor( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindGenericSensorInContext() YGenericSensor yFindGenericSensorInContext() YGenericSensor.FindGenericSensorInContext() YGenericSensor.FindGenericSensorInContext()

Permet de retrouver un capteur générique d'après un identifiant donné dans un Context YAPI.

function **FindGenericSensorInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur générique sans ambiguïté

Retourne :

un objet de classe `YGenericSensor` qui permet ensuite de contrôler le capteur générique.

YGenericSensor.FirstGenericSensor() YGenericSensor
yFirstGenericSensor()
YGenericSensor.FirstGenericSensor()
YGenericSensor.FirstGenericSensor()

Commence l'énumération des capteurs génériques accessibles par la librairie.

function **FirstGenericSensor**( )

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.

YGenericSensor.FirstGenericSensorInContext() YGenericSensor yFirstGenericSensorInContext() YGenericSensor.FirstGenericSensorInContext() YGenericSensor.FirstGenericSensorInContext()

Commence l'énumération des capteurs génériques accessibles par la librairie.

```
function FirstGenericSensorInContext( yctx)
```

Utiliser la fonction `YGenericSensor.nextGenericSensor()` pour itérer sur les autres capteurs génériques.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YGenericSensor`, correspondant au premier capteur générique accessible en ligne, ou `null` si il n'y a pas de capteurs génériques disponibles.

genericsensor→calibrateFromPoints() YGenericSensor
genericsensor.calibrateFromPoints()
genericsensor.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→clearCache() YGenericSensor
genericsensor.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur générique. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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()
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()
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() YGenericSensor
genericsensor→currentValue()
genericsensor.get_currentValue()
genericsensor.get_currentValue()

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_dataLogger() YGenericSensor
genericsensor→dataLogger()
genericsensor.get_dataLogger()
genericsensor.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

genericsensor→get_errorMessage() YGenericSensor
genericsensor→errorMessage()
genericsensor.get_errorMessage()

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'identifant matériel du capteur générique (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le capteur générique en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

genericsensor→get_functionDescriptor() YGenericSensor
genericsensor→functionDescriptor()
genericsensor.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

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() YGenericSensor
genericsensor→functionId()
genericsensor.get_functionId()

Retourne l'identifiant matériel du capteur générique, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur générique (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

genericsensor→get_hardwareId() YGenericSensor
genericsensor→hardwareId()
genericsensor.get_hardwareId()

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 example 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()
genericsensor.get_highestValue()

Retourne la valeur maximale observée pour la measure depuis le démarrage du module.

function **get_highestValue**( )

Retourne :

une valeur numérique représentant la valeur maximale observée pour la measure 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()
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()
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()
genericsensor.get_lowestValue()

Retourne la valeur minimale observée pour la measure depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :

une valeur numérique représentant la valeur minimale observée pour la measure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

genericsensor→get_module() YGenericSensor
genericsensor→module()
genericsensor.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`

genericsensor→get_recordedData() YGenericSensor
genericsensor→recordedData()
genericsensor.get_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**)

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 find de l'intercalle 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()
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()
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_sensorState() YGenericSensor
genericsensor→sensorState()
genericsensor.get_sensorState()
genericsensor.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :

un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

genericsensor→get_signalBias() YGenericSensor
genericsensor→signalBias()
genericsensor.get_signalBias()
genericsensor.get_signalBias()

Retourne le biais du signal électrique pour la correction du point zéro.

function **get_signalBias**( )

Un biais positif correspond à la correction d'un signal trop positif, tandis qu'un biais négatif correspond à la correction d'un signal trop négatif.

Retourne :
une valeur numérique représentant le biais du signal électrique pour la correction du point zéro

En cas d'erreur, déclenche une exception ou retourne `Y_SIGNALBIAS_INVALID`.

genericsensor→get_signalRange() YGenericSensor
genericsensor→signalRange()
genericsensor.get_signalRange()
genericsensor.get_signalRange()

Retourne la plage de signal électrique utilisé par le capteur.

function **get_signalRange**( )

Retourne :
une chaîne de caractères représentant la plage de signal électrique utilisé par le capteur

En cas d'erreur, déclenche une exception ou retourne `Y_SIGNALRANGE_INVALID`.

genericsensor→get_signalSampling() YGenericSensor
genericsensor→signalSampling()
genericsensor.get_signalSampling()
genericsensor.get_signalSampling()

Retourne la méthode d'échantillonnage du signal utilisée.

function **get_signalSampling**( )

La méthode HIGH_RATE effectue les mesures le plus rapidement possible, sans aucun filtrage. La méthode HIGH_RATE_FILTERED rajoute un filtre médian sur une fenêtre de 7 échantillons. La méthode LOW_NOISE utilise une fréquence d'aquisition réduite pour réduire le bruit. La méthode LOW_NOISE_FILTERED combine la fréquence réduite avec un filtre médian, pour obtenir des mesures aussi stables que possible même sur un signal bruité.

Retourne :

une valeur parmi Y_SIGNALSAMPLING_HIGH_RATE, Y_SIGNALSAMPLING_HIGH_RATE_FILTERED, Y_SIGNALSAMPLING_LOW_NOISE et Y_SIGNALSAMPLING_LOW_NOISE_FILTERED représentant la méthode d'échantillonnage du signal utilisée

En cas d'erreur, déclenche une exception ou retourne Y_SIGNALSAMPLING_INVALID.

genericsensor→get_signalUnit() YGenericSensor
genericsensor→signalUnit()
genericsensor.get_signalUnit()
genericsensor.get_signalUnit()

Retourne l'unité du signal électrique utilisé par le capteur.

function **get_signalUnit**( )

Retourne :
une chaîne de caractères représentant l'unité du signal électrique utilisé par le capteur

En cas d'erreur, déclenche une exception ou retourne `Y_SIGNALUNIT_INVALID`.

genericsensor→get_signalValue() YGenericSensor
genericsensor→signalValue()
genericsensor.get_signalValue()
genericsensor.get_signalValue()

Retourne la valeur actuelle du signal électrique mesuré par le capteur.

function **get_signalValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle du signal électrique mesuré par le capteur

En cas d'erreur, déclenche une exception ou retourne `Y_SIGNALVALUE_INVALID`.

genericsensor→get_unit() YGenericSensor
genericsensor→unit()genericsensor.get_unit()
genericsensor.get_unit()

Retourne l'unité dans laquelle la measure est exprimée.

function **get_unit**( )

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la measure 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()
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→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 occasionellement 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→loadAttribute() YGenericSensor
genericsensor.loadAttribute()
genericsensor.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

genericsensor→loadCalibrationPoints() YGenericSensor
genericsensor.loadCalibrationPoints()
genericsensor.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→muteValueCallbacks() YGenericSensor
genericsensor.muteValueCallbacks()
genericsensor.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→nextGenericSensor() YGenericSensor
genericsensor.nextGenericSensor()
genericsensor.nextGenericSensor()

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() YGenericSensor
genericsensor.registerTimedReportCallback()
genericsensor.registerTimedReportCallback()

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() YGenericSensor
genericsensor.registerValueCallback()
genericsensor.registerValueCallback()

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() YGenericSensor
genericsensor→setHighestValue()
genericsensor.set_highestValue()
genericsensor.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.

genericsensor→set_logFrequency() YGenericSensor
genericsensor→setLogFrequency()
genericsensor.set_logFrequency()
genericsensor.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.

genericsensor→set_logicalName() YGenericSensor
genericsensor→setLogicalName()
genericsensor.set_logicalName()
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() YGenericSensor
genericsensor→setLowestValue()
genericsensor.set_lowestValue()
genericsensor.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.

genericsensor→set_reportFrequency() YGenericSensor
genericsensor→setReportFrequency()
genericsensor.set_reportFrequency()
genericsensor.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.

genericsensor→set_resolution() YGenericSensor
genericsensor→setResolution()
genericsensor.set_resolution()
genericsensor.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.

genericsensor→set_signalBias() YGenericSensor
genericsensor→setSignalBias()
genericsensor.set_signalBias()
genericsensor.set_signalBias()

Modifie le biais du signal électrique pour la correction du point zéro.

function **set_signalBias**(**newval**)

Si votre signal électrique est positif lorsqu'il devrait être nul, configurez un biais positif de la même valeur afin de corriger l'erreur.

Paramètres :

newval une valeur numérique représentant le biais du signal électrique pour la correction du point zéro

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→set_signalRange() YGenericSensor
genericsensor→setSignalRange()
genericsensor.set_signalRange()
genericsensor.set_signalRange()

Modifie la plage de signal électrique utilisé par le capteur.

function **set_signalRange**(**newval**)

La valeur par défaut est "-999999.999...999999.999".

Paramètres :

newval une chaîne de caractères représentant la plage de signal électrique utilisé 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_signalSampling() YGenericSensor
genericsensor→setSignalSampling()
genericsensor.set_signalSampling()
genericsensor.set_signalSampling()

Modifie la méthode d'échantillonnage du signal à utiliser.

function **set_signalSampling**(**newval**)

La méthode HIGH_RATE effectue les mesures le plus rapidement possible, sans aucun filtrage. La méthode HIGH_RATE_FILTERED rajoute un filtre médian sur une fenêtre de 7 échantillons. La méthode LOW_NOISE utilise une fréquence d'aquisition réduite pour réduire le bruit. La méthode LOW_NOISE_FILTERED combine la fréquence réduite avec un filtre médian, pour obtenir des mesures aussi stables que possible même sur un signal bruité.

Paramètres :

newval une valeur parmi Y_SIGNALSAMPLING_HIGH_RATE, Y_SIGNALSAMPLING_HIGH_RATE_FILTERED, Y_SIGNALSAMPLING_LOW_NOISE et Y_SIGNALSAMPLING_LOW_NOISE_FILTERED représentant la méthode d'échantillonnage du signal à 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.

genericsensor→set_unit() YGenericSensor
genericsensor→setUnit()genericsensor.set_unit()
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() YGenericSensor
genericsensor→setUserData()
genericsensor.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

genericsensor→set_valueRange() YGenericSensor
genericsensor→setValueRange()
genericsensor.set_valueRange()
genericsensor.set_valueRange()

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. La valeur par défaut est "-999999.999...999999.999".

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→startDataLogger() YGenericSensor
genericsensor.startDataLogger()
genericsensor.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

genericsensor→stopDataLogger() YGenericSensor
genericsensor.stopDataLogger()
genericsensor.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

genericsensor→unmuteValueCallbacks() YGenericSensor
genericsensor.unmuteValueCallbacks()
genericsensor.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→wait_async() YGenericSensor
genericsensor.wait_async()

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.

genericsensor→zeroAdjust() YGenericSensor
genericsensor.zeroAdjust()
genericsensor.zeroAdjust()

Ajuste le biais du signal de sorte à ce que la valeur actuelle du signal soit interprétée comme zéro (tare).

function **zeroAdjust**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

3.28. Interface de la fonction GPS

La fonction Gps permet d'extraire les données de positionnement du module GPS. Cette classe permet d'obtenir toutes les informations nécessaires. Cependant, si vous souhaitez définir des callbacks sur des changement de position, utilisez plutôt les classes YLatitude et YLongitude.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_gps.js'></script>
cpp	#include "yocto_gps.h"
m	#import "yocto_gps.h"
pas	uses yocto_gps;
vb	yocto_gps.vb
cs	yocto_gps.cs
java	import com.yoctopuce.YoctoAPI.YGps;
uwp	import com.yoctopuce.YoctoAPI.YGps;
py	from yocto_gps import *
php	require_once('yocto_gps.php');
es	in HTML: <script src="../../lib/yocto_gps.js"></script> in node.js: require('yoctolib-es2017/yocto_gps.js');

Fonction globales

yFindGps(**func**)

Permet de retrouver un GPS d'après un identifiant donné.

yFindGpsInContext(**yctx**, **func**)

Permet de retrouver un GPS d'après un identifiant donné dans un Context YAPI.

yFirstGps()

Commence l'énumération des le GPS accessibles par la librairie.

yFirstGpsInContext(**yctx**)

Commence l'énumération des le GPS accessibles par la librairie.

Méthodes des objets YGps

gps→clearCache()

Invalide le cache.

gps→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du GPS au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

gps→get_advertisedValue()

Retourne la valeur courante du GPS (pas plus de 6 caractères).

gps→get_altitude()

Retourne l'altitude courante, Attention: la technologie GPS ne permet d'obtenir une altitude précise, des erreurs de plusieurs dizaine des mètres sont courantes.

gps→get_coordSystem()

Retourne le système de représentation utilisé pour les données de positionnement.

gps→get_dateTime()

Retourne l'heure courante au format "AAAA/MM/JJ hh:mm:ss".

gps→get_dilution()

Retourne la dilution de précision horizontale.

gps→get_direction()

Retourne la direction du déplacement en degrés par rapport au nord vrai (géographique).

gps→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du GPS.

gps→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du GPS.

gps→get_friendlyName()

Retourne un identifiant global du GPS au format `NOM_MODULE.NOM_FONCTION`.

gps→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

gps→get_functionId()

Retourne l'identifiant matériel du GPS, sans référence au module.

gps→get_groundSpeed()

Retourne la vitesse au sol actuelle en Km/h.

gps→get_hardwareId()

Retourne l'identifiant matériel unique du GPS au format `SERIAL.FUNCTIONID`.

gps→get_isFixed()

Retourne TRUE si le récepteur a trouvé suffisamment de satellites pour fonctionner.

gps→get_latitude()

Retourne la latitude courante.

gps→get_logicalName()

Retourne le nom logique du GPS.

gps→get_longitude()

Retourne la longitude courante.

gps→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

gps→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

gps→get_satCount()

Retourne le nombre de satellites visibles.

gps→get_unixTime()

Retourne l'heure courante au format Unix (nombre de secondes écoulées depuis le 1er janvier 1970).

gps→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

gps→get_utcOffset()

Retourne le nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone).

gps→isOnline()

Vérifie si le module hébergeant le GPS est joignable, sans déclencher d'erreur.

gps→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le GPS est joignable, sans déclencher d'erreur.

gps→load(**msValidity**)

Met en cache les valeurs courantes du GPS, avec une durée de validité spécifiée.

gps→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

gps→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du GPS, avec une durée de validité spécifiée.

gps→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

gps→nextGps()

Continue l'énumération des le GPS commencée à l'aide de `yFirstGps()`.

gps→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

gps→set_coordSystem(**newval**)

Change le système de représentation utilisé pour les données de positionnement.

gps→set_logicalName(**newval**)

Modifie le nom logique du GPS.

gps→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`.

gps→set_utcOffset(**newval**)

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

gps→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

gps→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.

YGps.FindGps() YGps yFindGps()YGps.FindGps()YGps.FindGps()

Permet de retrouver un GPS d'après un identifiant donné.

function **FindGps**(**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 GPS soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGps.isOnline()` pour tester si le GPS 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le GPS sans ambiguïté

Retourne :

un objet de classe `YGps` qui permet ensuite de contrôler le GPS.

YGps.FindGpsInContext() YGps yFindGpsInContext()YGps.FindGpsInContext() YGps.FindGpsInContext()

Permet de retrouver un GPS d'après un identifiant donné dans un Context YAPI.

function **FindGpsInContext**(**yctx**, **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 GPS soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGps.isOnline()` pour tester si le GPS 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le GPS sans ambiguïté

Retourne :

un objet de classe `YGps` qui permet ensuite de contrôler le GPS.

YGps.FirstGps() YGps yFirstGps()YGps.FirstGps()YGps.FirstGps()

Commence l'énumération des le GPS accessibles par la librairie.

function **FirstGps**( )

Utiliser la fonction `YGps.nextGps()` pour itérer sur les autres le GPS.

Retourne :

un pointeur sur un objet `YGps`, correspondant au premier GPS accessible en ligne, ou `null` si il n'y a pas du GPS disponibles.

YGps.FirstGpsInContext() YGps yFirstGpsInContext()YGps.FirstGpsInContext() YGps.FirstGpsInContext()

Commence l'énumération des le GPS accessibles par la librairie.

function **FirstGpsInContext**(**yctx**)

Utiliser la fonction `YGps.nextGps()` pour itérer sur les autres le GPS.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YGps`, correspondant au premier GPS accessible en ligne, ou `null` si il n'y a pas du GPS disponibles.

gps→clearCache()gps.clearCache() YGps

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du GPS. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

gps→describe()gps.describe() YGps

Retourne un court texte décrivant de manière non-ambigüe l'instance du GPS au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :

une chaîne de caractères décrivant le GPS (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

gps→get_advertisedValue() YGps
gps→advertisedValue()gps.get_advertisedValue()
gps.get_advertisedValue()

Retourne la valeur courante du GPS (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante du GPS (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

gps→get_altitude() YGps
gps→altitude()gps.get_altitude()gps.get_altitude()

Retourne l'altitude courante, Attention: la technologie GPS ne permet d'obtenir une altitude précise, des erreurs de plusieurs dizaine des mètres sont courantes.

function **get_altitude**( )

Retourne :

une valeur numérique représentant l'altitude courante, Attention: la technologie GPS ne permet d'obtenir une altitude précise, des erreurs de plusieurs dizaine des mètres sont courantes

En cas d'erreur, déclenche une exception ou retourne `Y_ALTITUDE_INVALID`.

gps→get_coordSystem() YGps
gps→coordSystem()gps.get_coordSystem()
gps.get_coordSystem()

Retourne le système de représentation utilisé pour les données de positionnement.

function **get_coordSystem**( )

Retourne :

une valeur parmi Y_COORDSYSTEM_GPS_DMS, Y_COORDSYSTEM_GPS_DM et Y_COORDSYSTEM_GPS_D représentant le système de représentation utilisé pour les données de positionnement

En cas d'erreur, déclenche une exception ou retourne Y_COORDSYSTEM_INVALID.

gps→get_dateTime() YGps
gps→dateTime()gps.get_dateTime()
gps.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`.

gps→get_dilution() YGps
gps→dilution()gps.get_dilution()gps.get_dilution()

Retourne la dilution de précision horizontale.

```
function get_dilution( )
```

Plus ce chiffre est petit, plus la précision est grande.

Retourne :
une valeur numérique représentant la dilution de précision horizontale

En cas d'erreur, déclenche une exception ou retourne `Y_DILUTION_INVALID`.

gps→get_direction() YGps
gps→direction()gps.get_direction()
gps.get_direction()

Retourne la direction du déplacement en degrés par rapport au nord vrai (géographique).

function **get_direction**( )

Retourne :

une valeur numérique représentant la direction du déplacement en degrés par rapport au nord vrai (géographique)

En cas d'erreur, déclenche une exception ou retourne `Y_DIRECTION_INVALID`.

gps→get_errorMessage() YGps
gps→errorMessage()gps.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du GPS.

```
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 GPS.

gps→get_errorType() YGps
gps→errorType()gps.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du GPS.

```
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 GPS.

gps→get_friendlyName() YGps
gps→friendlyName()gps.get_friendlyName()

Retourne un identifiant global du GPS au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du GPS si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du GPS (par exemple: MyCustomName.relay1)

Retourne :
une chaîne de caractères identifiant le GPS en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

gps→get_functionDescriptor() YGps
gps→functionDescriptor()
gps.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

gps→get_functionId() YGps
gps→functionId()gps.get_functionId()

Retourne l'identifiant matériel du GPS, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le GPS (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

gps→get_groundSpeed() YGps
gps→groundSpeed()gps.get_groundSpeed()
gps.get_groundSpeed()

Retourne la vitesse au sol actuelle en Km/h.

function **get_groundSpeed**( )

Retourne :
une valeur numérique représentant la vitesse au sol actuelle en Km/h

En cas d'erreur, déclenche une exception ou retourne `Y_GROUNDSPEED_INVALID`.

gps→get_hardwareId() YGps
gps→hardwareId()gps.get_hardwareId()

Retourne l'identifiant matériel unique du GPS au format SERIAL.FUNCTIONID.

function **get_hardwareId**( )

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du GPS (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le GPS (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

gps→get_isFixed() YGps
gps→isFixed()gps.get_isFixed()gps.get_isFixed()

Retourne TRUE si le récepteur a trouvé suffisamment de satellites pour fonctionner.

function **get_isFixed**( )

Retourne :

soit Y_ISFIXED_FALSE, soit Y_ISFIXED_TRUE, selon TRUE si le récepteur a trouvé suffisamment de satellites pour fonctionner

En cas d'erreur, déclenche une exception ou retourne Y_ISFIXED_INVALID.

gps→get_latitude() YGps
gps→latitude()gps.get_latitude()gps.get_latitude()

Retourne la latitude courante.

function **get_latitude**( )

Retourne :
une chaîne de caractères représentant la latitude courante

En cas d'erreur, déclenche une exception ou retourne `Y_LATITUDE_INVALID`.

gps→get_logicalName() YGps
gps→logicalName()gps.get_logicalName()
gps.get_logicalName()

Retourne le nom logique du GPS.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du GPS.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

gps→get_longitude() YGps
gps→longitude()gps.get_longitude()
gps.get_longitude()

Retourne la longitude courante.

function **get_longitude**( )

Retourne :
une chaîne de caractères représentant la longitude courante

En cas d'erreur, déclenche une exception ou retourne `Y_LONGITUDE_INVALID`.

gps→get_module() YGps
gps→module()gps.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

gps→get_satCount() YGps
gps→satCount()gps.get_satCount()
gps.get_satCount()

Retourne le nombre de satellites visibles.

function **get_satCount**( )

Retourne :
un entier représentant le nombre de satellites visibles

En cas d'erreur, déclenche une exception ou retourne `Y_SATCOUNT_INVALID`.

gps→get_unixTime() YGps
gps→unixTime()gps.get_unixTime()
gps.get_unixTime()

Retourne l'heure courante au format Unix (nombre de secondes écoulées depuis le 1er janvier 1970).

function **get_unixTime**( )

Retourne :
un entier représentant l'heure courante au format Unix (nombre de secondes écoulées depuis le 1er janvier 1970)

En cas d'erreur, déclenche une exception ou retourne `Y_UNIXTIME_INVALID`.

gps→get_userData() YGps
gps→userData()gps.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.

gps→get_utcOffset() YGps
gps→utcOffset()gps.get_utcOffset()
gps.get_utcOffset()

Retourne le nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone).

function **get_utcOffset**( )

Retourne :
un entier représentant le nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone)

En cas d'erreur, déclenche une exception ou retourne `Y_UTCOFFSET_INVALID`.

gps→isOnline()gps.isOnline() YGps

Vérifie si le module hébergeant le GPS est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du GPS sont valides au moment de l'appel, le module est considéré joignable. 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 GPS est joignable, `false` sinon

gps→load()gps.load() YGps

Met en cache les valeurs courantes du GPS, 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 occasionellement 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.

gps→loadAttribute()gps.loadAttribute() YGps
gps.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

gps→muteValueCallbacks() YGps
gps.muteValueCallbacks()gps.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gps→nextGps()gps.nextGps()gps.nextGps() YGps

Continue l'énumération des le GPS commencée à l'aide de `yFirstGps()`.

function **nextGps**( )

Retourne :

un pointeur sur un objet `YGps` accessible en ligne, ou `null` lorsque l'énumération est terminée.

gps→registerValueCallback() YGps
gps.registerValueCallback()
gps.registerValueCallback()

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.

gps→set_coordSystem() YGps
gps→setCoordSystem()gps.set_coordSystem()
gps.set_coordSystem()

Change le système de représentation utilisé pour les données de positionnement.

function **set_coordSystem**(**newval**)

Paramètres :

newval une valeur parmi `Y_COORDSYSTEM_GPS_DMS`, `Y_COORDSYSTEM_GPS_DM` et `Y_COORDSYSTEM_GPS_D`

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gps→set_logicalName() YGps
gps→setLogicalName()gps.set_logicalName()
gps.set_logicalName()

Modifie le nom logique du GPS.

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 GPS.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gps→set_userData() YGps
gps→setUserData()gps.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

gps→set_utcOffset() YGps
gps→setUtcOffset()gps.set_utcOffset()
gps.set_utcOffset()

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

function **set_utcOffset**(**newval**)

Le décallage est automatiquement arrondi au quart d'heure le plus proche. Si l'heure UTC est connue, l'heure courante sera automatiquement adaptée en fonction du décalage choisi.

Paramètres :

newval un entier représentant le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone)

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gps→unmuteValueCallbacks() YGps
gps.unmuteValueCallbacks()
gps.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gps→wait_async()gps.wait_async() YGps

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 GroundSpeed

La classe YGroundSpeed permet de lire la vitesse sol sur les capteurs de géolocalisation Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_groundspeed.js'></script>
cpp	#include "yocto_groundspeed.h"
m	#import "yocto_groundspeed.h"
pas	uses yocto_groundspeed;
vb	yocto_groundspeed.vb
cs	yocto_groundspeed.cs
java	import com.yoctopuce.YoctoAPI.YGroundSpeed;
uwp	import com.yoctopuce.YoctoAPI.YGroundSpeed;
py	from yocto_groundspeed import *
php	require_once('yocto_groundspeed.php');
es	in HTML: <script src="../../lib/yocto_groundspeed.js"></script> in node.js: require('yoctolib-es2017/yocto_groundspeed.js');

Fonction globales	
yFindGroundSpeed(**func**)	Permet de retrouver un capteur de vitesse/sol d'après un identifiant donné.
yFindGroundSpeedInContext(**yctx**, **func**)	Permet de retrouver un capteur de vitesse/sol d'après un identifiant donné dans un Context YAPI.
yFirstGroundSpeed()	Commence l'énumération des capteurs de vitesse/sol accessibles par la librairie.
yFirstGroundSpeedInContext(**yctx**)	Commence l'énumération des capteurs de vitesse/sol accessibles par la librairie.

Méthodes des objets YGroundSpeed	
groundspeed→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.
groundspeed→clearCache()	Invalide le cache.
groundspeed→describe()	Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de vitesse/sol au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
groundspeed→get_advertisedValue()	Retourne la valeur courante du capteur de vitesse/sol (pas plus de 6 caractères).
groundspeed→get_currentRawValue()	Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en km/h, sous forme de nombre à virgule.
groundspeed→get_currentValue()	Retourne la valeur actuelle de la vitesse/sol, en km/h, sous forme de nombre à virgule.
groundspeed→get_dataLogger()	Retourne l'objet YDataLogger du module qui héberge le senseur.
groundspeed→get_errorMessage()	

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de vitesse/sol.

groundspeed→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de vitesse/sol.

groundspeed→get_friendlyName()

Retourne un identifiant global du capteur de vitesse/sol au format `NOM_MODULE.NOM_FONCTION`.

groundspeed→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

groundspeed→get_functionId()

Retourne l'identifiant matériel du capteur de vitesse/sol, sans référence au module.

groundspeed→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de vitesse/sol au format `SERIAL.FUNCTIONID`.

groundspeed→get_highestValue()

Retourne la valeur maximale observée pour la vitesse/sol depuis le démarrage du module.

groundspeed→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.

groundspeed→get_logicalName()

Retourne le nom logique du capteur de vitesse/sol.

groundspeed→get_lowestValue()

Retourne la valeur minimale observée pour la vitesse/sol depuis le démarrage du module.

groundspeed→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

groundspeed→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

groundspeed→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.

groundspeed→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

groundspeed→get_resolution()

Retourne la résolution des valeurs mesurées.

groundspeed→get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

groundspeed→get_unit()

Retourne l'unité dans laquelle la vitesse/sol est exprimée.

groundspeed→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

groundspeed→isOnline()

Vérifie si le module hébergeant le capteur de vitesse/sol est joignable, sans déclencher d'erreur.

groundspeed→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le capteur de vitesse/sol est joignable, sans déclencher d'erreur.

groundspeed→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

groundspeed→load(msValidity)

Met en cache les valeurs courantes du capteur de vitesse/sol, avec une durée de validité spécifiée.

groundspeed→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

groundspeed→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

groundspeed→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du capteur de vitesse/sol, avec une durée de validité spécifiée.

groundspeed→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

groundspeed→nextGroundSpeed()

Continue l'énumération des capteurs de vitesse/sol commencée à l'aide de `yFirstGroundSpeed()`.

groundspeed→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

groundspeed→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

groundspeed→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

groundspeed→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

groundspeed→set_logicalName(newval)

Modifie le nom logique du capteur de vitesse/sol.

groundspeed→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

groundspeed→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

groundspeed→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

groundspeed→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`.

groundspeed→startDataLogger()

Démarre l'enregistreur de données du module.

groundspeed→stopDataLogger()

Arrête l'enregistreur de données du module.

groundspeed→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

groundspeed→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.

YGroundSpeed.FindGroundSpeed() YGroundSpeed yFindGroundSpeed() YGroundSpeed.FindGroundSpeed() YGroundSpeed.FindGroundSpeed()

Permet de retrouver un capteur de vitesse/sol d'après un identifiant donné.

```
function FindGroundSpeed( 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 vitesse/sol soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGroundSpeed.isOnline()` pour tester si le capteur de vitesse/sol 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le capteur de vitesse/sol sans ambiguïté

Retourne :

un objet de classe `YGroundSpeed` qui permet ensuite de contrôler le capteur de vitesse/sol.

YGroundSpeed.FindGroundSpeedInContext() YGroundSpeed
yFindGroundSpeedInContext()
YGroundSpeed.FindGroundSpeedInContext()
YGroundSpeed.FindGroundSpeedInContext()

Permet de retrouver un capteur de vitesse/sol d'après un identifiant donné dans un Context YAPI.

function **FindGroundSpeedInContext**(**yctx**, **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 vitesse/sol soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGroundSpeed.isOnline()` pour tester si le capteur de vitesse/sol 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de vitesse/sol sans ambiguïté

Retourne :

un objet de classe `YGroundSpeed` qui permet ensuite de contrôler le capteur de vitesse/sol.

YGroundSpeed.FirstGroundSpeed() YGroundSpeed yFirstGroundSpeed() YGroundSpeed.FirstGroundSpeed() YGroundSpeed.FirstGroundSpeed()

Commence l'énumération des capteurs de vitesse/sol accessibles par la librairie.

```
function FirstGroundSpeed( )
```

Utiliser la fonction `YGroundSpeed.nextGroundSpeed()` pour itérer sur les autres capteurs de vitesse/sol.

Retourne :

un pointeur sur un objet `YGroundSpeed`, correspondant au premier capteur vitesse/sol accessible en ligne, ou `null` si il n'y a pas de capteurs de vitesse/sol disponibles.

YGroundSpeed.FirstGroundSpeedInContext() YGroundSpeed yFirstGroundSpeedInContext() YGroundSpeed.FirstGroundSpeedInContext() YGroundSpeed.FirstGroundSpeedInContext()

Commence l'énumération des capteurs de vitesse/sol accessibles par la librairie.

function **FirstGroundSpeedInContext**(**yctx**)

Utiliser la fonction `YGroundSpeed.nextGroundSpeed()` pour itérer sur les autres capteurs de vitesse/sol.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YGroundSpeed`, correspondant au premier capteur vitesse/sol accessible en ligne, ou `null` si il n'y a pas de capteurs de vitesse/sol disponibles.

groundspeed→calibrateFromPoints() YGroundSpeed
groundspeed.calibrateFromPoints()
groundspeed.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

groundspeed→clearCache() YGroundSpeed
groundspeed.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur de vitesse/sol. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

groundspeed→describe()groundspeed.describe() YGroundSpeed

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de vitesse/sol au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le capteur de vitesse/sol (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

groundspeed→get_advertisedValue() YGroundSpeed
groundspeed→advertisedValue()
groundspeed.get_advertisedValue()
groundspeed.get_advertisedValue()

Retourne la valeur courante du capteur de vitesse/sol (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de vitesse/sol (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

groundspeed→get_currentRawValue() YGroundSpeed
groundspeed→currentRawValue()
groundspeed.get_currentRawValue()
groundspeed.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en km/h, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en km/h, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

groundspeed→get_currentValue() YGroundSpeed
groundspeed→currentValue()
groundspeed.get_currentValue()
groundspeed.get_currentValue()

Retourne la valeur actuelle de la vitesse/sol, en km/h, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :

une valeur numérique représentant la valeur actuelle de la vitesse/sol, en km/h, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

groundspeed→get_dataLogger() YGroundSpeed
groundspeed→dataLogger()
groundspeed.get_dataLogger()
groundspeed.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

groundspeed→get_errorMessage() YGroundSpeed
groundspeed→errorMessage()
groundspeed.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de vitesse/sol.

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 vitesse/sol.

groundspeed→get_errorType() YGroundSpeed
groundspeed→errorType()
groundspeed.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de vitesse/sol.

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 vitesse/sol.

groundspeed→get_friendlyName() YGroundSpeed
groundspeed→friendlyName()
groundspeed.get_friendlyName()

Retourne un identifiant global du capteur de vitesse/sol au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du capteur de vitesse/sol si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du capteur de vitesse/sol (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le capteur de vitesse/sol en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

groundspeed→get_functionDescriptor() YGroundSpeed
groundspeed→functionDescriptor()
groundspeed.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

groundspeed→get_functionId() YGroundSpeed
groundspeed→functionId()
groundspeed.get_functionId()

Retourne l'identifiant matériel du capteur de vitesse/sol, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de vitesse/sol (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

groundspeed→get_hardwareId() YGroundSpeed
groundspeed→hardwareId()
groundspeed.get_hardwareId()

Retourne l'identifiant matériel unique du capteur de vitesse/sol 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 vitesse/sol (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le capteur de vitesse/sol (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

groundspeed→get_highestValue() YGroundSpeed
groundspeed→highestValue()
groundspeed.get_highestValue()
groundspeed.get_highestValue()

Retourne la valeur maximale observée pour la vitesse/sol depuis le démarrage du module.

function **get_highestValue**( )

Retourne :

une valeur numérique représentant la valeur maximale observée pour la vitesse/sol depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

groundspeed→get_logFrequency() YGroundSpeed
groundspeed→logFrequency()
groundspeed.get_logFrequency()
groundspeed.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`.

groundspeed→get_logicalName() YGroundSpeed
groundspeed→logicalName()
groundspeed.get_logicalName()
groundspeed.get_logicalName()

Retourne le nom logique du capteur de vitesse/sol.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du capteur de vitesse/sol.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

groundspeed→get_lowestValue() YGroundSpeed
groundspeed→lowestValue()
groundspeed.get_lowestValue()
groundspeed.get_lowestValue()

Retourne la valeur minimale observée pour la vitesse/sol depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :

une valeur numérique représentant la valeur minimale observée pour la vitesse/sol depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

groundspeed→get_module() YGroundSpeed
groundspeed→module()groundspeed.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

groundspeed→get_recordedData() YGroundSpeed
groundspeed→recordedData()
groundspeed.get_recordedData()
groundspeed.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 find de l'intercalle 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.

groundspeed→get_reportFrequency() YGroundSpeed
groundspeed→reportFrequency()
groundspeed.get_reportFrequency()
groundspeed.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`.

groundspeed→get_resolution() YGroundSpeed
groundspeed→resolution()
groundspeed.get_resolution()
groundspeed.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`.

groundspeed→get_sensorState() YGroundSpeed
groundspeed→sensorState()
groundspeed.get_sensorState()
groundspeed.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :
un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

groundspeed→get_unit() YGroundSpeed
groundspeed→unit()groundspeed.get_unit()
groundspeed.get_unit()

Retourne l'unité dans laquelle la vitesse/sol est exprimée.

function **get_unit**( )

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la vitesse/sol est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

groundspeed→get_userData() YGroundSpeed
groundspeed→userData()
groundspeed.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.

groundspeed→isOnline()groundspeed.isOnline() YGroundSpeed

Vérifie si le module hébergeant le capteur de vitesse/sol est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du capteur de vitesse/sol sont valides au moment de l'appel, le module est considéré joignable. 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 vitesse/sol est joignable, `false` sinon

groundspeed→load()groundspeed.load() YGroundSpeed

Met en cache les valeurs courantes du capteur de vitesse/sol, 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 occasionellement 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.

groundspeed→loadAttribute() YGroundSpeed
groundspeed.loadAttribute()
groundspeed.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

groundspeed→loadCalibrationPoints() YGroundSpeed
groundspeed.loadCalibrationPoints()
groundspeed.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

groundspeed→muteValueCallbacks() YGroundSpeed
groundspeed.muteValueCallbacks()
groundspeed.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

groundspeed→nextGroundSpeed() YGroundSpeed
groundspeed.nextGroundSpeed()
groundspeed.nextGroundSpeed()

Continue l'énumération des capteurs de vitesse/sol commencée à l'aide de `yFirstGroundSpeed()`.

function **nextGroundSpeed**( )

Retourne :

un pointeur sur un objet `YGroundSpeed` accessible en ligne, ou `null` lorsque l'énumération est terminée.

groundspeed→registerTimedReportCallback() YGroundSpeed
groundspeed.registerTimedReportCallback()
groundspeed.registerTimedReportCallback()

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.

groundspeed→registerValueCallback() YGroundSpeed
groundspeed.registerValueCallback()
groundspeed.registerValueCallback()

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.

groundspeed→set_highestValue() YGroundSpeed
groundspeed→setHighestValue()
groundspeed.set_highestValue()
groundspeed.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.

groundspeed→set_logFrequency() YGroundSpeed
groundspeed→setLogFrequency()
groundspeed.set_logFrequency()
groundspeed.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.

groundspeed→set_logicalName() YGroundSpeed
groundspeed→setLogicalName()
groundspeed.set_logicalName()
groundspeed.set_logicalName()

Modifie le nom logique du capteur de vitesse/sol.

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 vitesse/sol.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

groundspeed→set_lowestValue() YGroundSpeed
groundspeed→setLowestValue()
groundspeed.set_lowestValue()
groundspeed.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.

groundspeed→set_reportFrequency() YGroundSpeed
groundspeed→setReportFrequency()
groundspeed.set_reportFrequency()
groundspeed.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.

groundspeed→set_resolution() YGroundSpeed
groundspeed→setResolution()
groundspeed.set_resolution()
groundspeed.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.

groundspeed→set_userData() YGroundSpeed
groundspeed→setUserData()
groundspeed.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

groundspeed→startDataLogger() YGroundSpeed
groundspeed.startDataLogger()
groundspeed.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

groundspeed→stopDataLogger() YGroundSpeed
groundspeed.stopDataLogger()
groundspeed.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

groundspeed→unmuteValueCallbacks() YGroundSpeed
groundspeed.unmuteValueCallbacks()
groundspeed.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

groundspeed→wait_async() YGroundSpeed
groundspeed.wait_async()

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

function **wait_async**(**callback**, **context**)

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout.

3.30. Interface de la fonction Gyro

La classe YSensor est la classe parente de tous les senseurs Yoctopuce. Elle permet de lire la valeur courante et l'unité de n'importe quel capteur, de lire les valeurs min/max, de configurer la fréquence d'enregistrement autonome des données et de récupérer les mesures enregistrées. Elle permet aussi d'enregistrer un callback appelé lorsque la valeur mesurée change ou à intervalle prédéfini. L'utilisation de cette classe plutôt qu'une de ces sous-classes permet de créer des application génériques, compatibles même avec les capteurs Yoctopuce futurs. Note: la classe YAnButton est le seul type d'entrée analogique qui n'hérite pas de YSensor.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_gyro.js'></script>
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;
uwp	import com.yoctopuce.YoctoAPI.YGyro;
py	from yocto_gyro import *
php	require_once('yocto_gyro.php');
es	in HTML: <script src="../../lib/yocto_gyro.js"></script> in node.js: require('yoctolib-es2017/yocto_gyro.js');

Fonction globales

yFindGyro(**func**)

Permet de retrouver un gyroscope d'après un identifiant donné.

yFindGyroInContext(**yctx**, **func**)

Permet de retrouver un gyroscope d'après un identifiant donné dans un Context YAPI.

yFirstGyro()

Commence l'énumération des gyroscopes accessibles par la librairie.

yFirstGyroInContext(**yctx**)

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→clearCache()

Invalide le cache.

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_bandwidth()

Retourne la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

gyro→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés par seconde, sous forme de nombre à virgule.

gyro→get_currentValue()

Retourne la valeur actuelle de la vitesse angulaire, en degrés par seconde, sous forme de nombre à virgule.

gyro→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

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 estimatée 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 estimatée 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 estimatée 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 estimatée 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_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

gyro→load(msValidity)

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

gyro→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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_bandwidth(newval)

Modifie la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

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→startDataLogger()

Démarre l'enregistreur de données du module.

gyro→stopDataLogger()

Arrête l'enregistreur de données du module.

gyro→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

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() YGyro yFindGyro()YGyro.FindGyro()YGyro.FindGyro()

Permet de retrouver un gyroscope d'après un identifiant donné.

```
function FindGyro( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindGyroInContext() YGyro yFindGyroInContext()YGyro.FindGyroInContext() YGyro.FindGyroInContext()

Permet de retrouver un gyroscope d'après un identifiant donné dans un Context YAPI.

function **FindGyroInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le gyroscope sans ambiguïté

Retourne :

un objet de classe `YGyro` qui permet ensuite de contrôler le gyroscope.

YGyro.FirstGyro() YGyro yFirstGyro()YGyro.FirstGyro()YGyro.FirstGyro()

Commence l'énumération des gyroscopes accessibles par la librairie.

function **FirstGyro**( )

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.

YGyro.FirstGyroInContext() YGyro yFirstGyroInContext()YGyro.FirstGyroInContext() YGyro.FirstGyroInContext()

Commence l'énumération des gyroscopes accessibles par la librairie.

function **FirstGyroInContext**(**yctx**)

Utiliser la fonction `YGyro.nextGyro()` pour itérer sur les autres gyroscopes.

Paramètres :

yctx un contexte YAPI.

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() YGyro
gyro.calibrateFromPoints()
gyro.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→clearCache()gyro.clearCache() YGyro

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du gyroscope. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le gyroscope (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

gyro→get_advertisedValue() YGyro
gyro→advertisedValue()gyro.get_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_bandwidth() YGyro
gyro→bandwidth()gyro.get_bandwidth()
gyro.get_bandwidth()

Retourne la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

function **get_bandwidth**( )

Retourne :
un entier représentant la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement)

En cas d'erreur, déclenche une exception ou retourne `Y_BANDWIDTH_INVALID`.

gyro→get_currentRawValue() YGyro
gyro→currentRawValue()gyro.get_currentRawValue()
gyro.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés par seconde, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés par seconde, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

gyro→get_currentValue() YGyro
gyro→currentValue()gyro.get_currentValue()
gyro.get_currentValue()

Retourne la valeur actuelle de la vitesse angulaire, en degrés par seconde, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle de la vitesse angulaire, en degrés par seconde, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

gyro→get_dataLogger() YGyro
gyro→dataLogger()gyro.get_dataLogger()
gyro.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

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'identifant 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() YGyro
gyro→functionDescriptor()
gyro.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

gyro→get_functionId() YGyro
gyro→functionId()gyro.get_functionId()

Retourne l'identifiant matériel du gyroscope, sans référence au module.

function **get_functionId**( )

Par example `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 example 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()
gyro.get_heading()

Retourne une estimation du cap (angle de lacet), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

function **get_heading**( )

L'axe de lacet peut être attribué à n'importe laquelle des direction physiques X, Y ou Z du module à l'aide des méthodes de la classe `YRefFrame`.

Retourne :
un nombre à virgule correspondant au cap, exprimé en degrés (entre 0 et 360).

gyro→get_highestValue() YGyro
gyro→highestValue()gyro.get_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()
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()
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()
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_pitch() YGyro
gyro→pitch()gyro.get_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 direction physiques X, Y ou Z du module à l'aide des méthodes de la classe `YRefFrame`.

Retourne :
un nombre à virgule correspondant à l'assiette, exprimée en degrés (entre -90 et +90).

gyro→get_quaternionW() YGyro
gyro→quaternionW()gyro.get_quaternionW()
gyro.get_quaternionW()

Retourne la composante `w` (composante réelle) 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_quaternionW**( )

Retourne :

un nombre à virgule correspondant à la composante `w` du quaternion.

gyro→get_quaternionX() YGyro
gyro→quaternionX()gyro.get_quaternionX()
gyro.get_quaternionX()

Retourne la composante x 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_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()
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()
gyro.get_quaternionZ()

Retourne la composante z 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_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()
gyro.get_recordedData()

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

function **get_recordedData**(**startTime**, **endTime**)

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la find de l'intercalle 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()
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() YGyro
gyro→resolution()gyro.get_resolution()
gyro.get_resolution()

Retourne la résolution des valeurs mesurées.

function **get_resolution**( )

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

gyro→get_roll() YGyro
gyro→roll()gyro.get_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_sensorState() YGyro
gyro→sensorState()gyro.get_sensorState()
gyro.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :

un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

gyro→get_unit() YGyro
gyro→unit()gyro.get_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()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()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()gyro.get_zValue()

Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.

function **get_zValue**( )

Retourne :

une valeur numérique représentant la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_ZVALUE_INVALID`.

gyro→isOnline()gyro.isOnline() YGyro

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du gyroscope sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si le gyroscope est joignable, `false` sinon

gyro→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 occasionellement 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→loadAttribute()gyro.loadAttribute() YGyro gyro.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

gyro→loadCalibrationPoints() YGyro
gyro.loadCalibrationPoints()
gyro.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→muteValueCallbacks() YGyro
gyro.muteValueCallbacks()
gyro.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→nextGyro()gyro.nextGyro()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() YGyro
gyro.registerAnglesCallback()
gyro.registerAnglesCallback()

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() YGyro
gyro.registerQuaternionCallback()
gyro.registerQuaternionCallback()

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() YGyro
gyro.registerTimedReportCallback()
gyro.registerTimedReportCallback()

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() YGyro
gyro.registerValueCallback()
gyro.registerValueCallback()

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_bandwidth() YGyro
gyro→setBandwidth()gyro.set_bandwidth()
gyro.set_bandwidth()

Modifie la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

function **set_bandwidth**(**newval**)

Lorsque la fréquence est plus basse, un moyennage est effectué.

Paramètres :

newval un entier représentant la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement)

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_highestValue() YGyro
gyro→setHighestValue()gyro.set_highestValue()
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()
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()
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()
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() YGyro
gyro→setReportFrequency()
gyro.set_reportFrequency()
gyro.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.

gyro→set_resolution() YGyro
gyro→setResolution()gyro.set_resolution()
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→startDataLogger()gyro.startDataLogger() YGyro gyro.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

gyro→stopDataLogger()gyro.stopDataLogger() YGyro gyro.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

gyro→unmuteValueCallbacks() YGyro
gyro.unmuteValueCallbacks()
gyro.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.31. 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>
cpp	#include "yocto_hubport.h"
m	#import "yocto_hubport.h"
pas	uses yocto_hubport;
vb	yocto_hubport.vb
cs	yocto_hubport.cs
java	import com.yoctopuce.YoctoAPI.YHubPort;
uwp	import com.yoctopuce.YoctoAPI.YHubPort;
py	from yocto_hubport import *
php	require_once('yocto_hubport.php');
es	in HTML: <script src="../../lib/yocto_hubport.js"></script> in node.js: require('yoctolib-es2017/yocto_hubport.js');

Fonction globales

yFindHubPort(**func**)

Permet de retrouver un port de Yocto-hub d'après un identifiant donné.

yFindHubPortInContext(**yctx**, **func**)

Permet de retrouver un port de Yocto-hub d'après un identifiant donné dans un Context YAPI.

yFirstHubPort()

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

yFirstHubPortInContext(**yctx**)

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

Méthodes des objets YHubPort

hubport→clearCache()

Invalide le cache.

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→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

hubport→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

hubport→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

hubport→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YHubPort.FindHubPort() YHubPort yFindHubPort()YHubPort.FindHubPort() YHubPort.FindHubPort()

Permet de retrouver un port de Yocto-hub d'après un identifiant donné.

```
function FindHubPort( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindHubPortInContext() YHubPort
yFindHubPortInContext()
YHubPort.FindHubPortInContext()
YHubPort.FindHubPortInContext()

Permet de retrouver un port de Yocto-hub d'après un identifiant donné dans un Context YAPI.

function **FindHubPortInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le port de Yocto-hub sans ambiguïté

Retourne :

un objet de classe `YHubPort` qui permet ensuite de contrôler le port de Yocto-hub.

YHubPort.FirstHubPort() YHubPort yFirstHubPort()YHubPort.FirstHubPort() YHubPort.FirstHubPort()

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

function **FirstHubPort**( )

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.

YHubPort.FirstHubPortInContext() YHubPort
yFirstHubPortInContext()
YHubPort.FirstHubPortInContext()
YHubPort.FirstHubPortInContext()

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

function **FirstHubPortInContext**(**yctx**)

Utiliser la fonction `YHubPort.nextHubPort()` pour itérer sur les autres port de Yocto-hub.

Paramètres :

yctx un contexte YAPI.

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→clearCache()hubport.clearCache() YHubPort

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du port de Yocto-hub. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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()
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()
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()
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'identifant 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 example `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 example 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()
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_portState() YHubPort
hubport→portState()hubport.get_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→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 occasionellement 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→loadAttribute()hubport.loadAttribute() YHubPort hubport.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

hubport→muteValueCallbacks() YHubPort
hubport.muteValueCallbacks()
hubport.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

hubport→nextHubPort()hubport.nextHubPort() YHubPort hubport.nextHubPort()

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() YHubPort
hubport.registerValueCallback()
hubport.registerValueCallback()

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()
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()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'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

hubport→unmuteValueCallbacks() YHubPort
hubport.unmuteValueCallbacks()
hubport.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.32. Interface de la fonction Humidity

La classe YHumidity permet de lire et de configurer les capteurs d'humidité Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_humidity.js'></script>
cpp	#include "yocto_humidity.h"
m	#import "yocto_humidity.h"
pas	uses yocto_humidity;
vb	yocto_humidity.vb
cs	yocto_humidity.cs
java	import com.yoctopuce.YoctoAPI.YHumidity;
uwp	import com.yoctopuce.YoctoAPI.YHumidity;
py	from yocto_humidity import *
php	require_once('yocto_humidity.php');
es	in HTML: <script src="../../lib/yocto_humidity.js"></script> in node.js: require('yoctolib-es2017/yocto_humidity.js');

Fonction globales
yFindHumidity(**func**) Permet de retrouver un capteur d'humidité d'après un identifiant donné.
yFindHumidityInContext(**yctx**, **func**) Permet de retrouver un capteur d'humidité d'après un identifiant donné dans un Context YAPI.
yFirstHumidity() Commence l'énumération des capteurs d'humidité accessibles par la librairie.
yFirstHumidityInContext(**yctx**) 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→clearCache() Invalide le cache.
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_absHum() Retourne la valeur actuelle de l'humidité absolue, en gramme par mètre cube d'air.
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), en %RH, sous forme de nombre à virgule.
humidity→get_currentValue() Retourne la valeur actuelle de l'humidité, en %RH, sous forme de nombre à virgule.
humidity→get_dataLogger() Retourne l'objet YDataLogger du module qui héberge le senseur.

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 depuis le démarrage du module.

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 depuis le démarrage du module.

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_relHum()

Retourne la valeur actuelle de l'humidité relative, en pour cent.

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_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

humidity→load(**msValidity**)

Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.

humidity→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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.

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.

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_unit(**newval**)

Change l'unité principale dans laquelle l'humidité est exprimée.

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→startDataLogger()

Démarre l'enregistreur de données du module.

humidity→stopDataLogger()

Arrête l'enregistreur de données du module.

humidity→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

humidity→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YHumidity.FindHumidity() YHumidity yFindHumidity()YHumidity.FindHumidity() YHumidity.FindHumidity()

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

```
function FindHumidity( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindHumidityInContext() YHumidity
yFindHumidityInContext()
YHumidity.FindHumidityInContext()
YHumidity.FindHumidityInContext()

Permet de retrouver un capteur d'humidité d'après un identifiant donné dans un Context YAPI.

function **FindHumidityInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur d'humidité sans ambiguïté

Retourne :

un objet de classe `YHumidity` qui permet ensuite de contrôler le capteur d'humidité.

YHumidity.FirstHumidity() YHumidity yFirstHumidity()YHumidity.FirstHumidity() YHumidity.FirstHumidity()

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

function **FirstHumidity**( )

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.

YHumidity.FirstHumidityInContext() YHumidity
yFirstHumidityInContext()
YHumidity.FirstHumidityInContext()
YHumidity.FirstHumidityInContext()

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

function **FirstHumidityInContext**(**yctx**)

Utiliser la fonction `YHumidity.nextHumidity()` pour itérer sur les autres capteurs d'humidité.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YHumidity`, correspondant au premier capteur d'humidité accessible en ligne, ou `null` si il n'y a pas de capteurs d'humidité disponibles.

humidity→calibrateFromPoints() YHumidity
humidity.calibrateFromPoints()
humidity.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→clearCache()humidity.clearCache() YHumidity

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur d'humidité. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le capteur d'humidité (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

humidity→get_absHum() YHumidity
humidity→absHum()humidity.get_absHum()
humidity.get_absHum()

Retourne la valeur actuelle de l'humidité absolue, en gramme par mètre cube d'air.

function **get_absHum**( )

Retourne :
une valeur numérique représentant la valeur actuelle de l'humidité absolue, en gramme par mètre cube d'air

En cas d'erreur, déclenche une exception ou retourne `Y_ABSHUM_INVALID`.

humidity→get_advertisedValue() YHumidity
humidity→advertisedValue()
humidity.get_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()
humidity.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en %RH, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :
une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en %RH, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

humidity→get_currentValue() YHumidity
humidity→currentValue()humidity.get_currentValue()
humidity.get_currentValue()

Retourne la valeur actuelle de l'humidité, en %RH, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle de l'humidité, en %RH, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

humidity→get_dataLogger() YHumidity
humidity→dataLogger()humidity.get_dataLogger()
humidity.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

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'identifant 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() YHumidity
humidity→functionDescriptor()
humidity.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

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 example `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 example 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()
humidity.get_highestValue()

Retourne la valeur maximale observée pour l'humidité depuis le démarrage du module.

function **get_highestValue**( )

Retourne :

une valeur numérique représentant la valeur maximale observée pour l'humidité depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

humidity→get_logFrequency() YHumidity
humidity→logFrequency()
humidity.get_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()
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()
humidity.get_lowestValue()

Retourne la valeur minimale observée pour l'humidité depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :
une valeur numérique représentant la valeur minimale observée pour l'humidité depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

humidity→get_module() YHumidity
humidity→module()humidity.get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

function **get_module**( )

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :
une instance de YModule

humidity→get_recordedData() YHumidity
humidity→recordedData()
humidity.get_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 find de l'intercalle 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_relHum() YHumidity
humidity→relHum()humidity.get_relHum()
humidity.get_relHum()

Retourne la valeur actuelle de l'humidité relative, en pour cent.

function **get_relHum**( )

Retourne :
une valeur numérique représentant la valeur actuelle de l'humidité relative, en pour cent

En cas d'erreur, déclenche une exception ou retourne `Y_RELHUM_INVALID`.

humidity→get_reportFrequency() YHumidity
humidity→reportFrequency()
humidity.get_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()
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_sensorState() YHumidity
humidity→sensorState()humidity.get_sensorState()
humidity.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :
un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

humidity→get_unit() YHumidity
humidity→unit()humidity.get_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→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 occasionellement 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→loadAttribute()humidity.loadAttribute() YHumidity humidity.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

humidity→loadCalibrationPoints() YHumidity
humidity.loadCalibrationPoints()
humidity.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→muteValueCallbacks() YHumidity
humidity.muteValueCallbacks()
humidity.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→nextHumidity()humidity.nextHumidity() YHumidity humidity.nextHumidity()

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() YHumidity
humidity.registerTimedReportCallback()
humidity.registerTimedReportCallback()

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() YHumidity
humidity.registerValueCallback()
humidity.registerValueCallback()

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()
humidity.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.

humidity→set_logFrequency() YHumidity
humidity→setLogFrequency()
humidity.set_logFrequency()
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() YHumidity
humidity→setLogicalName()
humidity.set_logicalName()
humidity.set_logicalName()

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() YHumidity
humidity→setLowestValue()
humidity.set_lowestValue()
humidity.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.

humidity→set_reportFrequency() YHumidity
humidity→setReportFrequency()
humidity.set_reportFrequency()
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()
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_unit() YHumidity
humidity→setUnit()humidity.set_unit()
humidity.set_unit()

Change l'unité principale dans laquelle l'humidité est exprimée.

```
function set_unit( newval)
```

Cette unité est une chaîne de charactère. Si la chaîne commence par une lettre 'g', la valeur principale est l'humidité absolue en g/m3. Autrement, la valeur principale sera l'humidité relative, en pour cent.

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.

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→startDataLogger() YHumidity
humidity.startDataLogger()
humidity.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

humidity→stopDataLogger() YHumidity
humidity.stopDataLogger()
humidity.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

humidity→unmuteValueCallbacks() YHumidity
humidity.unmuteValueCallbacks()
humidity.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.33. Interface de la fonction Latitude

La classe YLatitude permet de lire la latitude sur les capteurs de géolocalisation Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_latitude.js'></script>
cpp	#include "yocto_latitude.h"
m	#import "yocto_latitude.h"
pas	uses yocto_latitude;
vb	yocto_latitude.vb
cs	yocto_latitude.cs
java	import com.yoctopuce.YoctoAPI.YLatitude;
uwp	import com.yoctopuce.YoctoAPI.YLatitude;
py	from yocto_latitude import *
php	require_once('yocto_latitude.php');
es	in HTML: <script src="../../lib/yocto_latitude.js"></script> in node.js: require('yoctolib-es2017/yocto_latitude.js');

Fonction globales

yFindLatitude(**func**)

Permet de retrouver un capteur de latitude d'après un identifiant donné.

yFindLatitudeInContext(**yctx**, **func**)

Permet de retrouver un capteur de latitude d'après un identifiant donné dans un Context YAPI.

yFirstLatitude()

Commence l'énumération des capteurs de latitude accessibles par la librairie.

yFirstLatitudeInContext(**yctx**)

Commence l'énumération des capteurs de latitude accessibles par la librairie.

Méthodes des objets `YLatitude`

latitude→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.

latitude→clearCache()

Invalide le cache.

latitude→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de latitude au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

latitude→get_advertisedValue()

Retourne la valeur courante du capteur de latitude (pas plus de 6 caractères).

latitude→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en deg/1000, sous forme de nombre à virgule.

latitude→get_currentValue()

Retourne la valeur actuelle de la latitude, en deg/1000, sous forme de nombre à virgule.

latitude→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

latitude→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de latitude.

latitude→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de latitude.

latitude→get_friendlyName()

Retourne un identifiant global du capteur de latitude au format NOM_MODULE.NOM_FONCTION.

latitude→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

latitude→get_functionId()

Retourne l'identifiant matériel du capteur de latitude, sans référence au module.

latitude→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de latitude au format SERIAL.FUNCTIONID.

latitude→get_highestValue()

Retourne la valeur maximale observée pour la latitude depuis le démarrage du module.

latitude→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.

latitude→get_logicalName()

Retourne le nom logique du capteur de latitude.

latitude→get_lowestValue()

Retourne la valeur minimale observée pour la latitude depuis le démarrage du module.

latitude→get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

latitude→get_module_async(**callback**, **context**)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

latitude→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.

latitude→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

latitude→get_resolution()

Retourne la résolution des valeurs mesurées.

latitude→get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

latitude→get_unit()

Retourne l'unité dans laquelle la latitude est exprimée.

latitude→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

latitude→isOnline()

Vérifie si le module hébergeant le capteur de latitude est joignable, sans déclencher d'erreur.

latitude→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le capteur de latitude est joignable, sans déclencher d'erreur.

latitude→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

latitude→load(**msValidity**)

Met en cache les valeurs courantes du capteur de latitude, avec une durée de validité spécifiée.

latitude→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

latitude→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

latitude→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du capteur de latitude, avec une durée de validité spécifiée.

latitude→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

latitude→nextLatitude()

Continue l'énumération des capteurs de latitude commencée à l'aide de `yFirstLatitude()`.

latitude→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

latitude→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

latitude→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

latitude→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

latitude→set_logicalName(newval)

Modifie le nom logique du capteur de latitude.

latitude→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

latitude→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

latitude→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

latitude→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`.

latitude→startDataLogger()

Démarre l'enregistreur de données du module.

latitude→stopDataLogger()

Arrête l'enregistreur de données du module.

latitude→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

latitude→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.

YLatitude.FindLatitude() YLatitude yFindLatitude()YLatitude.FindLatitude() YLatitude.FindLatitude()

Permet de retrouver un capteur de latitude d'après un identifiant donné.

```
function FindLatitude( 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 latitude soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YLatitude.isOnline()` pour tester si le capteur de latitude 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le capteur de latitude sans ambiguïté

Retourne :

un objet de classe `YLatitude` qui permet ensuite de contrôler le capteur de latitude.

YLatitude.FindLatitudeInContext() YLatitude
yFindLatitudeInContext()
YLatitude.FindLatitudeInContext()
YLatitude.FindLatitudeInContext()

Permet de retrouver un capteur de latitude d'après un identifiant donné dans un Context YAPI.

function **FindLatitudeInContext**(**yctx**, **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 latitude soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YLatitude.isOnline()` pour tester si le capteur de latitude 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de latitude sans ambiguïté

Retourne :

un objet de classe `YLatitude` qui permet ensuite de contrôler le capteur de latitude.

YLatitude.FirstLatitude() YLatitude yFirstLatitude()YLatitude.FirstLatitude() YLatitude.FirstLatitude()

Commence l'énumération des capteurs de latitude accessibles par la librairie.

function **FirstLatitude**( )

Utiliser la fonction `YLatitude.nextLatitude()` pour itérer sur les autres capteurs de latitude.

Retourne :

un pointeur sur un objet `YLatitude`, correspondant au premier capteur delatitude accessible en ligne, ou `null` si il n'y a pas de capteurs de latitude disponibles.

YLatitude.FirstLatitudeInContext() YLatitude
yFirstLatitudeInContext()
YLatitude.FirstLatitudeInContext()
YLatitude.FirstLatitudeInContext()

Commence l'énumération des capteurs de latitude accessibles par la librairie.

function **FirstLatitudeInContext**(**yctx**)

Utiliser la fonction `YLatitude.nextLatitude()` pour itérer sur les autres capteurs de latitude.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YLatitude`, correspondant au premier capteur delatitude accessible en ligne, ou `null` si il n'y a pas de capteurs de latitude disponibles.

latitude→calibrateFromPoints() YLatitude
latitude.calibrateFromPoints()
latitude.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

latitude→clearCache()latitude.clearCache() YLatitude

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur de latitude. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

latitude→describe()latitude.describe() YLatitude

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de latitude au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le capteur de latitude (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

latitude→get_advertisedValue() YLatitude
latitude→advertisedValue()
latitude.get_advertisedValue()
latitude.get_advertisedValue()

Retourne la valeur courante du capteur de latitude (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante du capteur de latitude (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

latitude→get_currentRawValue() YLatitude
latitude→currentRawValue()
latitude.get_currentRawValue()
latitude.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en deg/1000, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en deg/1000, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

latitude→get_currentValue() YLatitude
latitude→currentValue()latitude.get_currentValue()
latitude.get_currentValue()

Retourne la valeur actuelle de la latitude, en deg/1000, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle de la latitude, en deg/1000, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

latitude→get_dataLogger() YLatitude
latitude→dataLogger()latitude.get_dataLogger()
latitude.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

latitude→get_errorMessage() YLatitude
latitude→errorMessage()latitude.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de latitude.

```
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 latitude.

latitude→get_errorType() YLatitude
latitude→errorType()latitude.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de latitude.

```
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 latitude.

latitude→get_friendlyName() YLatitude
latitude→friendlyName()latitude.get_friendlyName()

Retourne un identifiant global du capteur de latitude au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du capteur de latitude si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du capteur de latitude (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le capteur de latitude en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

latitude→get_functionDescriptor() YLatitude
latitude→functionDescriptor()
latitude.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

latitude→get_functionId() YLatitude
latitude→functionId()latitude.get_functionId()

Retourne l'identifiant matériel du capteur de latitude, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de latitude (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

latitude→get_hardwareId() YLatitude
latitude→hardwareId()latitude.get_hardwareId()

Retourne l'identifiant matériel unique du capteur de latitude 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 latitude (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le capteur de latitude (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

latitude→get_highestValue() YLatitude
latitude→highestValue()latitude.get_highestValue()
latitude.get_highestValue()

Retourne la valeur maximale observée pour la latitude depuis le démarrage du module.

function **get_highestValue**( )

Retourne :

une valeur numérique représentant la valeur maximale observée pour la latitude depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

latitude→get_logFrequency() YLatitude
latitude→logFrequency()latitude.get_logFrequency()
latitude.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`.

latitude→get_logicalName() YLatitude
latitude→logicalName()latitude.get_logicalName()
latitude.get_logicalName()

Retourne le nom logique du capteur de latitude.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du capteur de latitude.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

latitude→get_lowestValue() YLatitude
latitude→lowestValue()latitude.get_lowestValue()
latitude.get_lowestValue()

Retourne la valeur minimale observée pour la latitude depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :
une valeur numérique représentant la valeur minimale observée pour la latitude depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

latitude→get_module() YLatitude
latitude→module()latitude.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

latitude→get_recordedData() YLatitude
latitude→recordedData()latitude.get_recordedData()
latitude.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 find de l'intercalle 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.

latitude→get_reportFrequency() YLatitude
latitude→reportFrequency()
latitude.get_reportFrequency()
latitude.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`.

latitude→get_resolution() YLatitude
latitude→resolution()latitude.get_resolution()
latitude.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`.

latitude→get_sensorState() YLatitude
latitude→sensorState()latitude.get_sensorState()
latitude.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :

un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

latitude→get_unit() YLatitude
latitude→unit()latitude.get_unit()latitude.get_unit()

Retourne l'unité dans laquelle la latitude est exprimée.

function **get_unit**( )

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la latitude est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

latitude→get_userData() YLatitude
latitude→userData()latitude.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.

latitude→isOnline()latitude.isOnline() YLatitude

Vérifie si le module hébergeant le capteur de latitude est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du capteur de latitude sont valides au moment de l'appel, le module est considéré joignable. 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 latitude est joignable, `false` sinon

latitude→load()latitude.load() YLatitude

Met en cache les valeurs courantes du capteur de latitude, 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 occasionellement 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.

latitude→loadAttribute()latitude.loadAttribute() YLatitude latitude.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

latitude→loadCalibrationPoints() YLatitude
latitude.loadCalibrationPoints()
latitude.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

latitude→muteValueCallbacks() YLatitude
latitude.muteValueCallbacks()
latitude.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

latitude→nextLatitude()latitude.nextLatitude() YLatitude latitude.nextLatitude()

Continue l'énumération des capteurs de latitude commencée à l'aide de `yFirstLatitude()`.

function **nextLatitude**( )

Retourne :

un pointeur sur un objet `YLatitude` accessible en ligne, ou `null` lorsque l'énumération est terminée.

latitude→registerTimedReportCallback() YLatitude
latitude.registerTimedReportCallback()
latitude.registerTimedReportCallback()

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.

latitude→registerValueCallback() YLatitude
latitude.registerValueCallback()
latitude.registerValueCallback()

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.

latitude→set_highestValue() YLatitude
latitude→setHighestValue()
latitude.set_highestValue()
latitude.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.

latitude→set_logFrequency() YLatitude
latitude→setLogFrequency()
latitude.set_logFrequency()
latitude.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.

latitude→set_logicalName() YLatitude
latitude→setLogicalName()latitude.set_logicalName()
latitude.set_logicalName()

Modifie le nom logique du capteur de latitude.

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 latitude.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

latitude→set_lowestValue() YLatitude
latitude→setLowestValue()latitude.set_lowestValue()
latitude.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.

latitude→set_reportFrequency() YLatitude
latitude→setReportFrequency()
latitude.set_reportFrequency()
latitude.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.

latitude→set_resolution() YLatitude
latitude→setResolution()latitude.set_resolution()
latitude.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.

latitude→set_userData() YLatitude
latitude→setUserData()latitude.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

latitude→startDataLogger()latitude.startDataLogger() YLatitude latitude.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

latitude→stopDataLogger()latitude.stopDataLogger() YLatitude latitude.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

latitude→unmuteValueCallbacks() YLatitude
latitude.unmuteValueCallbacks()
latitude.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

latitude→wait_async()latitude.wait_async() YLatitude

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. Interface de la fonction Led

La librairie de programmation Yoctopuce permet non seulement d'allumer la LED à une intensité donnée, mais aussi de la faire osciller à plusieurs fréquences.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_led.js'></script>
cpp     #include "yocto_led.h"
m       #import "yocto_led.h"
pas     uses yocto_led;
vb      yocto_led.vb
cs      yocto_led.cs
java    import com.yoctopuce.YoctoAPI.YLed;
uwp     import com.yoctopuce.YoctoAPI.YLed;
py      from yocto_led import *
php     require_once('yocto_led.php');
es      in HTML: <script src="../../lib/yocto_led.js"></script>
        in node.js: require('yoctolib-es2017/yocto_led.js');
```

Fonction globales
yFindLed(**func**) Permet de retrouver une LED d'après un identifiant donné.
yFindLedInContext(**yctx**, **func**) Permet de retrouver une LED d'après un identifiant donné dans un Context YAPI.
yFirstLed() Commence l'énumération des LEDs accessibles par la librairie.
yFirstLedInContext(**yctx**) Commence l'énumération des LEDs accessibles par la librairie.
Méthodes des objets YLed
led→clearCache() Invalide le cache.
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→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

led→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes de la LED, avec une durée de validité spécifiée.

led→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

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() YLed yFindLed()YLed.FindLed()YLed.FindLed()

Permet de retrouver une LED d'après un identifiant donné.

```
function FindLed( 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindLedInContext() YLed yFindLedInContext()YLed.FindLedInContext() YLed.FindLedInContext()

Permet de retrouver une LED d'après un identifiant donné dans un Context YAPI.

function **FindLedInContext**(**yctx**, **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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence la LED sans ambiguïté

Retourne :

un objet de classe `YLed` qui permet ensuite de contrôler la LED.

YLed.FirstLed() YLed yFirstLed()YLed.FirstLed()YLed.FirstLed()

Commence l'énumération des LEDs accessibles par la librairie.

function **FirstLed**( )

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.

YLed.FirstLedInContext() YLed yFirstLedInContext()YLed.FirstLedInContext() YLed.FirstLedInContext()

Commence l'énumération des LEDs accessibles par la librairie.

function **FirstLedInContext**(**yctx**)

Utiliser la fonction `YLed.nextLed()` pour itérer sur les autres LEDs.

Paramètres :

yctx un contexte YAPI.

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→clearCache()led.clearCache() YLed

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de la LED. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

Retourne :

une chaîne de caractères décrivant la LED (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

led→get_advertisedValue() YLed
led→advertisedValue()led.get_advertisedValue()
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()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'identifant 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 example `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 example 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()
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()
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_power() YLed
led→power()led.get_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→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 occasionellement 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→loadAttribute()led.loadAttribute() YLed led.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

led→muteValueCallbacks()led.muteValueCallbacks() YLed led.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

led→nextLed()led.nextLed()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() YLed
led.registerValueCallback()
led.registerValueCallback()

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()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()
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()
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()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→unmuteValueCallbacks() YLed
led.unmuteValueCallbacks()
led.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.35. Interface de la fonction LightSensor

La classe YLightSensor permet de lire et de configurer les capteurs de lumière Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, elle permet d'effectuer facilement une calibration linéaire à un point pour compenser l'effet d'une vitre ou d'un filtre placé devant le capteur. Pour certains capteurs de lumière ayant plusieurs modes de fonctionnement, cette classe permet aussi de configurer le mode désiré.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_lightsensor.js'></script>
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;
uwp     import com.yoctopuce.YoctoAPI.YLightSensor;
py      from yocto_lightsensor import *
php     require_once('yocto_lightsensor.php');
es      in HTML: <script src="../../lib/yocto_lightsensor.js"></script>
        in node.js: require('yoctolib-es2017/yocto_lightsensor.js');
```

Fonction globales
yFindLightSensor(**func**) Permet de retrouver un capteur de lumière d'après un identifiant donné.
yFindLightSensorInContext(**yctx**, **func**) Permet de retrouver un capteur de lumière d'après un identifiant donné dans un Context YAPI.
yFirstLightSensor() Commence l'énumération des capteurs de lumière accessibles par la librairie.
yFirstLightSensorInContext(**yctx**) 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→clearCache() Invalide le cache.
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), en l'unité spécifiée, sous forme de nombre à virgule.
lightsensor→get_currentValue()

Retourne la valeur actuelle de la lumière ambiante, en l'unité spécifiée, sous forme de nombre à virgule.

lightsensor→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

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 depuis le démarrage du module.

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 depuis le démarrage du module.

lightsensor→get_measureType()

Retourne le type de mesure de lumière utilisé par le module.

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_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

lightsensor→load(**msValidity**)

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

lightsensor→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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.

lightsensor→set_logFrequency(**newval**)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

lightsensor→set_logicalName(**newval**)

Modifie le nom logique du capteur de lumière.

lightsensor→set_lowestValue(**newval**)

Modifie la mémoire de valeur minimale observée.

lightsensor→set_measureType(**newval**)

Change le type dde mesure de lumière effectuée par le capteur.

lightsensor→set_reportFrequency(**newval**)

Modifie la fréquence de notification périodique des valeurs mesurées.

lightsensor→set_resolution(**newval**)

Modifie la résolution des valeurs physique mesurées.

lightsensor→set_userData(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

lightsensor→startDataLogger()

Démarre l'enregistreur de données du module.

lightsensor→stopDataLogger()

Arrête l'enregistreur de données du module.

lightsensor→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

lightsensor→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YLightSensor.FindLightSensor() YLightSensor yFindLightSensor()YLightSensor.FindLightSensor() YLightSensor.FindLightSensor()

Permet de retrouver un capteur de lumière d'après un identifiant donné.

```
function FindLightSensor( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindLightSensorInContext() YLightSensor yFindLightSensorInContext() YLightSensor.FindLightSensorInContext() YLightSensor.FindLightSensorInContext()

Permet de retrouver un capteur de lumière d'après un identifiant donné dans un Context YAPI.

function **FindLightSensorInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de lumière sans ambiguïté

Retourne :

un objet de classe `YLightSensor` qui permet ensuite de contrôler le capteur de lumière.

YLightSensor.FirstLightSensor() YLightSensor yFirstLightSensor()YLightSensor.FirstLightSensor() YLightSensor.FirstLightSensor()

Commence l'énumération des capteurs de lumière accessibles par la librairie.

function **FirstLightSensor**( )

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.

YLightSensor.FirstLightSensorInContext() YLightSensor yFirstLightSensorInContext()
YLightSensor.FirstLightSensorInContext()
YLightSensor.FirstLightSensorInContext()

Commence l'énumération des capteurs de lumière accessibles par la librairie.

```
function FirstLightSensorInContext( yctx)
```

Utiliser la fonction `YLightSensor.nextLightSensor()` pour itérer sur les autres capteurs de lumière.

Paramètres :

yctx un contexte YAPI.

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 lightsensor.calibrate()

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() YLightSensor
lightsensor.calibrateFromPoints()
lightsensor.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→clearCache()lightsensor.clearCache() YLightSensor

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur de lumière. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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()
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()
lightsensor.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

lightsensor→get_currentValue() YLightSensor
lightsensor→currentValue()
lightsensor.get_currentValue()
lightsensor.get_currentValue()

Retourne la valeur actuelle de la lumière ambiante, en l'unité spécifiée, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle de la lumière ambiante, en l'unité spécifiée, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

lightsensor→get_dataLogger() YLightSensor
lightsensor→dataLogger()
lightsensor.get_dataLogger()
lightsensor.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

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'identifant matériel du capteur de lumière (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le capteur de lumière en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

lightsensor→get_functionDescriptor() YLightSensor
lightsensor→functionDescriptor()
lightsensor.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

function **get_functionDescriptor**( )

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur 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 example `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 example 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()
lightsensor.get_highestValue()

Retourne la valeur maximale observée pour la lumière ambiante depuis le démarrage du module.

function **get_highestValue**( )

Retourne :

une valeur numérique représentant la valeur maximale observée pour la lumière ambiante depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

lightsensor→get_logFrequency() YLightSensor
lightsensor→logFrequency()
lightsensor.get_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()
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()
lightsensor.get_lowestValue()

Retourne la valeur minimale observée pour la lumière ambiante depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :

une valeur numérique représentant la valeur minimale observée pour la lumière ambiante depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

lightsensor→get_measureType() YLightSensor
lightsensor→measureType()
lightsensor.get_measureType()
lightsensor.get_measureType()

Retourne le type de mesure de lumière utilisé par le module.

function **get_measureType**( )

Retourne :

une valeur parmi `Y_MEASURETYPE_HUMAN_EYE`, `Y_MEASURETYPE_WIDE_SPECTRUM`, `Y_MEASURETYPE_INFRARED`, `Y_MEASURETYPE_HIGH_RATE` et `Y_MEASURETYPE_HIGH_ENERGY` représentant le type de mesure de lumière utilisé par le module

En cas d'erreur, déclenche une exception ou retourne `Y_MEASURETYPE_INVALID`.

lightsensor→get_module() YLightSensor
lightsensor→module()lightsensor.get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

function **get_module**( )

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :
une instance de `YModule`

lightsensor→get_recordedData() YLightSensor
lightsensor→recordedData()
lightsensor.get_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**)

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 find de l'intercalle 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()
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()
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_sensorState() YLightSensor
lightsensor→sensorState()
lightsensor.get_sensorState()
lightsensor.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :

un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

lightsensor→get_unit() YLightSensor
lightsensor→unit()lightsensor.get_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→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 occasionellement 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→loadAttribute() YLightSensor lightsensor.loadAttribute()lightsensor.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

lightsensor→loadCalibrationPoints() YLightSensor
lightsensor.loadCalibrationPoints()
lightsensor.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→muteValueCallbacks() YLightSensor
lightsensor.muteValueCallbacks()
lightsensor.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→nextLightSensor() YLightSensor
lightsensor.nextLightSensor()
lightsensor.nextLightSensor()

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() YLightSensor
lightsensor.registerTimedReportCallback()
lightsensor.registerTimedReportCallback()

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

function **registerTimedReportCallback**(**callback**)

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

lightsensor→registerValueCallback() YLightSensor
lightsensor.registerValueCallback()
lightsensor.registerValueCallback()

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()
lightsensor.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.

lightsensor→set_logFrequency() YLightSensor
lightsensor→setLogFrequency()
lightsensor.set_logFrequency()
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()
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()
lightsensor.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.

lightsensor→set_measureType() YLightSensor
lightsensor→setMeasureType()
lightsensor.set_measureType()
lightsensor.set_measureType()

Change le type dde mesure de lumière effectuée par le capteur.

function **set_measureType**(**newval**)

La mesure peut soit approximer la réponse de l'oeil humain, soit donner une valeur ciblant un spectre particulier, en fonction des possibilités offertes par le récepteur de lumière. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur parmi `Y_MEASURETYPE_HUMAN_EYE`, `Y_MEASURETYPE_WIDE_SPECTRUM`, `Y_MEASURETYPE_INFRARED`, `Y_MEASURETYPE_HIGH_RATE` et `Y_MEASURETYPE_HIGH_ENERGY`

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_reportFrequency() YLightSensor
lightsensor→setReportFrequency()
lightsensor.set_reportFrequency()
lightsensor.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.

lightsensor→set_resolution() YLightSensor
lightsensor→setResolution()
lightsensor.set_resolution()
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→startDataLogger() YLightSensor
lightsensor.startDataLogger()
lightsensor.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

lightsensor→stopDataLogger() YLightSensor
lightsensor.stopDataLogger()
lightsensor.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

lightsensor→unmuteValueCallbacks() YLightSensor
lightsensor.unmuteValueCallbacks()
lightsensor.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.36. Interface de la fonction Longitude

La classe YLongitude permet de lire la longitude sur les capteurs de géolocalisation Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js    <script type='text/javascript' src='yocto_longitude.js'></script>
cpp   #include "yocto_longitude.h"
m     #import "yocto_longitude.h"
pas   uses yocto_longitude;
vb    yocto_longitude.vb
cs    yocto_longitude.cs
java  import com.yoctopuce.YoctoAPI.YLongitude;
uwp   import com.yoctopuce.YoctoAPI.YLongitude;
py    from yocto_longitude import *
php   require_once('yocto_longitude.php');
es    in HTML: <script src="../../lib/yocto_longitude.js"></script>
      in node.js: require('yoctolib-es2017/yocto_longitude.js');
```

Fonction globales
yFindLongitude(**func**) Permet de retrouver un capteur de longitude d'après un identifiant donné.
yFindLongitudeInContext(**yctx**, **func**) Permet de retrouver un capteur de longitude d'après un identifiant donné dans un Context YAPI.
yFirstLongitude() Commence l'énumération des capteurs de longitude accessibles par la librairie.
yFirstLongitudeInContext(**yctx**) Commence l'énumération des capteurs de longitude accessibles par la librairie.
Méthodes des objets `YLongitude`
longitude→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.
longitude→clearCache() Invalide le cache.
longitude→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de longitude au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
longitude→get_advertisedValue() Retourne la valeur courante du capteur de longitude (pas plus de 6 caractères).
longitude→get_currentRawValue() Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en deg/1000, sous forme de nombre à virgule.
longitude→get_currentValue() Retourne la valeur actuelle de la longitude, en deg/1000, sous forme de nombre à virgule.
longitude→get_dataLogger() Retourne l'objet YDataLogger du module qui héberge le senseur.
longitude→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de longitude.

longitude→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de longitude.

longitude→get_friendlyName()

Retourne un identifiant global du capteur de longitude au format `NOM_MODULE.NOM_FONCTION`.

longitude→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

longitude→get_functionId()

Retourne l'identifiant matériel du capteur de longitude, sans référence au module.

longitude→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de longitude au format `SERIAL.FUNCTIONID`.

longitude→get_highestValue()

Retourne la valeur maximale observée pour la longitude depuis le démarrage du module.

longitude→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.

longitude→get_logicalName()

Retourne le nom logique du capteur de longitude.

longitude→get_lowestValue()

Retourne la valeur minimale observée pour la longitude depuis le démarrage du module.

longitude→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

longitude→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

longitude→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.

longitude→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

longitude→get_resolution()

Retourne la résolution des valeurs mesurées.

longitude→get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

longitude→get_unit()

Retourne l'unité dans laquelle la longitude est exprimée.

longitude→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

longitude→isOnline()

Vérifie si le module hébergeant le capteur de longitude est joignable, sans déclencher d'erreur.

longitude→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le capteur de longitude est joignable, sans déclencher d'erreur.

longitude→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

longitude→load(**msValidity**)

Met en cache les valeurs courantes du capteur de longitude, avec une durée de validité spécifiée.

longitude→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

longitude→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

longitude→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du capteur de longitude, avec une durée de validité spécifiée.

longitude→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

longitude→nextLongitude()

Continue l'énumération des capteurs de longitude commencée à l'aide de `yFirstLongitude()`.

longitude→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

longitude→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

longitude→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

longitude→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

longitude→set_logicalName(newval)

Modifie le nom logique du capteur de longitude.

longitude→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

longitude→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

longitude→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

longitude→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`.

longitude→startDataLogger()

Démarre l'enregistreur de données du module.

longitude→stopDataLogger()

Arrête l'enregistreur de données du module.

longitude→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

longitude→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.

YLongitude.FindLongitude() YLongitude yFindLongitude()YLongitude.FindLongitude() YLongitude.FindLongitude()

Permet de retrouver un capteur de longitude d'après un identifiant donné.

```
function FindLongitude( 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 longitude soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YLongitude.isOnline()` pour tester si le capteur de longitude 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le capteur de longitude sans ambiguïté

Retourne :

un objet de classe `YLongitude` qui permet ensuite de contrôler le capteur de longitude.

YLongitude.FindLongitudeInContext() YLongitude
yFindLongitudeInContext()
YLongitude.FindLongitudeInContext()
YLongitude.FindLongitudeInContext()

Permet de retrouver un capteur de longitude d'après un identifiant donné dans un Context YAPI.

function **FindLongitudeInContext**(**yctx**, **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 longitude soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YLongitude.isOnline()` pour tester si le capteur de longitude 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de longitude sans ambiguïté

Retourne :

un objet de classe `YLongitude` qui permet ensuite de contrôler le capteur de longitude.

YLongitude.FirstLongitude() YLongitude yFirstLongitude()YLongitude.FirstLongitude() YLongitude.FirstLongitude()

Commence l'énumération des capteurs de longitude accessibles par la librairie.

function **FirstLongitude**( )

Utiliser la fonction `YLongitude.nextLongitude()` pour itérer sur les autres capteurs de longitude.

Retourne :

un pointeur sur un objet `YLongitude`, correspondant au premier capteur delongitude accessible en ligne, ou `null` si il n'y a pas de capteurs de longitude disponibles.

YLongitude.FirstLongitudeInContext() YLongitude
yFirstLongitudeInContext()
YLongitude.FirstLongitudeInContext()
YLongitude.FirstLongitudeInContext()

Commence l'énumération des capteurs de longitude accessibles par la librairie.

function **FirstLongitudeInContext**(**yctx**)

Utiliser la fonction `YLongitude.nextLongitude()` pour itérer sur les autres capteurs de longitude.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YLongitude`, correspondant au premier capteur delongitude accessible en ligne, ou `null` si il n'y a pas de capteurs de longitude disponibles.

longitude→calibrateFromPoints() YLongitude
longitude.calibrateFromPoints()
longitude.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

longitude→clearCache()longitude.clearCache() YLongitude

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur de longitude. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

longitude→describe()longitude.describe() YLongitude

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de longitude au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le capteur de longitude (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

longitude→get_advertisedValue() YLongitude
longitude→advertisedValue()
longitude.get_advertisedValue()
longitude.get_advertisedValue()

Retourne la valeur courante du capteur de longitude (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de longitude (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

longitude→get_currentRawValue() YLongitude
longitude→currentRawValue()
longitude.get_currentRawValue()
longitude.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en deg/1000, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en deg/1000, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

longitude→get_currentValue() YLongitude
longitude→currentValue()
longitude.get_currentValue()
longitude.get_currentValue()

Retourne la valeur actuelle de la longitude, en deg/1000, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :

une valeur numérique représentant la valeur actuelle de la longitude, en deg/1000, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

longitude→get_dataLogger() YLongitude
longitude→dataLogger()longitude.get_dataLogger()
longitude.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

longitude→get_errorMessage() YLongitude
longitude→errorMessage()
longitude.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de longitude.

```
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 longitude.

longitude→get_errorType() YLongitude
longitude→errorType()longitude.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de longitude.

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 longitude.

longitude→get_friendlyName() YLongitude
longitude→friendlyName()
longitude.get_friendlyName()

Retourne un identifiant global du capteur de longitude au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du capteur de longitude si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du capteur de longitude (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le capteur de longitude en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

longitude→get_functionDescriptor() YLongitude
longitude→functionDescriptor()
longitude.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

longitude→get_functionId() YLongitude
longitude→functionId()longitude.get_functionId()

Retourne l'identifiant matériel du capteur de longitude, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de longitude (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

longitude→get_hardwareId() YLongitude
longitude→hardwareId()longitude.get_hardwareId()

Retourne l'identifiant matériel unique du capteur de longitude 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 longitude (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le capteur de longitude (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

longitude→get_highestValue() YLongitude
longitude→highestValue()
longitude.get_highestValue()
longitude.get_highestValue()

Retourne la valeur maximale observée pour la longitude depuis le démarrage du module.

function **get_highestValue**( )

Retourne :
une valeur numérique représentant la valeur maximale observée pour la longitude depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

longitude→get_logFrequency() YLongitude
longitude→logFrequency()
longitude.get_logFrequency()
longitude.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`.

longitude→get_logicalName() YLongitude
longitude→logicalName()
longitude.get_logicalName()
longitude.get_logicalName()

Retourne le nom logique du capteur de longitude.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du capteur de longitude.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

longitude→get_lowestValue() YLongitude
longitude→lowestValue()longitude.get_lowestValue()
longitude.get_lowestValue()

Retourne la valeur minimale observée pour la longitude depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :
une valeur numérique représentant la valeur minimale observée pour la longitude depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

longitude→get_module() YLongitude
longitude→module()longitude.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

longitude→get_recordedData() YLongitude
longitude→recordedData()
longitude.get_recordedData()
longitude.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 find de l'intercalle 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.

longitude→get_reportFrequency() YLongitude
longitude→reportFrequency()
longitude.get_reportFrequency()
longitude.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`.

longitude→get_resolution() YLongitude
longitude→resolution()longitude.get_resolution()
longitude.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`.

longitude→get_sensorState() YLongitude
longitude→sensorState()longitude.get_sensorState()
longitude.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :
un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

longitude→get_unit() YLongitude
longitude→unit()longitude.get_unit()
longitude.get_unit()

Retourne l'unité dans laquelle la longitude est exprimée.

function **get_unit**( )

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la longitude est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

longitude→get_userData() YLongitude
longitude→userData()longitude.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.

longitude→isOnline()longitude.isOnline() YLongitude

Vérifie si le module hébergeant le capteur de longitude est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du capteur de longitude sont valides au moment de l'appel, le module est considéré joignable. 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 longitude est joignable, `false` sinon

longitude→load()longitude.load() YLongitude

Met en cache les valeurs courantes du capteur de longitude, 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 occasionellement 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.

longitude→loadAttribute()longitude.loadAttribute() YLongitude longitude.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

longitude→loadCalibrationPoints() YLongitude
longitude.loadCalibrationPoints()
longitude.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

longitude→muteValueCallbacks() YLongitude
longitude.muteValueCallbacks()
longitude.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

longitude→nextLongitude()longitude.nextLongitude() YLongitude longitude.nextLongitude()

Continue l'énumération des capteurs de longitude commencée à l'aide de `yFirstLongitude()`.

function **nextLongitude**( )

Retourne :

un pointeur sur un objet `YLongitude` accessible en ligne, ou `null` lorsque l'énumération est terminée.

longitude→registerTimedReportCallback() YLongitude
longitude.registerTimedReportCallback()
longitude.registerTimedReportCallback()

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.

longitude→registerValueCallback() YLongitude
longitude.registerValueCallback()
longitude.registerValueCallback()

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.

longitude→set_highestValue() YLongitude
longitude→setHighestValue()
longitude.set_highestValue()
longitude.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.

longitude→set_logFrequency() YLongitude
longitude→setLogFrequency()
longitude.set_logFrequency()
longitude.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.

longitude→set_logicalName() YLongitude
longitude→setLogicalName()
longitude.set_logicalName()
longitude.set_logicalName()

Modifie le nom logique du capteur de longitude.

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 longitude.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

longitude→set_lowestValue() YLongitude
longitude→setLowestValue()
longitude.set_lowestValue()
longitude.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.

longitude→set_reportFrequency() YLongitude
longitude→setReportFrequency()
longitude.set_reportFrequency()
longitude.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.

longitude→set_resolution() YLongitude
longitude→setResolution()longitude.set_resolution()
longitude.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.

longitude→set_userData() YLongitude
longitude→setUserData()longitude.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

longitude→startDataLogger() YLongitude
longitude.startDataLogger()
longitude.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

longitude→stopDataLogger() YLongitude
longitude.stopDataLogger()
longitude.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

longitude→unmuteValueCallbacks() YLongitude
longitude.unmuteValueCallbacks()
longitude.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

longitude→wait_async()longitude.wait_async() YLongitude

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 Magnetometer

La classe YSensor est la classe parente de tous les senseurs Yoctopuce. Elle permet de lire la valeur courante et l'unité de n'importe quel capteur, de lire les valeurs min/max, de configurer la fréquence d'enregistrement autonome des données et de récupérer les mesures enregistrées. Elle permet aussi d'enregistrer un callback appelé lorsque la valeur mesurée change ou à intervalle prédéfini. L'utilisation de cette classe plutôt qu'une de ces sous-classes permet de créer des application génériques, compatibles même avec les capteurs Yoctopuce futurs. Note: la classe YAnButton est le seul type d'entrée analogique qui n'hérite pas de YSensor.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_magnetometer.js'></script>
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;
uwp	import com.yoctopuce.YoctoAPI.YMagnetometer;
py	from yocto_magnetometer import *
php	require_once('yocto_magnetometer.php');
es	in HTML: <script src="../../lib/yocto_magnetometer.js"></script> in node.js: require('yoctolib-es2017/yocto_magnetometer.js');

Fonction globales

yFindMagnetometer(**func**)

Permet de retrouver un magnétomètre d'après un identifiant donné.

yFindMagnetometerInContext(**yctx**, **func**)

Permet de retrouver un magnétomètre d'après un identifiant donné dans un Context YAPI.

yFirstMagnetometer()

Commence l'énumération des magnétomètres accessibles par la librairie.

yFirstMagnetometerInContext(**yctx**)

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→clearCache()

Invalide le cache.

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_bandwidth()

Retourne la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

magnetometer→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en mT, sous forme de nombre à virgule.

magnetometer→get_currentValue()

Retourne la valeur actuelle du champ magnétique, en mT, sous forme de nombre à virgule.

magnetometer→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

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_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

magnetometer→load(**msValidity**)

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

magnetometer→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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_bandwidth(**newval**)

Modifie la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

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→startDataLogger()

Démarre l'enregistreur de données du module.

magnetometer→stopDataLogger()

Arrête l'enregistreur de données du module.

magnetometer→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

magnetometer→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YMagnetometer.FindMagnetometer() YMagnetometer yFindMagnetometer() YMagnetometer.FindMagnetometer() YMagnetometer.FindMagnetometer()

Permet de retrouver un magnétomètre d'après un identifiant donné.

```
function FindMagnetometer( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindMagnetometerInContext() YMagnetometer yFindMagnetometerInContext() YMagnetometer.FindMagnetometerInContext() YMagnetometer.FindMagnetometerInContext()

Permet de retrouver un magnétomètre d'après un identifiant donné dans un Context YAPI.

function **FindMagnetometerInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le magnétomètre sans ambiguïté

Retourne :

un objet de classe `YMagnetometer` qui permet ensuite de contrôler le magnétomètre.

YMagnetometer.FirstMagnetometer() YMagnetometer yFirstMagnetometer() YMagnetometer.FirstMagnetometer() YMagnetometer.FirstMagnetometer()

Commence l'énumération des magnétomètres accessibles par la librairie.

```
function FirstMagnetometer( )
```

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.

YMagnetometer.FirstMagnetometerInContext() YMagnetometer yFirstMagnetometerInContext() YMagnetometer.FirstMagnetometerInContext() YMagnetometer.FirstMagnetometerInContext()

Commence l'énumération des magnétomètres accessibles par la librairie.

function **FirstMagnetometerInContext**(**yctx**)

Utiliser la fonction `YMagnetometer.nextMagnetometer()` pour itérer sur les autres magnétomètres.

Paramètres :

yctx un contexte YAPI.

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() YMagnetometer
magnetometer.calibrateFromPoints()
magnetometer.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→clearCache() YMagnetometer
magnetometer.clearCache()

Invalide le cache.

```
function clearCache( )
```

Invalide le cache des valeurs courantes du magnétomètre. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le magnétomètre (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

magnetometer→get_advertisedValue() YMagnetometer
magnetometer→advertisedValue()
magnetometer.get_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_bandwidth() YMagnetometer
magnetometer→bandwidth()
magnetometer.get_bandwidth()
magnetometer.get_bandwidth()

Retourne la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

function **get_bandwidth**( )

Retourne :
un entier représentant la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement)

En cas d'erreur, déclenche une exception ou retourne `Y_BANDWIDTH_INVALID`.

magnetometer→get_currentRawValue() YMagnetometer
magnetometer→currentRawValue()
magnetometer.get_currentRawValue()
magnetometer.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en mT, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en mT, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

magnetometer→get_currentValue() YMagnetometer
magnetometer→currentValue()
magnetometer.get_currentValue()
magnetometer.get_currentValue()

Retourne la valeur actuelle du champ magnétique, en mT, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :

une valeur numérique représentant la valeur actuelle du champ magnétique, en mT, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

magnetometer→get_dataLogger() YMagnetometer
magnetometer→dataLogger()
magnetometer.get_dataLogger()
magnetometer.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

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'identifant 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 example `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 example 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()
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()
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()
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()
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() YMagnetometer
magnetometer→module()
magnetometer.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

magnetometer→get_recordedData() YMagnetometer
magnetometer→recordedData()
magnetometer.get_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 find de l'intercalle 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()
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()
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_sensorState() YMagnetometer
magnetometer→sensorState()
magnetometer.get_sensorState()
magnetometer.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :
un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

magnetometer→get_unit() YMagnetometer
magnetometer→unit()magnetometer.get_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.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.

magnetometer→get_xValue() YMagnetometer
magnetometer→xValue()magnetometer.get_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()
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()
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→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 occasionellement 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→loadAttribute() YMagnetometer
magnetometer.loadAttribute()
magnetometer.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

magnetometer→loadCalibrationPoints() YMagnetometer
magnetometer.loadCalibrationPoints()
magnetometer.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→muteValueCallbacks() YMagnetometer
magnetometer.muteValueCallbacks()
magnetometer.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→nextMagnetometer() YMagnetometer
magnetometer.nextMagnetometer()
magnetometer.nextMagnetometer()

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() YMagnetometer
magnetometer.registerTimedReportCallback()
magnetometer.registerTimedReportCallback()

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

function **registerTimedReportCallback**(**callback**)

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

magnetometer→registerValueCallback() YMagnetometer
magnetometer.registerValueCallback()
magnetometer.registerValueCallback()

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_bandwidth() YMagnetometer
magnetometer→setBandwidth()
magnetometer.set_bandwidth()
magnetometer.set_bandwidth()

Modifie la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

function **set_bandwidth**(**newval**)

Lorsque la fréquence est plus basse, un moyennage est effectué.

Paramètres :

newval un entier représentant la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement)

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_highestValue() YMagnetometer
magnetometer→setHighestValue()
magnetometer.set_highestValue()
magnetometer.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.

magnetometer→set_logFrequency() YMagnetometer
magnetometer→setLogFrequency()
magnetometer.set_logFrequency()
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()
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()
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() YMagnetometer
magnetometer→setReportFrequency()
magnetometer.set_reportFrequency()
magnetometer.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.

magnetometer→set_resolution() YMagnetometer
magnetometer→setResolution()
magnetometer.set_resolution()
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() YMagnetometer
magnetometer→setUserData()
magnetometer.set_userData()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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→startDataLogger() YMagnetometer
magnetometer.startDataLogger()
magnetometer.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

magnetometer→stopDataLogger() YMagnetometer
magnetometer.stopDataLogger()
magnetometer.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

magnetometer→unmuteValueCallbacks() YMagnetometer
magnetometer.unmuteValueCallbacks()
magnetometer.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→wait_async() YMagnetometer
magnetometer.wait_async()

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. Valeur mesurée

Les objets YMeasure sont utilisés dans l'interface de programmation Yoctopuce pour représenter une valeur observée un moment donnée. Ces objets sont utilisés en particulier en conjonction avec la classe YDataSet.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_api.js'></script>
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;
uwp	import com.yoctopuce.YoctoAPI.YModule;
py	from yocto_api import *
php	require_once('yocto_api.php');
es	in HTML: <script src="../../lib/yocto_api.js"></script> in node.js: require('yoctolib-es2017/yocto_api.js');

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()
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()
measure.get_endTimeUTC()

Retourne l'heure absolue de la fin de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

function **get_endTimeUTC**( )

Lors que l'enregistrement de données se fait à une fréquence supérieure à une mesure par seconde, le timestamp peuvent inclurent une fraction décimale.

Retourne :
un nombre réel positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 UTC et la fin de la mesure.

measure→get_maxValue() YMeasure
measure→maxValue()measure.get_maxValue()
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()
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()
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 inclurent une fraction décimale.

Retourne :

un nombre réel positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 UTC et la début de la mesure.

3.39. Interface de la fonction MessageBox

La fonction YMessageBox permet de recevoir et d'envoyer des messages SMS à l'aide des modules Yoctopuce dotés de connectivité cellulaire.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_messagebox.js'></script>
cpp	#include "yocto_messagebox.h"
m	#import "yocto_messagebox.h"
pas	uses yocto_messagebox;
vb	yocto_messagebox.vb
cs	yocto_messagebox.cs
java	import com.yoctopuce.YoctoAPI.YMessageBox;
uwp	import com.yoctopuce.YoctoAPI.YMessageBox;
py	from yocto_messagebox import *
php	require_once('yocto_messagebox.php');
es	in HTML: <script src="../../lib/yocto_messagebox.js"></script> in node.js: require('yoctolib-es2017/yocto_messagebox.js');

Fonction globales

yFindMessageBox(**func**)

Permet de retrouver une interface de messagerie d'après un identifiant donné.

yFindMessageBoxInContext(**yctx**, **func**)

Permet de retrouver une interface de messagerie d'après un identifiant donné dans un Context YAPI.

yFirstMessageBox()

Commence l'énumération des interfaces de messagerie accessibles par la librairie.

yFirstMessageBoxInContext(**yctx**)

Commence l'énumération des interfaces de messagerie accessibles par la librairie.

Méthodes des objets YMessageBox

messagebox→clearCache()

Invalide le cache.

messagebox→clearPduCounters()

Réinitialise les compteurs d'unités SMS transmises et reçues.

messagebox→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface de messagerie au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

messagebox→get_advertisedValue()

Retourne la valeur courante de l'interface de messagerie (pas plus de 6 caractères).

messagebox→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface de messagerie.

messagebox→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface de messagerie.

messagebox→get_friendlyName()

Retourne un identifiant global de l'interface de messagerie au format `NOM_MODULE.NOM_FONCTION`.

messagebox→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

messagebox→get_functionId()

Retourne l'identifiant matériel de l'interface de messagerie, sans référence au module.

messagebox→get_hardwareId()

Retourne l'identifiant matériel unique de l'interface de messagerie au format `SERIAL.FUNCTIONID`.

messagebox→get_logicalName()

Retourne le nom logique de l'interface de messagerie.

messagebox→get_messages()

Retourne la liste des messages reçus et non effacés.

messagebox→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

messagebox→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

messagebox→get_pduReceived()

Retourne le nombre d'unités SMS reçues jusqu'à présent.

messagebox→get_pduSent()

Retourne le nombre d'unités SMS envoyées jusqu'à présent.

messagebox→get_slotsCount()

Retourne le nombre total de positions de stockage dans la carte SIM.

messagebox→get_slotsInUse()

Retourne le nombre de positions de stockage utilisées.

messagebox→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

messagebox→isOnline()

Vérifie si le module hébergeant l'interface de messagerie est joignable, sans déclencher d'erreur.

messagebox→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant l'interface de messagerie est joignable, sans déclencher d'erreur.

messagebox→load(**msValidity**)

Met en cache les valeurs courantes de l'interface de messagerie, avec une durée de validité spécifiée.

messagebox→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

messagebox→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes de l'interface de messagerie, avec une durée de validité spécifiée.

messagebox→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

messagebox→newMessage(**recipient**)

Crée un nouveau message SMS vide, qui pourra ensuite être librement paramétré puis envoyé.

messagebox→nextMessageBox()

Continue l'énumération des interfaces de messagerie commencée à l'aide de `yFirstMessageBox()`.

messagebox→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

messagebox→sendFlashMessage(**recipient**, **message**)

Envoie un SMS "Flash", aussi appelé message de classe 0.

messagebox→sendTextMessage(**recipient**, **message**)

Envoie un SMS textuel, avec les paramètres standards.

messagebox→set_logicalName(**newval**)

Modifie le nom logique de l'interface de messagerie.

messagebox→set_pduReceived(newval)

Modifie la valeur du compteur d'unités SMS reçues.

messagebox→set_pduSent(newval)

Modifie la valeur du compteur d'unités SMS envoyées.

messagebox→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`.

messagebox→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

messagebox→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.

YMessageBox.FindMessageBox() YMessageBox yFindMessageBox() YMessageBox.FindMessageBox() YMessageBox.FindMessageBox()

Permet de retrouver une interface de messagerie d'après un identifiant donné.

```
function FindMessageBox( 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 de messagerie soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YMessageBox.isOnline()` pour tester si l'interface de messagerie 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence l'interface de messagerie sans ambiguïté

Retourne :

un objet de classe `YMessageBox` qui permet ensuite de contrôler l'interface de messagerie.

YMessageBox.FindMessageBoxInContext() YMessageBox yFindMessageBoxInContext() YMessageBox.FindMessageBoxInContext() YMessageBox.FindMessageBoxInContext()

Permet de retrouver une interface de messagerie d'après un identifiant donné dans un Context YAPI.

function **FindMessageBoxInContext**(**yctx**, **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 de messagerie soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YMessageBox.isOnline()` pour tester si l'interface de messagerie 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'interface de messagerie sans ambiguïté

Retourne :

un objet de classe `YMessageBox` qui permet ensuite de contrôler l'interface de messagerie.

YMessageBox.FirstMessageBox() YMessageBox yFirstMessageBox()

Commence l'énumération des interfaces de messagerie accessibles par la librairie.

function **FirstMessageBox**( )

Utiliser la fonction `YMessageBox.nextMessageBox()` pour itérer sur les autres interfaces de messagerie.

Retourne :

un pointeur sur un objet `YMessageBox`, correspondant à la première interface de messagerie accessible en ligne, ou `null` si il n'y a pas de interfaces de messagerie disponibles.

YMessageBox.FirstMessageBoxInContext() YMessageBox
yFirstMessageBoxInContext()
YMessageBox.FirstMessageBoxInContext()
YMessageBox.FirstMessageBoxInContext()

Commence l'énumération des interfaces de messagerie accessibles par la librairie.

function **FirstMessageBoxInContext**(**yctx**)

Utiliser la fonction `YMessageBox.nextMessageBox()` pour itérer sur les autres interfaces de messagerie.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YMessageBox`, correspondant à la première interface de messagerie accessible en ligne, ou `null` si il n'y a pas de interfaces de messagerie disponibles.

messagebox→clearCache() YMessageBox
messagebox.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'interface de messagerie. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

messagebox→clearPduCounters() YMessageBox
messagebox.clearPduCounters()
messagebox.clearPduCounters()

Réinitialise les compteurs d'unités SMS transmises et reçues.

function **clearPduCounters**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

messagebox→describe()messagebox.describe() YMessageBox

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface de messagerie au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant l'interface de messagerie (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

messagebox→get_advertisedValue() YMessageBox
messagebox→advertisedValue()
messagebox.get_advertisedValue()
messagebox.get_advertisedValue()

Retourne la valeur courante de l'interface de messagerie (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :

une chaîne de caractères représentant la valeur courante de l'interface de messagerie (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

messagebox→get_errorMessage() YMessageBox
messagebox→errorMessage()
messagebox.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface de messagerie.

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 de messagerie.

messagebox→get_errorType() YMessageBox
messagebox→errorType()
messagebox.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface de messagerie.

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 de messagerie.

messagebox→get_friendlyName() YMessageBox
messagebox→friendlyName()
messagebox.get_friendlyName()

Retourne un identifiant global de l'interface de messagerie au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et de l'interface de messagerie si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel de l'interface de messagerie (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant l'interface de messagerie en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

messagebox→get_functionDescriptor() YMessageBox
messagebox→functionDescriptor()
messagebox.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

messagebox→get_functionId() YMessageBox
messagebox→functionId()
messagebox.get_functionId()

Retourne l'identifiant matériel de l'interface de messagerie, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant l'interface de messagerie (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

messagebox→get_hardwareId() YMessageBox
messagebox→hardwareId()
messagebox.get_hardwareId()

Retourne l'identifiant matériel unique de l'interface de messagerie 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 de messagerie (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant l'interface de messagerie (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

messagebox→get_logicalName() YMessageBox
messagebox→logicalName()
messagebox.get_logicalName()
messagebox.get_logicalName()

Retourne le nom logique de l'interface de messagerie.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique de l'interface de messagerie.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

messagebox→get_messages() YMessageBox
messagebox→messages()
messagebox.get_messages()
messagebox.get_messages()

Retourne la liste des messages reçus et non effacés.

```
function get_messages( )
```

Cette fonction décode automatiquement les SMS concaténés.

Retourne :
une liste d'objets YSms.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

messagebox→get_module() YMessageBox
messagebox→module()messagebox.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`

messagebox→get_pduReceived() YMessageBox
messagebox→pduReceived()
messagebox.get_pduReceived()
messagebox.get_pduReceived()

Retourne le nombre d'unités SMS reçues jusqu'à présent.

function **get_pduReceived**( )

Retourne :
un entier représentant le nombre d'unités SMS reçues jusqu'à présent

En cas d'erreur, déclenche une exception ou retourne `Y_PDURECEIVED_INVALID`.

messagebox→get_pduSent() YMessageBox
messagebox→pduSent()messagebox.get_pduSent()
messagebox.get_pduSent()

Retourne le nombre d'unités SMS envoyées jusqu'à présent.

function **get_pduSent**( )

Retourne :
un entier représentant le nombre d'unités SMS envoyées jusqu'à présent

En cas d'erreur, déclenche une exception ou retourne `Y_PDUSENT_INVALID`.

messagebox→get_slotsCount() YMessageBox
messagebox→slotsCount()
messagebox.get_slotsCount()
messagebox.get_slotsCount()

Retourne le nombre total de positions de stockage dans la carte SIM.

function **get_slotsCount**( )

Retourne :
un entier représentant le nombre total de positions de stockage dans la carte SIM

En cas d'erreur, déclenche une exception ou retourne `Y_SLOTSCOUNT_INVALID`.

messagebox→get_slotsInUse() YMessageBox
messagebox→slotsInUse()
messagebox.get_slotsInUse()
messagebox.get_slotsInUse()

Retourne le nombre de positions de stockage utilisées.

function **get_slotsInUse**( )

Retourne :
un entier représentant le nombre de positions de stockage utilisées

En cas d'erreur, déclenche une exception ou retourne `Y_SLOTSINUSE_INVALID`.

messagebox→get_userData() YMessageBox
messagebox→userData()messagebox.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.

messagebox→isOnline()messagebox.isOnline() YMessageBox

Vérifie si le module hébergeant l'interface de messagerie est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache de l'interface de messagerie sont valides au moment de l'appel, le module est considéré joignable. 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 de messagerie est joignable, `false` sinon

messagebox→load()messagebox.load() YMessageBox

Met en cache les valeurs courantes de l'interface de messagerie, 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 occasionellement 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.

messagebox→loadAttribute() YMessageBox
messagebox.loadAttribute()
messagebox.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

messagebox→muteValueCallbacks() YMessageBox
messagebox.muteValueCallbacks()
messagebox.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

messagebox→newMessage() YMessageBox
messagebox.newMessage()
messagebox.newMessage()

Crée un nouveau message SMS vide, qui pourra ensuite être librement paramétré puis envoyé.

function **newMessage**(**recipient**)

Paramètres :

recipient une chaîne de caractères contenant le numéro de téléphone, du destinataire, soit au format national, soit au format international commençant par un signe plus

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

messagebox→nextMessageBox() YMessageBox
messagebox.nextMessageBox()
messagebox.nextMessageBox()

Continue l'énumération des interfaces de messagerie commencée à l'aide de `yFirstMessageBox()`.

function **nextMessageBox**( )

Retourne :

un pointeur sur un objet `YMessageBox` accessible en ligne, ou `null` lorsque l'énumération est terminée.

messagebox→registerValueCallback() YMessageBox
messagebox.registerValueCallback()
messagebox.registerValueCallback()

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.

messagebox→sendFlashMessage() YMessageBox
messagebox.sendFlashMessage()
messagebox.sendFlashMessage()

Envoie un SMS "Flash", aussi appelé message de classe 0.

function **sendFlashMessage**(**recipient**, **message**)

Les messages flash s'affichent directement sur l'écran du téléphone du destinataire, et ne sont en principe pas sauvés sur la carte SIM. Cette fonction est capable d'envoyer des messages de plus de 160 caractères, à l'aide de la technique de concaténation de SMS. Les caractères accentués de l'alphabet ISO-latin sont supportés. Pour envoyer des messages avec des caractères unicodes plus spéciaux tels que des caractères asiatiques et des émoticônes, créez un message avec la méthode `newMessage` et définissez son contenu avec la méthode `addText` et `addUnicodeData`.

Paramètres :

recipient une chaîne de caractères contenant le numéro de téléphone, du destinataire, soit au format national, soit au format international commençant par un signe plus

message le texte du message à envoyer

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

messagebox→sendTextMessage() YMessageBox
messagebox.sendTextMessage()
messagebox.sendTextMessage()

Envoie un SMS textuel, avec les paramètres standards.

function **sendTextMessage**(**recipient**, **message**)

Cette fonction est capable d'envoyer des messages de plus de 160 caractères, à l'aide de la technique de concaténation de SMS. Les caractères accentués de l'alphabet ISO-latin sont supportés. Pour envoyer des messages avec des caractères unicodes plus spéciaux tels que des caractères asiatiques et des émoticônes, créez un message avec la méthode `newMessage` et définissez son contenu avec les méthodes `addText` et `addUnicodeData`.

Paramètres :

recipient une chaîne de caractères contenant le numéro de téléphone, du destinataire, soit au format national, soit au format international commençant par un signe plus

message le texte du message à envoyer

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

messagebox→set_logicalName() YMessageBox
messagebox→setLogicalName()
messagebox.set_logicalName()
messagebox.set_logicalName()

Modifie le nom logique de l'interface de messagerie.

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 de messagerie.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

messagebox→set_pduReceived() YMessageBox
messagebox→setPduReceived()
messagebox.set_pduReceived()
messagebox.set_pduReceived()

Modifie la valeur du compteur d'unités SMS reçues.

function **set_pduReceived**(**newval**)

Paramètres :

newval un entier représentant la valeur du compteur d'unités SMS reçues

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

messagebox→set_pduSent() YMessageBox
messagebox→setPduSent()
messagebox.set_pduSent()
messagebox.set_pduSent()

Modifie la valeur du compteur d'unités SMS envoyées.

function **set_pduSent**(**newval**)

Paramètres :

newval un entier représentant la valeur du compteur d'unités SMS envoyées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

messagebox→set_userData() YMessageBox
messagebox→setUserData()
messagebox.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

messagebox→unmuteValueCallbacks() YMessageBox
messagebox.unmuteValueCallbacks()
messagebox.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

messagebox→wait_async() YMessageBox
messagebox.wait_async()

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 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>
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;
uwp     import com.yoctopuce.YoctoAPI.YModule;
py      from yocto_api import *
php     require_once('yocto_api.php');
es      in HTML: <script src="../../lib/yocto_api.js"></script>
        in node.js: require('yoctolib-es2017/yocto_api.js');
```

Fonction globales
yFindModule(**func**) Permet de retrouver un module d'après son numéro de série ou son nom logique.
yFindModuleInContext(**yctx**, **func**) Permet de retrouver un module d'après un identifiant donné dans un Context YAPI.
yFirstModule() Commence l'énumération des modules accessibles par la librairie.
Méthodes des objets YModule
module→checkFirmware(**path**, **onlynew**) Teste si le fichier byn est valide pour le module.
module→clearCache() Invalide le cache.
module→describe() Retourne un court texte décrivant le module.
module→download(**pathname**) Télécharge le fichier choisi du module et retourne son contenu.
module→functionBaseType(**functionIndex**) Retourne le type de base de la *n*ième fonction du module.
module→functionCount() Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.
module→functionId(**functionIndex**) Retourne l'identifiant matériel de la *n*ième fonction du module.
module→functionName(**functionIndex**) Retourne le nom logique de la *n*ième fonction du module.
module→functionType(**functionIndex**) Retourne le type de la *n*ième fonction du module.
module→functionValue(**functionIndex**) Retourne la valeur publiée par la *n*ième fonction du module.
module→get_allSettings()

Retourne tous les paramètres de configuration 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_functionIds(**funType**)
Retourne les identifiants matériels des fonctions correspondant au type passé en argument.

module→get_hardwareId()
Retourne l'identifiant unique du module.

module→get_icon2d()
Retourne l'icône du module.

module→get_lastLogs()
Retourne une chaine de charactère 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_parentHub()
Retourne le numéro de série du YoctoHub sur lequel est connecté le module.

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_subDevices()
Retourne la liste des modules branchés au module courant.

module→get_upTime()
Retourne le numbre de millisecondes écoulées depuis la mise sous tension du module

module→get_url()
Retourne l'URL utilisée pour accéder au module.

module→get_usbCurrent()
Retourne le courant consommé par le module sur le bus USB, en milliampères.

module→get_userData()
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

module→get_userVar()

Retourne la valeur entière précédemment stockée dans cet attribut.

module→hasFunction(**funcId**)

Teste la présence d'une fonction pour le module courant.

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→log(**text**)

Ajoute un message arbitraire dans les logs du module.

module→nextModule()

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

module→reboot(**secBeforeReboot**)

Agende un simple redémarrage du module dans un nombre donné de secondes.

module→registerLogCallback(**callback**)

Enregistre une fonction de callback qui sera appelée à chaque fois le module émet un message de log.

module→revertFromFlash()

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

module→saveToFlash()

Sauve les réglages courants dans la mémoire non volatile du module.

module→set_allSettings(**settings**)

Rétablit tous les paramètres du module.

module→set_allSettingsAndFiles(**settings**)

Rétablit tous les paramètres de configuration et fichiers sur un module.

module→set_beacon(**newval**)

Allume ou éteint la balise de localisation du module.

module→set_logicalName(**newval**)

Change le nom logique du module.

module→set_luminosity(**newval**)

Modifie la luminosité des leds informatives du module.

module→set_userData(**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→set_userVar(**newval**)

Stocke une valeur 32 bits dans la mémoire volatile du module.

module→triggerFirmwareUpdate(**secBeforeReboot**)

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

module→updateFirmware(**path**)

Prepare une mise à jour de firmware du module.

module→updateFirmwareEx(**path**, **force**)

Prepare une mise à jour de firmware du module.

module→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YModule.FindModule() YModule yFindModule()YModule.FindModule() YModule.FindModule()

Permet de retrouver un module d'après son numéro de série ou son nom logique.

function **FindModule**(**func**)

Cette fonction n'exige pas que le module soit en ligne au moment ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindModuleInContext() YModule
yFindModuleInContext()
YModule.FindModuleInContext()
YModule.FindModuleInContext()

Permet de retrouver un module d'après un identifiant donné dans un Context YAPI.

function **FindModuleInContext**(**yctx**, **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 module soit en ligne au moment ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le module sans ambiguïté

Retourne :

un objet de classe `YModule` qui permet ensuite de contrôler le module.

YModule.FirstModule() YModule yFirstModule()YModule.FirstModule() YModule.FirstModule()

Commence l'énumération des modules accessibles par la librairie.

function **FirstModule**( )

Utiliser la fonction `YModule.nextModule()` pour itérer sur les autres modules.

Retourne :

un pointeur sur un objet `YModule`, correspondant au premier module accessible en ligne, ou `null` si aucun module n'a été trouvé.

module→checkFirmware()module.checkFirmware() YModule module.checkFirmware()

Teste si le fichier byn est valide pour le module.

function **checkFirmware**(**path**, **onlynew**)

Cette méthode est utile pour vérifier si il est nécessaire de mettre à jour le module avec un nouveau firmware. Il est possible de passer un répertoire qui contiens plusieurs fichier `.byn`. Dans ce cas cette methode retourne le path du fichier `.byn` compatible le plus récent. Si le parametre `onlynew` est vrais, les firmwares équivalents ou plus anciens que le firmware actuellement installé sont ignorés.

Paramètres :

path le path d'un fichier `.byn` ou d'un répertoire contenant plusieurs fichier `.byn`

onlynew retourne uniquement les fichiers strictement plus récents

Retourne :

le path du fichier `.byn` à utiliser, ou une chaîne vide si aucun firmware plus récent n'est disponible En cas d'erreur, déclenche une exception ou retourne une chaine de caractère qui comment par "error:".

module→clearCache()module.clearCache() YModule

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du module. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 module.download()

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 `YAPI_INVALID_STRING`.

module→functionBaseType() YModule
module.functionBaseType()

Retourne le type de base de la *n*ième fonction du module.

function **functionBaseType**(**functionIndex**)

Par exemple, le type de base de toutes les fonctions de mesure est "Sensor".

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 type de base de la fonction

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

module→functionCount()module.functionCount() YModule

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

function **functionCount**( )

Retourne :
le nombre de fonctions sur le module

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→functionId()module.functionId() YModule

Retourne l'identifiant matériel de la *n*ième fonction du module.

function **functionId**(**functionIndex**)

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant à l'identifiant matériel unique de la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

module→functionName()module.functionName() YModule

Retourne le nom logique de la *n*ième fonction du module.

function **functionName**(**functionIndex**)

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant au nom logique de la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

module→functionType()module.functionType() YModule

Retourne le type de la *n*ième fonction du module.

function **functionType**(**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 type de la fonction

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

module→functionValue()module.functionValue() YModule

Retourne la valeur publiée par la *n*ième fonction du module.

function **functionValue**(**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_allSettings() YModule
module→allSettings()module.get_allSettings()
module.get_allSettings()

Retourne tous les paramètres de configuration du module.

function **get_allSettings**( )

Utile pour sauvgarder les noms logiques, les calibrations et fichies uploadés d'un module.

Retourne :
un objet binaire avec tous les paramètres

En cas d'erreur, déclenche une exception ou retourne un objet binaire de taille 0.

module→get_beacon() YModule
module→beacon()module.get_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()
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_functionIds() YModule
module→functionIds()module.get_functionIds()
module.get_functionIds()

Retourne les identifiants matériels des fonctions correspondant au type passé en argument.

function **get_functionIds**(**funType**)

Paramètres :

funType Le type de fonction (Relay, LightSensor, Voltage,...)

Retourne :

un tableau de chaînes de caractère.

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()
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. En cas d'erreur, déclenche une exception ou retourne `YAPI_INVALID_STRING`.

module→get_lastLogs() YModule
module→lastLogs()module.get_lastLogs()
module.get_lastLogs()

Retourne une chaine de charactère contenant les derniers logs du module.

function **get_lastLogs**( )

Cette méthode retourne les derniers logs qui sont encore stocké dans le module.

Retourne :

une chaîne de caractère contenant les derniers logs du module. En cas d'erreur, déclenche une exception ou retourne `YAPI_INVALID_STRING`.

module→get_logicalName() YModule
module→logicalName()module.get_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()
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()
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()
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()
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()
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()
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()
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()
module.get_upTime()

Retourne le numbre de millisecondes écoulées depuis la mise sous tension du module

function **get_upTime**( )

Retourne :
un entier représentant le numbre de millisecondes écoulées depuis la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne `Y_UPTIME_INVALID`.

module→get_usbCurrent() YModule
module→usbCurrent()module.get_usbCurrent()
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→get_userVar() YModule
module→userVar()module.get_userVar()
module.get_userVar()

Retourne la valeur entière précédemment stockée dans cet attribut.

function **get_userVar**( )

Au démarrage du module (ou après un redémarrage), la valeur est toujours zéro.

Retourne :
un entier représentant la valeur entière précédemment stockée dans cet attribut

En cas d'erreur, déclenche une exception ou retourne `Y_USERVAR_INVALID`.

module→hasFunction()module.hasFunction() YModule module.hasFunction()

Teste la présence d'une fonction pour le module courant.

```
function hasFunction( funcId)
```

La méthode prend en paramètre l'identifiant de la fonction (relay1, voltage2,...) et retourne un booléen.

Paramètres :

funcId identifiant matériel de la fonction

Retourne :

vrai si le module inclut la fonction demandée

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→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→log()module.log()module.log() YModule

Ajoute un message arbitraire dans les logs du module.

function **log**(**text**)

Cette fonction est utile en particulier pour tracer l'exécution de callbacks HTTP. Si un saut de ligne est désiré après le message, il doit être inclus dans la chaîne de caractère.

Paramètres :

text le message à ajouter aux logs 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→nextModule()module.nextModule() YModule module.nextModule()

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()module.reboot() YModule

Agende un simple redémarrage du module dans un nombre donné de secondes.

```
function reboot( secBeforeReboot)
```

Paramètres :

secBeforeReboot nombre de secondes avant de redémarrer

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→revertFromFlash() YModule
module.revertFromFlash()module.revertFromFlash()

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

function **revertFromFlash**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→saveToFlash()module.saveToFlash() YModule module.saveToFlash()

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'appelez pas cette fonction dans une boucle.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→set_allSettings() YModule
module→setAllSettings()module.set_allSettings()
module.set_allSettings()

Rétablit tous les paramètres du module.

function **set_allSettings**(**settings**)

Utile pour restorer les noms logiques et les calibrations du module depuis une sauvgarde. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si les réglages doivent être préservés.

Paramètres :

settings un objet binaire avec touts les paramètres

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→set_allSettingsAndFiles() YModule
module→setAllSettingsAndFiles()
module.set_allSettingsAndFiles()
module.set_allSettingsAndFiles()

Rétablit tous les paramètres de configuration et fichiers sur un module.

function **set_allSettingsAndFiles**(**settings**)

Cette méthode est utile pour récupérer les noms logiques, les calibrations, les fichiers uploadés, etc. du module depuis une sauvgarde. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si les réglages doivent être préservés.

Paramètres :

settings un buffer binaire avec touts les paramètres

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→set_beacon() YModule
module→setBeacon()module.set_beacon()
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()
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()
module.set_luminosity()

Modifie la luminosité des leds informatives du module.

function **set_luminosity**(**newval**)

Le paramêtre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la luminosité des leds informatives du module

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→set_userData() YModule
module→setUserData()module.set_userData()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

```
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→set_userVar() YModule
module→setUserVar()module.set_userVar()
module.set_userVar()

Stocke une valeur 32 bits dans la mémoire volatile du module.

function **set_userVar**(**newval**)

Cet attribut est à la disposition du programmeur pour y stocker par exemple une variable d'état. Au démarrage du module (ou après un redémarrage), la valeur est toujours zéro.

Paramètres :

newval un entier

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→triggerFirmwareUpdate() YModule
module.triggerFirmwareUpdate()
module.triggerFirmwareUpdate()

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→updateFirmware()module.updateFirmware() YModule module.updateFirmware()

Prepare une mise à jour de firmware du module.

function **updateFirmware**(**path**)

Cette méthode retourne un object `YFirmwareUpdate` qui est utilisé pour mettre à jour le firmware du module.

Paramètres :

path le path du fichier `.byn` à utiliser

Retourne :

un object `YFirmwareUpdate` ou NULL en cas d'erreur

module→updateFirmwareEx() YModule
module.updateFirmwareEx()
module.updateFirmwareEx()

Prepare une mise à jour de firmware du module.

function **updateFirmwareEx**(**path**, **force**)

Cette méthode retourne un object `YFirmwareUpdate` qui est utilisé pour mettre à jour le firmware du module.

Paramètres :

path le path du fichier `.byn` à utiliser

force vrai pour forceer la mise à jour même si un prérequis ne semble pas satisfait

Retourne :

un object `YFirmwareUpdate` ou NULL en cas d'erreur

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.

3.41. Interface de la fonction Motor

La librairie de programmation yoctopuce permet de piloter la puissance envoyée au moteur pour le faire tourner aussi bien dans un sens que dans l'autre, mais aussi de piloter des accélérations linéaires: le moteur accélére alors tout seul sans que vous vous ayez à vous en occuper. La librairie permet aussi de freiner le moteur: cela est réalisé en court-circuitant les pôles du moteur, ce qui le transforme en frein électro-magnétique.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_motor.js'></script>
cpp     #include "yocto_motor.h"
m       #import "yocto_motor.h"
pas     uses yocto_motor;
vb      yocto_motor.vb
cs      yocto_motor.cs
java    import com.yoctopuce.YoctoAPI.YMotor;
uwp     import com.yoctopuce.YoctoAPI.YMotor;
py      from yocto_motor import *
php     require_once('yocto_motor.php');
es      in HTML: <script src="../../lib/yocto_motor.js"></script>
        in node.js: require('yoctolib-es2017/yocto_motor.js');
```

Fonction globales
yFindMotor(**func**) Permet de retrouver un moteur d'après un identifiant donné.
yFindMotorInContext(**yctx**, **func**) Permet de retrouver un moteur d'après un identifiant donné dans un Context YAPI.
yFirstMotor() Commence l'énumération des moteur accessibles par la librairie.
yFirstMotorInContext(**yctx**) Commence l'énumération des moteur accessibles par la librairie.
Méthodes des objets `YMotor`
motor→brakingForceMove(**targetPower**, **delay**) Modifie progressivement la force de freinage appliquée au moteur sur une durée donnée.
motor→clearCache() Invalide le cache.
motor→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du moteur au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
motor→drivingForceMove(**targetPower**, **delay**) Modifie progressivement la puissance envoyée au moteur sur une durée donnée.
motor→get_advertisedValue() Retourne la valeur courante du moteur (pas plus de 6 caractères).
motor→get_brakingForce() Retourne la force de freinage appliquée au moteur, sous forme de pourcentage.
motor→get_cutOffVoltage() Retourne la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.
motor→get_drivingForce()

Retourne la puissance actuelle envoyée au moteur, sous forme de nombre réel entre -100% et +100%.

motor→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moteur.

motor→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moteur.

motor→get_failSafeTimeout()

Retourne le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

motor→get_frequency()

Retourne la fréquence du signal PWM utilisé pour contrôler le moteur.

motor→get_friendlyName()

Retourne un identifiant global du moteur au format `NOM_MODULE.NOM_FONCTION`.

motor→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

motor→get_functionId()

Retourne l'identifiant matériel du moteur, sans référence au module.

motor→get_hardwareId()

Retourne l'identifiant matériel unique du moteur au format `SERIAL.FUNCTIONID`.

motor→get_logicalName()

Retourne le nom logique du moteur.

motor→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

motor→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

motor→get_motorStatus()

Retourne l'état du contrôleur de moteur.

motor→get_overCurrentLimit()

Retourne la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

motor→get_starterTime()

Retourne la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

motor→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

motor→isOnline()

Vérifie si le module hébergeant le moteur est joignable, sans déclencher d'erreur.

motor→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le moteur est joignable, sans déclencher d'erreur.

motor→keepALive()

Réarme la sécurité failsafe du contrôleur.

motor→load(**msValidity**)

Met en cache les valeurs courantes du moteur, avec une durée de validité spécifiée.

motor→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

motor→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du moteur, avec une durée de validité spécifiée.

motor→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

motor→nextMotor()

Continue l'énumération des moteur commencée à l'aide de `yFirstMotor()`.

motor→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

motor→resetStatus()

Réinitialise l'état du contrôleur à IDLE.

motor→set_brakingForce(**newval**)

Modifie immédiatement la force de freinage appliquée au moteur (en pourcents).

motor→set_cutOffVoltage(**newval**)

Modifie la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

motor→set_drivingForce(**newval**)

Modifie immédiatement la puissance envoyée au moteur.

motor→set_failSafeTimeout(**newval**)

Modifie le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

motor→set_frequency(**newval**)

Modifie la fréquence du signal PWM utilisée pour contrôler le moteur.

motor→set_logicalName(**newval**)

Modifie le nom logique du moteur.

motor→set_overCurrentLimit(**newval**)

Modifie la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

motor→set_starterTime(**newval**)

Modifie la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

motor→set_userData(**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`.

motor→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

motor→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YMotor.FindMotor() YMotor yFindMotor()YMotor.FindMotor()YMotor.FindMotor()

Permet de retrouver un moteur d'après un identifiant donné.

```
function FindMotor( 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 moteur soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YMotor.isOnline()` pour tester si le moteur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le moteur sans ambiguïté

Retourne :

un objet de classe `YMotor` qui permet ensuite de contrôler le moteur.

YMotor.FindMotorInContext() YMotor yFindMotorInContext()YMotor.FindMotorInContext() YMotor.FindMotorInContext()

Permet de retrouver un moteur d'après un identifiant donné dans un Context YAPI.

```
function FindMotorInContext( yctx, 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 moteur soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YMotor.isOnline()` pour tester si le moteur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence le moteur sans ambiguïté

Retourne :

un objet de classe `YMotor` qui permet ensuite de contrôler le moteur.

YMotor.FirstMotor() YMotor yFirstMotor()YMotor.FirstMotor()YMotor.FirstMotor()

Commence l'énumération des moteur accessibles par la librairie.

function **FirstMotor**( )

Utiliser la fonction `YMotor.nextMotor()` pour itérer sur les autres moteur.

Retourne :

un pointeur sur un objet `YMotor`, correspondant au premier moteur accessible en ligne, ou `null` si il n'y a pas de moteur disponibles.

YMotor.FirstMotorInContext() YMotor yFirstMotorInContext()YMotor.FirstMotorInContext() YMotor.FirstMotorInContext()

Commence l'énumération des moteur accessibles par la librairie.

function **FirstMotorInContext**(**yctx**)

Utiliser la fonction `YMotor.nextMotor()` pour itérer sur les autres moteur.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YMotor`, correspondant au premier moteur accessible en ligne, ou `null` si il n'y a pas de moteur disponibles.

motor→brakingForceMove() YMotor
motor.brakingForceMove()motor.brakingForceMove()

Modifie progressivement la force de freinage appliquée au moteur sur une durée donnée.

function **brakingForceMove**(**targetPower**, **delay**)

Paramètres :

targetPower	force de freinage finale, en pourcentage
delay	durée (en ms) sur laquelle le changement de puissance sera effectué

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→clearCache()motor.clearCache() YMotor

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du moteur. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

motor→describe()motor.describe() YMotor

Retourne un court texte décrivant de manière non-ambigüe l'instance du moteur au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le moteur (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

motor→drivingForceMove() YMotor
motor.drivingForceMove()motor.drivingForceMove()

Modifie progressivement la puissance envoyée au moteur sur une durée donnée.

function **drivingForceMove**(**targetPower**, **delay**)

Paramètres :

targetPower	puissance finale désirée, en pourcentage de -100% à +100%
delay	durée (en ms) sur laquelle le changement de puissance sera effectué

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→get_advertisedValue() YMotor
motor→advertisedValue()
motor.get_advertisedValue()
motor.get_advertisedValue()

Retourne la valeur courante du moteur (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante du moteur (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

motor→get_brakingForce() YMotor
motor→brakingForce()motor.get_brakingForce()
motor.get_brakingForce()

Retourne la force de freinage appliquée au moteur, sous forme de pourcentage.

function **get_brakingForce**( )

La valeur 0 correspond ne pas freiner (moteur en roue libre).

Retourne :
une valeur numérique représentant la force de freinage appliquée au moteur, sous forme de pourcentage

En cas d'erreur, déclenche une exception ou retourne `Y_BRAKINGFORCE_INVALID`.

motor→get_cutOffVoltage() YMotor
motor→cutOffVoltage()motor.get_cutOffVoltage()
motor.get_cutOffVoltage()

Retourne la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

function **get_cutOffVoltage**( )

Ce réglage permet d'éviter d'endommager un accumulateur un continuant à l'utiliser une fois "vide".

Retourne :

une valeur numérique représentant la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation

En cas d'erreur, déclenche une exception ou retourne `Y_CUTOFFVOLTAGE_INVALID`.

motor→get_drivingForce() YMotor
motor→drivingForce()motor.get_drivingForce()
motor.get_drivingForce()

Retourne la puissance actuelle envoyée au moteur, sous forme de nombre réel entre -100% et +100%.

function **get_drivingForce**( )

Retourne :

une valeur numérique représentant la puissance actuelle envoyée au moteur, sous forme de nombre réel entre -100% et +100%

En cas d'erreur, déclenche une exception ou retourne `Y_DRIVINGFORCE_INVALID`.

motor→get_errorMessage() YMotor
motor→errorMessage()motor.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moteur.

```
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 moteur.

motor→get_errorType() YMotor
motor→errorType()motor.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moteur.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :
un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du moteur.

motor→get_failSafeTimeout() YMotor
motor→failSafeTimeout()motor.get_failSafeTimeout()
motor.get_failSafeTimeout()

Retourne le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

function **get_failSafeTimeout**( )

Passé ce delai, le contrôleur arrêtera le moteur et passera en mode erreur FAILSAFE. La sécurité failsafe est désactivée quand la valeur est à zéro.

Retourne :

un entier représentant le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle

En cas d'erreur, déclenche une exception ou retourne `Y_FAILSAFETIMEOUT_INVALID`.

motor→get_frequency() YMotor
motor→frequency()motor.get_frequency()
motor.get_frequency()

Retourne la fréquence du signal PWM utilisé pour contrôler le moteur.

function **get_frequency**( )

Retourne :
une valeur numérique représentant la fréquence du signal PWM utilisé pour contrôler le moteur

En cas d'erreur, déclenche une exception ou retourne `Y_FREQUENCY_INVALID`.

motor→get_friendlyName() YMotor
motor→friendlyName()motor.get_friendlyName()

Retourne un identifiant global du moteur au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du moteur si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du moteur (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le moteur en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

motor→get_functionDescriptor() YMotor
motor→functionDescriptor()
motor.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

motor→get_functionId() YMotor
motor→functionId()motor.get_functionId()

Retourne l'identifiant matériel du moteur, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le moteur (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

motor→get_hardwareId() YMotor
motor→hardwareId()motor.get_hardwareId()

Retourne l'identifiant matériel unique du moteur au format SERIAL.FUNCTIONID.

function **get_hardwareId**( )

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du moteur (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le moteur (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

motor→get_logicalName() YMotor
motor→logicalName()motor.get_logicalName()
motor.get_logicalName()

Retourne le nom logique du moteur.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du moteur.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

motor→get_module() YMotor
motor→module()motor.get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

function **get_module**( )

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :
une instance de YModule

motor→get_motorStatus() YMotor
motor→motorStatus()motor.get_motorStatus()
motor.get_motorStatus()

Retourne l'état du contrôleur de moteur.

function **get_motorStatus**( )

Les états possibles sont: IDLE si le moteur est à l'arrêt/en roue libre, prêt à démarrer; FORWD si le contrôleur fait tourner le moteur en marche avant; BACKWD si le contrôleur fait tourner le moteur en marche arrière; BRAKE si le contrôleur est en train de freiner; LOVOLT si le contrôleur a détecté une tension trop basse; HICURR si le contrôleur a détecté une surconsommation; HIHEAT si le contrôleur a détecté une surchauffe; FAILSF si le contrôleur est passé en protection failsafe.

Si le contrôleur est en erreur (LOVOLT, HICURR, HIHEAT,FAILSF), il doit être explicitement réinitialisé avec la fonction `resetStatus`.

Retourne :

une valeur parmi `Y_MOTORSTATUS_IDLE`, `Y_MOTORSTATUS_BRAKE`, `Y_MOTORSTATUS_FORWD`, `Y_MOTORSTATUS_BACKWD`, `Y_MOTORSTATUS_LOVOLT`, `Y_MOTORSTATUS_HICURR`, `Y_MOTORSTATUS_HIHEAT` et `Y_MOTORSTATUS_FAILSF` représentant l'état du contrôleur de moteur

En cas d'erreur, déclenche une exception ou retourne `Y_MOTORSTATUS_INVALID`.

motor→get_overCurrentLimit() YMotor
motor→overCurrentLimit()
motor.get_overCurrentLimit()
motor.get_overCurrentLimit()

Retourne la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

function **get_overCurrentLimit**( )

Une valeur nulle signifie qu'aucune limite n'est définie.

Retourne :

un entier représentant la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur

En cas d'erreur, déclenche une exception ou retourne `Y_OVERCURRENTLIMIT_INVALID`.

motor→get_starterTime() YMotor
motor→starterTime()motor.get_starterTime()
motor.get_starterTime()

Retourne la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

function **get_starterTime**( )

Retourne :
un entier représentant la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage

En cas d'erreur, déclenche une exception ou retourne `Y_STARTERTIME_INVALID`.

motor→get_userData() YMotor
motor→userData()motor.get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

function **get_userData**( )

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.

motor→isOnline()motor.isOnline() YMotor

Vérifie si le module hébergeant le moteur est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du moteur sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si le moteur est joignable, `false` sinon

motor→keepALive()motor.keepALive() YMotor motor.keepALive()

Réarme la sécurité failsafe du contrôleur.

```
function keepALive( )
```

Lorsque le moteur est en marche et que la sécurité failsafe est activée, cette fonction doit être appelée périodiquement pour confirmer le bon fonctionnement du processus de contrôle. A défaut, le moteur s'arrêtera automatiquement au bout du temps prévu. Notez que l'appel à une fonction de type *set* du moteur réarme aussi la sécurité failsafe.

motor→load()motor.load() YMotor

Met en cache les valeurs courantes du moteur, avec une durée de validité spécifiée.

function **load**(**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 occasionellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→loadAttribute()motor.loadAttribute() YMotor motor.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

motor→muteValueCallbacks() YMotor
motor.muteValueCallbacks()
motor.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→nextMotor()motor.nextMotor() YMotor motor.nextMotor()

Continue l'énumération des moteur commencée à l'aide de `yFirstMotor()`.

function **nextMotor**( )

Retourne :

un pointeur sur un objet `YMotor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

motor→registerValueCallback() YMotor
motor.registerValueCallback()
motor.registerValueCallback()

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.

motor→resetStatus()motor.resetStatus() YMotor motor.resetStatus()

Réinitialise l'état du contrôleur à IDLE.

```
function resetStatus( )
```

Cette fonction doit être explicitement appelée après toute condition d'erreur pour permettre au contrôleur de repartir.

motor→set_brakingForce() YMotor
motor→setBrakingForce()motor.set_brakingForce()
motor.set_brakingForce()

Modifie immédiatement la force de freinage appliquée au moteur (en pourcents).

function **set_brakingForce**(**newval**)

La valeur 0 correspond à ne pas freiner (moteur en roue libre). Lorsque la force de freinage est changée, la puissance de traction est remise à zéro.

Paramètres :

newval une valeur numérique représentant immédiatement la force de freinage appliquée au moteur (en pourcents)

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→set_cutOffVoltage() YMotor
motor→setCutOffVoltage()motor.set_cutOffVoltage()
motor.set_cutOffVoltage()

Modifie la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

function **set_cutOffVoltage**(**newval**)

Ce réglage permet d'éviter d'endommager un accumulateur un continuant à l'utiliser une fois "vide". Attention, quel que soit le réglage du cutoff, le variateur passera en erreur si l'alimentation passe (même brièvement) en dessous de 3V.

Paramètres :

newval une valeur numérique représentant la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→set_drivingForce() YMotor
motor→setDrivingForce()motor.set_drivingForce()
motor.set_drivingForce()

Modifie immédiatement la puissance envoyée au moteur.

function **set_drivingForce**(**newval**)

La valeur est donnée en pourcentage de -100% à +100%. Si vous voulez ménager votre mécanique et éviter d'induire des consommations excessives qui pourraient dépasser les capacités du contrôleur, évitez les changements de régime trop brusques. Par exemple, passer brutalement de marche avant à marche arrière est une très mauvaise idée. A chaque fois que la puissance envoyée au moteur est changée, le freinage est remis à zéro.

Paramètres :

newval une valeur numérique représentant immédiatement la puissance envoyée au moteur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→set_failSafeTimeout() YMotor
motor→setFailSafeTimeout()
motor.set_failSafeTimeout()
motor.set_failSafeTimeout()

Modifie le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

```
function set_failSafeTimeout( newval)
```

Passé ce delai, le contrôleur arrêtera le moteur et passera en mode erreur FAILSAFE. La sécurité failsafe est désactivée quand la valeur est à zéro.

Paramètres :

newval un entier représentant le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→set_frequency() YMotor
motor→setFrequency()motor.set_frequency()
motor.set_frequency()

Modifie la fréquence du signal PWM utilisée pour contrôler le moteur.

```
function set_frequency( newval)
```

Une fréquence basse est généralement plus efficace (les composant chauffent moins et le moteur démarre plus facilement), mais un bruit audible peut être généré. Une fréquence élevée peut réduire le bruit, mais il y a plus d'énergie perdue en chaleur.

Paramètres :

newval une valeur numérique représentant la fréquence du signal PWM utilisée pour contrôler le moteur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→set_logicalName() YMotor
motor→setLogicalName()motor.set_logicalName()
motor.set_logicalName()

Modifie le nom logique du moteur.

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 moteur.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→set_overCurrentLimit() YMotor
motor→setOverCurrentLimit()
motor.set_overCurrentLimit()
motor.set_overCurrentLimit()

Modifie la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

function **set_overCurrentLimit**(**newval**)

Une valeur nulle signifie qu'aucune limite n'est définie. Attention, quel que soit le réglage choisi, le variateur passera en erreur si le courant passe, même brièvement, en dessus de 32A.

Paramètres :

newval un entier représentant la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→set_starterTime() YMotor
motor→setStarterTime()motor.set_starterTime()
motor.set_starterTime()

Modifie la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

function **set_starterTime**(**newval**)

Paramètres :

newval un entier représentant la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→set_userData() YMotor
motor→setUserData()motor.set_userData()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

motor→unmuteValueCallbacks() YMotor
motor.unmuteValueCallbacks()
motor.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→wait_async()motor.wait_async() YMotor

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 MultiAxisController

La librairie de programmation Yoctopuce permet de piloter un moteur pas à pas.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_multiaxiscontroller.js'></script>
cpp	#include "yocto_multiaxiscontroller.h"
m	#import "yocto_multiaxiscontroller.h"
pas	uses yocto_multiaxiscontroller;
vb	yocto_multiaxiscontroller.vb
cs	yocto_multiaxiscontroller.cs
java	import com.yoctopuce.YoctoAPI.YMultiAxisController;
uwp	import com.yoctopuce.YoctoAPI.YMultiAxisController;
py	from yocto_multiaxiscontroller import *
php	require_once('yocto_multiaxiscontroller.php');
es	in HTML: <script src="../../lib/yocto_multiaxiscontroller.js"></script> in node.js: require('yoctolib-es2017/yocto_multiaxiscontroller.js');

Fonction globales

yFindMultiAxisController(**func**)

Permet de retrouver un contrôleur multi-axe d'après un identifiant donné.

yFindMultiAxisControllerInContext(**yctx**, **func**)

Permet de retrouver un contrôleur multi-axe d'après un identifiant donné dans un Context YAPI.

yFirstMultiAxisController()

Commence l'énumération des contrôleur multi-axe accessibles par la librairie.

yFirstMultiAxisControllerInContext(**yctx**)

Commence l'énumération des contrôleur multi-axe accessibles par la librairie.

Méthodes des objets YMultiAxisController

multiaxiscontroller→abortAndBrake()

Stoppe le moteur en douceur dès que possible, sans attendre la fin de la commande actuelle.

multiaxiscontroller→abortAndHiZ()

Relâche le contrôle du moteur immédiatement, sans attendre la fin de la commande actuelle.

multiaxiscontroller→clearCache()

Invalide le cache.

multiaxiscontroller→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôleur multi-axe au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

multiaxiscontroller→emergencyStop()

Stoppe le moteur en urgence, sans autre précaution.

multiaxiscontroller→findHomePosition(**speed**)

Lance tous les moteur en arrière aux vitesses spécifiées, pour chercher les origines des axes.

multiaxiscontroller→get_advertisedValue()

Retourne la valeur courante du contrôleur multi-axe (pas plus de 6 caractères).

multiaxiscontroller→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôleur multi-axe.

multiaxiscontroller→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôleur multi-axe.

multiaxiscontroller→get_friendlyName()

Retourne un identifiant global du contrôleur multi-axe au format NOM_MODULE.NOM_FONCTION.

multiaxiscontroller→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

multiaxiscontroller→get_functionId()

Retourne l'identifiant matériel du contrôleur multi-axe, sans référence au module.

multiaxiscontroller→get_globalState()

Retourne l'état de fonctionnement global de l'ensemble des moteurs.

multiaxiscontroller→get_hardwareId()

Retourne l'identifiant matériel unique du contrôleur multi-axe au format SERIAL.FUNCTIONID.

multiaxiscontroller→get_logicalName()

Retourne le nom logique du contrôleur multi-axe.

multiaxiscontroller→get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

multiaxiscontroller→get_module_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

multiaxiscontroller→get_nAxis()

Retourne le nombre de contrôleurs à synchroniser.

multiaxiscontroller→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

multiaxiscontroller→isOnline()

Vérifie si le module hébergeant le contrôleur multi-axe est joignable, sans déclencher d'erreur.

multiaxiscontroller→isOnline_async(callback, context)

Vérifie si le module hébergeant le contrôleur multi-axe est joignable, sans déclencher d'erreur.

multiaxiscontroller→load(msValidity)

Met en cache les valeurs courantes du contrôleur multi-axe, avec une durée de validité spécifiée.

multiaxiscontroller→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

multiaxiscontroller→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du contrôleur multi-axe, avec une durée de validité spécifiée.

multiaxiscontroller→moveRel(relPos)

Contrôle les moteurs de manière synchrone pour atteindre une position relative donnée.

multiaxiscontroller→moveTo(absPos)

Contrôle les moteurs de manière synchrone pour atteindre une position absolue donnée.

multiaxiscontroller→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

multiaxiscontroller→nextMultiAxisController()

Continue l'énumération des contrôleur multi-axe commencée à l'aide de yFirstMultiAxisController().

multiaxiscontroller→pause(waitMs)

Garde le moteur dans le même état pour la durée spécifiée, avant d'exécuter la commande suivante.

multiaxiscontroller→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

multiaxiscontroller→reset()

Réinitialise tous les controlleurs et quittance toutes les alertes.

multiaxiscontroller→set_logicalName(newval)

Modifie le nom logique du contrôleur multi-axe.

multiaxiscontroller→set_nAxis(newval)

Modifie le nombre de contrôleurs à synchroniser.

multiaxiscontroller→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`.

multiaxiscontroller→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

multiaxiscontroller→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.

YMultiAxisController.FindMultiAxisController() YMultiAxisController yFindMultiAxisController()
YMultiAxisController.FindMultiAxisController()
YMultiAxisController.FindMultiAxisController()

Permet de retrouver un contrôleur multi-axe d'après un identifiant donné.

function **FindMultiAxisController**(**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 contrôleur multi-axe soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YMultiAxisController.isOnline()` pour tester si le contrôleur multi-axe 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le contrôleur multi-axe sans ambiguïté

Retourne :

un objet de classe `YMultiAxisController` qui permet ensuite de contrôler le contrôleur multi-axe.

YMultiAxisController.FindMultiAxisControllerInContext() YMultiAxisController yFindMultiAxisControllerInContext() YMultiAxisController.FindMultiAxisControllerInContext() YMultiAxisController.FindMultiAxisControllerInContext()

Permet de retrouver un contrôleur multi-axe d'après un identifiant donné dans un Context YAPI.

function **FindMultiAxisControllerInContext**(**yctx**, **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 contrôleur multi-axe soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YMultiAxisController.isOnline()` pour tester si le contrôleur multi-axe 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le contrôleur multi-axe sans ambiguïté

Retourne :

un objet de classe `YMultiAxisController` qui permet ensuite de contrôler le contrôleur multi-axe.

YMultiAxisController.FirstMultiAxisController() YMultiAxisController yFirstMultiAxisController() YMultiAxisController.FirstMultiAxisController() YMultiAxisController.FirstMultiAxisController()

Commence l'énumération des contrôleur multi-axe accessibles par la librairie.

function **FirstMultiAxisController**( )

Utiliser la fonction `YMultiAxisController.nextMultiAxisController()` pour itérer sur les autres contrôleur multi-axe.

Retourne :

un pointeur sur un objet `YMultiAxisController`, correspondant au premier contrôleur multi-axe accessible en ligne, ou `null` si il n'y a pas de contrôleur multi-axe disponibles.

YMultiAxisController.FirstMultiAxisControllerInContext() YMultiAxisController.yFirstMultiAxisControllerInContext() YMultiAxisController.FirstMultiAxisControllerInContext() YMultiAxisController.FirstMultiAxisControllerInContext()

Commence l'énumération des contrôleur multi-axe accessibles par la librairie.

function **FirstMultiAxisControllerInContext**(**yctx**)

Utiliser la fonction `YMultiAxisController.nextMultiAxisController()` pour itérer sur les autres contrôleur multi-axe.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YMultiAxisController`, correspondant au premier contrôleur multi-axe accessible en ligne, ou `null` si il n'y a pas de contrôleur multi-axe disponibles.

multiaxiscontroller→abortAndBrake() YMultiAxisController
multiaxiscontroller.abortAndBrake()
multiaxiscontroller.abortAndBrake()

Stoppe le moteur en douceur dès que possible, sans attendre la fin de la commande actuelle.

function **abortAndBrake**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

multiaxiscontroller→abortAndHiZ() YMultiAxisController
multiaxiscontroller.abortAndHiZ()
multiaxiscontroller.abortAndHiZ()

Relâche le contrôle du moteur immédiatement, sans attendre la fin de la commande actuelle.

function **abortAndHiZ**( )

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

multiaxiscontroller→clearCache() YMultiAxisController
multiaxiscontroller.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du contrôleur multi-axe. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

multiaxiscontroller→describe() YMultiAxisController
multiaxiscontroller.describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôleur multi-axe au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le contrôleur multi-axe (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

multiaxiscontroller→emergencyStop() YMultiAxisController
multiaxiscontroller.emergencyStop()
multiaxiscontroller.emergencyStop()

Stoppe le moteur en urgence, sans autre précaution.

function **emergencyStop**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

multiaxiscontroller→findHomePosition() YMultiAxisController
multiaxiscontroller.findHomePosition()
multiaxiscontroller.findHomePosition()

Lance tous les moteur en arrière aux vitesses spécifiées, pour chercher les origines des axes.

function **findHomePosition**(**speed**)

Paramètres :

speed vitesse désirée pour chaque axe, en pas par seconde.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

multiaxiscontroller→get_advertisedValue() YMultiAxisController
multiaxiscontroller→advertisedValue()
multiaxiscontroller.get_advertisedValue()
multiaxiscontroller.get_advertisedValue()

Retourne la valeur courante du contrôleur multi-axe (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante du contrôleur multi-axe (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

multiaxiscontroller→get_errorMessage() YMultiAxisController
multiaxiscontroller→errorMessage()
multiaxiscontroller.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôleur multi-axe.

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 contrôleur multi-axe.

multiaxiscontroller→get_errorType() YMultiAxisController
multiaxiscontroller→errorType()
multiaxiscontroller.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôleur multi-axe.

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 contrôleur multi-axe.

multiaxiscontroller→get_friendlyName() YMultiAxisController
multiaxiscontroller→friendlyName()
multiaxiscontroller.get_friendlyName()

Retourne un identifiant global du contrôleur multi-axe au format `NOM_MODULE.NOM_FONCTION`.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du contrôleur multi-axe si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du contrôleur multi-axe (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le contrôleur multi-axe en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

multiaxiscontroller→get_functionDescriptor() YMultiAxisController
multiaxiscontroller→functionDescriptor()
multiaxiscontroller.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

multiaxiscontroller→get_functionId() YMultiAxisController
multiaxiscontroller→functionId()
multiaxiscontroller.get_functionId()

Retourne l'identifiant matériel du contrôleur multi-axe, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le contrôleur multi-axe (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

multiaxiscontroller→get_globalState() YMultiAxisController
multiaxiscontroller→globalState()
multiaxiscontroller.get_globalState()
multiaxiscontroller.get_globalState()

Retourne l'état de fonctionnement global de l'ensemble des moteurs.

function **get_globalState**( )

Retourne :

une valeur parmi Y_GLOBALSTATE_ABSENT, Y_GLOBALSTATE_ALERT, Y_GLOBALSTATE_HI_Z, Y_GLOBALSTATE_STOP, Y_GLOBALSTATE_RUN et Y_GLOBALSTATE_BATCH représentant l'état de fonctionnement global de l'ensemble des moteurs

En cas d'erreur, déclenche une exception ou retourne Y_GLOBALSTATE_INVALID.

multiaxiscontroller→get_hardwareId() YMultiAxisController
multiaxiscontroller→hardwareId()
multiaxiscontroller.get_hardwareId()

Retourne l'identifiant matériel unique du contrôleur multi-axe au format SERIAL.FUNCTIONID.

function **get_hardwareId**( )

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du contrôleur multi-axe (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le contrôleur multi-axe (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

multiaxiscontroller→get_logicalName() YMultiAxisController
multiaxiscontroller→logicalName()
multiaxiscontroller.get_logicalName()
multiaxiscontroller.get_logicalName()

Retourne le nom logique du contrôleur multi-axe.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du contrôleur multi-axe.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

multiaxiscontroller→get_module() YMultiAxisController
multiaxiscontroller→module()
multiaxiscontroller.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`

multiaxiscontroller→get_nAxis() YMultiAxisController
multiaxiscontroller→nAxis()
multiaxiscontroller.get_nAxis()
multiaxiscontroller.get_nAxis()

Retourne le nombre de contrôleurs à synchroniser.

function **get_nAxis**( )

Retourne :
un entier représentant le nombre de contrôleurs à synchroniser

En cas d'erreur, déclenche une exception ou retourne `Y_NAXIS_INVALID`.

multiaxiscontroller→get_userData() YMultiAxisController
multiaxiscontroller→userData()
multiaxiscontroller.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.

multiaxiscontroller→isOnline() YMultiAxisController
multiaxiscontroller.isOnline()

Vérifie si le module hébergeant le contrôleur multi-axe est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du contrôleur multi-axe sont valides au moment de l'appel, le module est considéré joignable. 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 contrôleur multi-axe est joignable, `false` sinon

multiaxiscontroller→load()multiaxiscontroller.load() YMultiAxisController

Met en cache les valeurs courantes du contrôleur multi-axe, 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 occasionellement 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.

multiaxiscontroller→loadAttribute() YMultiAxisController
multiaxiscontroller.loadAttribute()
multiaxiscontroller.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

multiaxiscontroller→moveRel() YMultiAxisController
multiaxiscontroller.moveRel()
multiaxiscontroller.moveRel()

Contrôle les moteurs de manière synchrone pour atteindre une position relative donnée.

function **moveRel**(**relPos**)

Le temps nécessaire pour atteindre la position dépend des paramètres d'accélération et de vitesse maximale les plus faibles pour l'ensemble des moteurs. La position finale est atteinte pour tous les axes au même moment.

Paramètres :

relPos position relative désirée, en pas depuis la position actuelle.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

multiaxiscontroller→moveTo() YMultiAxisController
multiaxiscontroller.moveTo()
multiaxiscontroller.moveTo()

Contrôle les moteurs de manière synchrone pour atteindre une position absolue donnée.

function **moveTo**(**absPos**)

Le temps nécessaire pour atteindre la position dépend des paramètres d'accélération et de vitesse maximale les plus faibles pour l'ensemble des moteurs. La position finale est atteinte pour tous les axes au même moment.

Paramètres :

absPos position absolue désirée, en pas depuis chaque origine.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

multiaxiscontroller→muteValueCallbacks() YMultiAxisController
multiaxiscontroller.muteValueCallbacks()
multiaxiscontroller.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

multiaxiscontroller→nextMultiAxisController() YMultiAxisController
multiaxiscontroller.nextMultiAxisController()
multiaxiscontroller.nextMultiAxisController()

Continue l'énumération des contrôleur multi-axe commencée à l'aide de `yFirstMultiAxisController()`.

function **nextMultiAxisController**( )

Retourne :

un pointeur sur un objet `YMultiAxisController` accessible en ligne, ou `null` lorsque l'énumération est terminée.

multiaxiscontroller→pause() YMultiAxisController
multiaxiscontroller.pause()
multiaxiscontroller.pause()

Garde le moteur dans le même état pour la durée spécifiée, avant d'exécuter la commande suivante.

function **pause**(**waitMs**)

Paramètres :

waitMs temps d'attente, en 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.

multiaxiscontroller→registerValueCallback() YMultiAxisController
multiaxiscontroller.registerValueCallback()
multiaxiscontroller.registerValueCallback()

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.

multiaxiscontroller→reset()multiaxiscontroller.reset() YMultiAxisController multiaxiscontroller.reset()

Réinitialise tous les controlleurs et quittance toutes les alertes.

```
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.

multiaxiscontroller→set_logicalName() YMultiAxisController
multiaxiscontroller→setLogicalName()
multiaxiscontroller.set_logicalName()
multiaxiscontroller.set_logicalName()

Modifie le nom logique du contrôleur multi-axe.

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 contrôleur multi-axe.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

multiaxiscontroller→set_nAxis() YMultiAxisController
multiaxiscontroller→setNAxis()
multiaxiscontroller.set_nAxis()
multiaxiscontroller.set_nAxis()

Modifie le nombre de contrôleurs à synchroniser.

function **set_nAxis**(**newval**)

Paramètres :

newval un entier représentant le nombre de contrôleurs à synchroniser

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

multiaxiscontroller→set_userData() YMultiAxisController
multiaxiscontroller→setUserData()
multiaxiscontroller.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

multiaxiscontroller→unmuteValueCallbacks() YMultiAxisController
multiaxiscontroller.unmuteValueCallbacks()
multiaxiscontroller.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

multiaxiscontroller→wait_async() YMultiAxisController
multiaxiscontroller.wait_async()

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.

3.43. 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>
cpp	#include "yocto_network.h"
m	#import "yocto_network.h"
pas	uses yocto_network;
vb	yocto_network.vb
cs	yocto_network.cs
java	import com.yoctopuce.YoctoAPI.YNetwork;
uwp	import com.yoctopuce.YoctoAPI.YNetwork;
py	from yocto_network import *
php	require_once('yocto_network.php');
es	in HTML: <script src="../../lib/yocto_network.js"></script> in node.js: require('yoctolib-es2017/yocto_network.js');

Fonction globales

yFindNetwork(**func**)

Permet de retrouver une interface réseau d'après un identifiant donné.

yFindNetworkInContext(**yctx**, **func**)

Permet de retrouver une interface réseau d'après un identifiant donné dans un Context YAPI.

yFirstNetwork()

Commence l'énumération des interfaces réseau accessibles par la librairie.

yFirstNetworkInContext(**yctx**)

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→clearCache()

Invalide le cache.

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_callbackInitialDelay()

Retourne l'attente initiale avant la première notification par callback, en secondes.

network→get_callbackMaxDelay()

Retourne l'attente entre deux callback HTTP lorsque rien n'est à signaler, 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 callbacks HTTP, en secondes.

network→get_callbackSchedule()

Retourne la planification des callbacks HTTP, sous forme de chaîne de caractères.

network→get_callbackUrl()

Retourne l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

network→get_defaultPage()

Retourne la page HTML à envoyer pour l'URL "/"

network→get_discoverable()

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

network→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

network→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

network→get_friendlyName()

Retourne un identifiant global de l'interface réseau au format `NOM_MODULE.NOM_FONCTION`.

network→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

network→get_functionId()

Retourne l'identifiant matériel de l'interface réseau, sans référence au module.

network→get_hardwareId()

Retourne l'identifiant matériel unique de l'interface réseau au format `SERIAL.FUNCTIONID`.

network→get_httpPort()

Retourne la page HTML à envoyer pour l'URL "/"

network→get_ipAddress()

Retourne l'adresse IP utilisée par le module Yoctopuce.

network→get_ipConfig()

Retourne la configuration IP de l'interface réseau.

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_ntpServer()

Retourne l'adresse IP du serveur de NTP à utiliser pour maintenir le module à l'heure.

network→get_poeCurrent()

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

network→get_primaryDNS()

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

network→get_readiness()

Retourne l'état de fonctionnement atteint par l'interface réseau.

network→get_router()

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

network→get_secondaryDNS()

Retourne l'adresse IP du serveur de noms secondaire que le module doit utiliser.

network→get_subnetMask()

Retourne le masque de sous-réseau utilisé par le module.

network→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

network→get_userPassword()

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

network→get_wwwWatchdogDelay()

Retourne la durée de perte de connection WWW tolérée (en secondes) avant de déclancher 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→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

network→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

network→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

network→nextNetwork()

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

network→ping(**host**)

Ping l'adresse choisie 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_callbackInitialDelay(**newval**)

Modifie l'attente initiale avant la première notification par callback, en secondes.

network→set_callbackMaxDelay(**newval**)

Modifie l'attente entre deux callback HTTP lorsque rien n'est à signaler.

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 callbacks HTTP, en secondes.

network→set_callbackSchedule(newval)

Modifie la planification des callbacks HTTP, sous forme de chaîne de caractères.

network→set_callbackUrl(newval)

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

network→set_defaultPage(newval)

Modifie la page HTML par défaut du hub.

network→set_discoverable(newval)

Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

network→set_httpPort(newval)

Modifie la page HTML par défaut du hub.

network→set_logicalName(newval)

Modifie le nom logique de l'interface réseau.

network→set_ntpServer(newval)

Modifie l'adresse IP du serveur NTP que le module doit utiliser.

network→set_periodicCallbackSchedule(interval, offset)

Configure la planification de callbacks HTTP périodiques (fonction simplifiée).

network→set_primaryDNS(newval)

Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.

network→set_secondaryDNS(newval)

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

network→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

network→set_userPassword(newval)

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

network→set_wwwWatchdogDelay(newval)

Modifie la durée de perte de connection WWW tolérée (en secondes) avant de déclancher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

network→triggerCallback()

Déclenche un callback HTTP rapidement.

network→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

network→useDHCP(fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

network→useDHCPauto()

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

network→useStaticIP(ipAddress, subnetMaskLen, router)

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

network→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YNetwork.FindNetwork() YNetwork yFindNetwork()YNetwork.FindNetwork() YNetwork.FindNetwork()

Permet de retrouver une interface réseau d'après un identifiant donné.

```
function FindNetwork( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'interface réseau soit en ligne au moment ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindNetworkInContext() YNetwork
yFindNetworkInContext()
YNetwork.FindNetworkInContext()
YNetwork.FindNetworkInContext()

Permet de retrouver une interface réseau d'après un identifiant donné dans un Context YAPI.

function **FindNetworkInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'interface réseau sans ambiguïté

Retourne :

un objet de classe `YNetwork` qui permet ensuite de contrôler l'interface réseau.

YNetwork.FirstNetwork() YNetwork yFirstNetwork()YNetwork.FirstNetwork() YNetwork.FirstNetwork()

Commence l'énumération des interfaces réseau accessibles par la librairie.

function **FirstNetwork**( )

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.

YNetwork.FirstNetworkInContext() YNetwork
yFirstNetworkInContext()
YNetwork.FirstNetworkInContext()
YNetwork.FirstNetworkInContext()

Commence l'énumération des interfaces réseau accessibles par la librairie.

function **FirstNetworkInContext**(**yctx**)

Utiliser la fonction `YNetwork.nextNetwork()` pour itérer sur les autres interfaces réseau.

Paramètres :

yctx un contexte YAPI.

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→clearCache()network.clearCache() YNetwork

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'interface réseau. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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()
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()
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()
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()
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, Y_CALLBACKENCODING_YOCTO_API, Y_CALLBACKENCODING_JSON_NUM, Y_CALLBACKENCODING_EMONCMS, Y_CALLBACKENCODING_AZURE, Y_CALLBACKENCODING_INFLUXDB, Y_CALLBACKENCODING_MQTT et Y_CALLBACKENCODING_YOCTO_API_JZON 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_callbackInitialDelay() YNetwork
network→callbackInitialDelay()
network.get_callbackInitialDelay()
network.get_callbackInitialDelay()

Retourne l'attente initiale avant la première notification par callback, en secondes.

function **get_callbackInitialDelay**( )

Retourne :
un entier représentant l'attente initiale avant la première notification par callback, en secondes

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKINITIALDELAY_INVALID`.

network→get_callbackMaxDelay() YNetwork
network→callbackMaxDelay()
network.get_callbackMaxDelay()
network.get_callbackMaxDelay()

Retourne l'attente entre deux callback HTTP lorsque rien n'est à signaler, en secondes.

function **get_callbackMaxDelay**( )

Retourne :
un entier représentant l'attente entre deux callback HTTP lorsque rien n'est à signaler, en secondes

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKMAXDELAY_INVALID`.

network→get_callbackMethod() YNetwork
network→callbackMethod()
network.get_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()
network.get_callbackMinDelay()

Retourne l'attente minimale entre deux callbacks HTTP, en secondes.

function **get_callbackMinDelay**( )

Retourne :
un entier représentant l'attente minimale entre deux callbacks HTTP, en secondes

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKMINDELAY_INVALID`.

network→get_callbackSchedule() YNetwork
network→callbackSchedule()
network.get_callbackSchedule()
network.get_callbackSchedule()

Retourne la planification des callbacks HTTP, sous forme de chaîne de caractères.

function **get_callbackSchedule**( )

Retourne :

une chaîne de caractères représentant la planification des callbacks HTTP, sous forme de chaîne de caractères

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKSCHEDULE_INVALID`.

network→get_callbackUrl() YNetwork
network→callbackUrl()network.get_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_defaultPage() YNetwork
network→defaultPage()network.get_defaultPage()
network.get_defaultPage()

Retourne la page HTML à envoyer pour l'URL "/"

function **get_defaultPage**( )

Retourne :
une chaîne de caractères représentant la page HTML à envoyer pour l'URL "/"

En cas d'erreur, déclenche une exception ou retourne `Y_DEFAULTPAGE_INVALID`.

network→get_discoverable() YNetwork
network→discoverable()network.get_discoverable()
network.get_discoverable()

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

function **get_discoverable**( )

Retourne :

soit `Y_DISCOVERABLE_FALSE`, soit `Y_DISCOVERABLE_TRUE`, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour)

En cas d'erreur, déclenche une exception ou retourne `Y_DISCOVERABLE_INVALID`.

network→get_errorMessage() 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'identifant 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() YNetwork
network→functionDescriptor()
network.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

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 example `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 example 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_httpPort() YNetwork
network→httpPort()network.get_httpPort()
network.get_httpPort()

Retourne la page HTML à envoyer pour l'URL "/"

function **get_httpPort**( )

Retourne :
un entier représentant la page HTML à envoyer pour l'URL "/"

En cas d'erreur, déclenche une exception ou retourne `Y_HTTPPORT_INVALID`.

network→get_ipAddress() YNetwork
network→ipAddress()network.get_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_ipConfig() YNetwork
network→ipConfig()network.get_ipConfig()
network.get_ipConfig()

Retourne la configuration IP de l'interface réseau.

function **get_ipConfig**( )

Si l'interface réseau est configurée pour utiliser une adresse IP assignée manuellement (adresse IP statique) la chaîne commence par "STATIC:" et est suivie pas l'adresse IP, la longueur du masque de sous-réseau et l'adresse IP de la passerelle. Ces trois paramètres sont séparés par le caractère "/". Par exemple: "STATIC:192.168.1.14/16/192.168.1.1"

Si l'interface réseau est configurée pour utiliser une adresse assignée automatiquement par DHCP la chaîne commence par "DHCP:" et est suivie d'une adresse IP, d'une longueur du masque de sous-réseau et d'une adresse IP de passerelle. Ces trois paramètres sont séparés par le caractère "/" et sont utilisés si aucun serveur DHCP ne répond.

Retourne :
une chaîne de caractères représentant la configuration IP de l'interface réseau

En cas d'erreur, déclenche une exception ou retourne `Y_IPCONFIG_INVALID`.

network→get_logicalName() YNetwork
network→logicalName()network.get_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()
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_ntpServer() YNetwork
network→ntpServer()network.get_ntpServer()
network.get_ntpServer()

Retourne l'adresse IP du serveur de NTP à utiliser pour maintenir le module à l'heure.

function **get_ntpServer**( )

Retourne :
une chaîne de caractères représentant l'adresse IP du serveur de NTP à utiliser pour maintenir le module à l'heure

En cas d'erreur, déclenche une exception ou retourne `Y_NTPSERVER_INVALID`.

network→get_poeCurrent() YNetwork
network→poeCurrent()network.get_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()
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()
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 cable réseau, soit le point d'accès sans fil choisi n'est pas détecté. Le niveau 1 (LIVE_1) est atteint lorsque le réseau est détecté, mais n'est pas encore connecté. Pour un réseau sans fil, cela confirme la l'existence du SSID configuré. Le niveau 2 (LINK_2) est atteint lorsque le support matériel du réseau est fonctionnel. Pour une connection réseau filaire, le niveau 2 signifie que le cable est connecté aux deux bouts. Pour une connection à un point d'accès réseau sans fil, il démontre que les paramètres de sécurités configurés sont corrects. Pour une connection sans fil en mode ad-hoc, cela signifie qu'il y a au moins un partenaire sur le réseau ad-hoc. Le niveau 3 (DHCP_3) est atteint lorsque qu'une adresse IP a été obtenue par DHCP. Le niveau 4 (DNS_4) est atteint lorsqu'un serveur DNS est joignable par le réseau. Le niveau 5 (WWW_5) est atteint lorsque la connectivité globale à internet est avérée par l'obtention de l'heure courante sur une serveur NTP.

Retourne :

une valeur parmi `Y_READINESS_DOWN`, `Y_READINESS_EXISTS`, `Y_READINESS_LINKED`, `Y_READINESS_LAN_OK` et `Y_READINESS_WWW_OK` représentant l'état de fonctionnement atteint par l'interface réseau

En cas d'erreur, déclenche une exception ou retourne `Y_READINESS_INVALID`.

network→get_router() YNetwork
network→router()network.get_router()
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() YNetwork
network→secondaryDNS()
network.get_secondaryDNS()
network.get_secondaryDNS()

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()
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()
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()
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→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 occasionellement 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→loadAttribute()network.loadAttribute() YNetwork network.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

network→muteValueCallbacks() YNetwork
network.muteValueCallbacks()
network.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→nextNetwork()network.nextNetwork() YNetwork network.nextNetwork()

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()network.ping() YNetwork

Ping l'adresse choisie pour vérifier la connexion réseau.

```
function ping( host)
```

Envoie quatre requêtes ICMP ECHO_REQUEST à la cible 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() YNetwork
network.registerValueCallback()
network.registerValueCallback()

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() YNetwork
network→setAdminPassword()
network.set_adminPassword()
network.set_adminPassword()

Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.

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() YNetwork
network→setCallbackCredentials()
network.set_callbackCredentials()
network.set_callbackCredentials()

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() YNetwork
network→setCallbackEncoding()
network.set_callbackEncoding()
network.set_callbackEncoding()

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, Y_CALLBACKENCODING_YOCTO_API, Y_CALLBACKENCODING_JSON_NUM, Y_CALLBACKENCODING_EMONCMS, Y_CALLBACKENCODING_AZURE, Y_CALLBACKENCODING_INFLUXDB, Y_CALLBACKENCODING_MQTT et Y_CALLBACKENCODING_YOCTO_API_JZON 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_callbackInitialDelay() YNetwork
network→setCallbackInitialDelay()
network.set_callbackInitialDelay()
network.set_callbackInitialDelay()

Modifie l'attente initiale avant la première notification par callback, en secondes.

function **set_callbackInitialDelay**(**newval**)

Paramètres :

newval un entier représentant l'attente initiale avant la première notification 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_callbackMaxDelay() YNetwork
network→setCallbackMaxDelay()
network.set_callbackMaxDelay()
network.set_callbackMaxDelay()

Modifie l'attente entre deux callback HTTP lorsque rien n'est à signaler.

function **set_callbackMaxDelay**(**newval**)

Paramètres :

newval un entier représentant l'attente entre deux callback HTTP lorsque rien n'est à signaler

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()
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()
network.set_callbackMinDelay()

Modifie l'attente minimale entre deux callbacks HTTP, en secondes.

function **set_callbackMinDelay**(**newval**)

Paramètres :

newval un entier représentant l'attente minimale entre deux callbacks HTTP, 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_callbackSchedule() YNetwork
network→setCallbackSchedule()
network.set_callbackSchedule()
network.set_callbackSchedule()

Modifie la planification des callbacks HTTP, sous forme de chaîne de caractères.

function **set_callbackSchedule**(**newval**)

Paramètres :

newval une chaîne de caractères représentant la planification des callbacks HTTP, 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.

network→set_callbackUrl() YNetwork
network→setCallbackUrl()network.set_callbackUrl()
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_defaultPage() YNetwork
network→setDefaultPage()network.set_defaultPage()
network.set_defaultPage()

Modifie la page HTML par défaut du hub.

```
function set_defaultPage( newval)
```

Si aucune valeur n'est attribuée le hub retourne index.html qui est l'interface web du hub. Il est possible de changer cet page pour un fichier qui a été uploadé sur le hub.

Paramètres :

newval une chaîne de caractères représentant la page HTML par défaut du 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.

network→set_discoverable() YNetwork
network→setDiscoverable()
network.set_discoverable()
network.set_discoverable()

Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

function **set_discoverable**(**newval**)

Paramètres :

newval soit `Y_DISCOVERABLE_FALSE`, soit `Y_DISCOVERABLE_TRUE`, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour)

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_httpPort() YNetwork
network→setHttpPort()network.set_httpPort()
network.set_httpPort()

Modifie la page HTML par défaut du hub.

function **set_httpPort**(**newval**)

Si aucune valeur n'est attribuée le hub retourne index.html qui est l'interface web du hub. Il est possible de changer cet page pour un fichier qui a été uploadé sur le hub.

Paramètres :

newval un entier représentant la page HTML par défaut du 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.

network→set_logicalName() YNetwork
network→setLogicalName()
network.set_logicalName()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_ntpServer() YNetwork
network→setNtpServer()network.set_ntpServer()
network.set_ntpServer()

Modifie l'adresse IP du serveur NTP que le module doit utiliser.

function **set_ntpServer**(**newval**)

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 NTP 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_periodicCallbackSchedule() YNetwork
network→setPeriodicCallbackSchedule()
network.set_periodicCallbackSchedule()
network.set_periodicCallbackSchedule()

Configure la planification de callbacks HTTP périodiques (fonction simplifiée).

function **set_periodicCallbackSchedule**(**interval**, **offset**)

Paramètres :

interval la périodicité du callback sous forme textuelle, exprimée en secondes, minutes ou en heures. Par exemple: "60s", "5m", "1h", "48h".

offset un entier décrivant le décalage du callback par rapport au début de la période. Par exemple, si la périodicité est 24h, un offset de 7 déclanchera le callback chaque jour à 7h du matin.

Retourne :

YAPI_SUCCESS when the call succeeds.

On failure, throws an exception or returns a negative error code.

network→set_primaryDNS() YNetwork
network→setPrimaryDNS()network.set_primaryDNS()
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()
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'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

network→set_userPassword() YNetwork
network→setUserPassword()
network.set_userPassword()
network.set_userPassword()

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()
network.set_wwwWatchdogDelay()

Modifie la durée de perte de connection WWW tolérée (en secondes) avant de déclancher 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 déclancher 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→triggerCallback()network.triggerCallback() YNetwork network.triggerCallback()

Déclenche un callback HTTP rapidement.

function **triggerCallback**( )

Cette fonction peut même être appellée à l'intérieur d'un callback HTTP, dans quel cas le callback HTTP suivant sera appelé 5 secondes après la fin du callback courant, indépendamment de l'intervalle minimal configuré dans le module.

Retourne :
une chaîne de caractères contenant le résultat du ping.

network→unmuteValueCallbacks() YNetwork
network.unmuteValueCallbacks()
network.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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 network.useDHCP()

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→useDHCPauto()network.useDHCPauto() YNetwork network.useDHCPauto()

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

function **useDHCPauto**( )

En attendant qu'une adresse soit reçue (et indéfiniment si aucun serveur DHCP ne répond), le module utilise une adresse IP du réseau 169.254.0.0/16 (APIPA). N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

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 network.useStaticIP()

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.

3.44. contrôle d'OS

L'objet OsControl permet de controler le système d'exploitation sur lequel tourne un VirtualHub. OsControl n'est disponible que dans le VirtualHub software. Attention, cette fonctionnalité doit être explicitement activé au lancement du VirtualHub, avec l'option -o.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_oscontrol.js'></script>
cpp	#include "yocto_oscontrol.h"
m	#import "yocto_oscontrol.h"
pas	uses yocto_oscontrol;
vb	yocto_oscontrol.vb
cs	yocto_oscontrol.cs
java	import com.yoctopuce.YoctoAPI.YOsControl;
uwp	import com.yoctopuce.YoctoAPI.YOsControl;
py	from yocto_oscontrol import *
php	require_once('yocto_oscontrol.php');
es	in HTML: <script src="../../lib/yocto_oscontrol.js"></script> in node.js: require('yoctolib-es2017/yocto_oscontrol.js');

Fonction globales
yFindOsControl(**func**) Permet de retrouver un contrôle d'OS d'après un identifiant donné.
yFindOsControlInContext(**yctx**, **func**) Permet de retrouver un contrôle d'OS d'après un identifiant donné dans un Context YAPI.
yFirstOsControl() Commence l'énumération des contrôle d'OS accessibles par la librairie.
yFirstOsControlInContext(**yctx**) Commence l'énumération des contrôle d'OS accessibles par la librairie.
Méthodes des objets `YOsControl`
oscontrol→clearCache() Invalide le cache.
oscontrol→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'OS au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
oscontrol→get_advertisedValue() Retourne la valeur courante du contrôle d'OS (pas plus de 6 caractères).
oscontrol→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.
oscontrol→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.
oscontrol→get_friendlyName() Retourne un identifiant global du contrôle d'OS au format `NOM_MODULE.NOM_FONCTION`.
oscontrol→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
oscontrol→get_functionId() Retourne l'identifiant matériel du contrôle d'OS, sans référence au module.
oscontrol→get_hardwareId()

Retourne l'identifiant matériel unique du contrôle d'OS au format `SERIAL.FUNCTIONID`.

oscontrol→get_logicalName()

Retourne le nom logique du contrôle d'OS.

oscontrol→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

oscontrol→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

oscontrol→get_shutdownCountdown()

Retourne le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé.

oscontrol→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

oscontrol→isOnline()

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

oscontrol→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

oscontrol→load(**msValidity**)

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

oscontrol→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

oscontrol→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

oscontrol→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

oscontrol→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YOsControl.FindOsControl() YOsControl yFindOsControl()YOsControl.FindOsControl() YOsControl.FindOsControl()

Permet de retrouver un contrôle d'OS d'après un identifiant donné.

```
function FindOsControl( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindOsControlInContext() YOsControl
yFindOsControlInContext()
YOsControl.FindOsControlInContext()
YOsControl.FindOsControlInContext()

Permet de retrouver un contrôle d'OS d'après un identifiant donné dans un Context YAPI.

function **FindOsControlInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le contrôle d'OS sans ambiguïté

Retourne :

un objet de classe `YOsControl` qui permet ensuite de contrôler le contrôle d'OS.

YOsControl.FirstOsControl() YOsControl yFirstOsControl()YOsControl.FirstOsControl() YOsControl.FirstOsControl()

Commence l'énumération des contrôle d'OS accessibles par la librairie.

function **FirstOsControl**( )

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.

YOsControl.FirstOsControlInContext() YOsControl
yFirstOsControlInContext()
YOsControl.FirstOsControlInContext()
YOsControl.FirstOsControlInContext()

Commence l'énumération des contrôle d'OS accessibles par la librairie.

function **FirstOsControlInContext**(**yctx**)

Utiliser la fonction `YOsControl.nextOsControl()` pour itérer sur les autres contrôle d'OS.

Paramètres :

yctx un contexte YAPI.

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→clearCache()oscontrol.clearCache() YOsControl

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du contrôle d'OS. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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()
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'identifant 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() YOsControl
oscontrol→functionDescriptor()
oscontrol.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

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 example `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 example 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()
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_shutdownCountdown() YOsControl
oscontrol→shutdownCountdown()
oscontrol.get_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.

oscontrol→isOnline()oscontrol.isOnline() YOsControl

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du contrôle d'OS sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le contrôle d'OS est joignable, `false` sinon

oscontrol→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 occasionellement 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→loadAttribute()oscontrol.loadAttribute() YOsControl oscontrol.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

oscontrol→muteValueCallbacks() YOsControl
oscontrol.muteValueCallbacks()
oscontrol.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

oscontrol→nextOsControl() YOsControl
oscontrol.nextOsControl()oscontrol.nextOsControl()

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() YOsControl
oscontrol.registerValueCallback()
oscontrol.registerValueCallback()

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() YOsControl
oscontrol→setLogicalName()
oscontrol.set_logicalName()
oscontrol.set_logicalName()

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 oscontrol.shutdown()

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→unmuteValueCallbacks() YOsControl
oscontrol.unmuteValueCallbacks()
oscontrol.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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.

3.45. Interface de la fonction Power

La classe YPower permet de lire et de configurer les capteurs de puissance Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, elle d'accéder au compteur d'énergie et à l'estimation du facteur de puissance.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_power.js'></script>
cpp	#include "yocto_power.h"
m	#import "yocto_power.h"
pas	uses yocto_power;
vb	yocto_power.vb
cs	yocto_power.cs
java	import com.yoctopuce.YoctoAPI.YPower;
uwp	import com.yoctopuce.YoctoAPI.YPower;
py	from yocto_power import *
php	require_once('yocto_power.php');
es	in HTML: <script src="../../lib/yocto_power.js"></script> in node.js: require('yoctolib-es2017/yocto_power.js');

Fonction globales
yFindPower(**func**) Permet de retrouver un capteur de puissance electrique d'après un identifiant donné.
yFindPowerInContext(**yctx**, **func**) Permet de retrouver un capteur de puissance electrique d'après un identifiant donné dans un Context YAPI.
yFirstPower() Commence l'énumération des capteurs de puissance electrique accessibles par la librairie.
yFirstPowerInContext(**yctx**) Commence l'énumération des capteurs de puissance electrique 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→clearCache() Invalide le cache.
power→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de puissance electrique au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
power→get_advertisedValue() Retourne la valeur courante du capteur de puissance electrique (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), en Watt, sous forme de nombre à virgule.
power→get_currentValue() Retourne la valeur actuelle de la puissance électrique, en Watt, sous forme de nombre à virgule.
power→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

power→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance electrique.

power→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance electrique.

power→get_friendlyName()

Retourne un identifiant global du capteur de puissance electrique 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 electrique, sans référence au module.

power→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de puissance electrique au format SERIAL.FUNCTIONID.

power→get_highestValue()

Retourne la valeur maximale observée pour la puissance électrique depuis le démarrage du module.

power→get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

power→get_logicalName()

Retourne le nom logique du capteur de puissance electrique.

power→get_lowestValue()

Retourne la valeur minimale observée pour la puissance électrique depuis le démarrage du module.

power→get_meter()

Retourne la valeur actuelle du compteur d'energie, calculée par le wattmètre en intégrant la consommation instantanée.

power→get_meterTimer()

Retourne le temps écoulé depuis la dernière initilialisation 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_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

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 electrique est joignable, sans déclencher d'erreur.

power→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le capteur de puissance electrique est joignable, sans déclencher d'erreur.

power→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

power→load(**msValidity**)

Met en cache les valeurs courantes du capteur de puissance electrique, avec une durée de validité spécifiée.

power→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

power→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

power→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du capteur de puissance electrique, avec une durée de validité spécifiée.

power→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

power→nextPower()

Continue l'énumération des capteurs de puissance electrique commencée à l'aide de `yFirstPower()`.

power→registerTimedReportCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

power→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

power→reset()

Réinitialise le compteur d'énergie.

power→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

power→set_logFrequency(**newval**)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

power→set_logicalName(**newval**)

Modifie le nom logique du capteur de puissance electrique.

power→set_lowestValue(**newval**)

Modifie la mémoire de valeur minimale observée.

power→set_reportFrequency(**newval**)

Modifie la fréquence de notification périodique des valeurs mesurées.

power→set_resolution(**newval**)

Modifie la résolution des valeurs physique mesurées.

power→set_userData(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

power→startDataLogger()

Démarre l'enregistreur de données du module.

power→stopDataLogger()

Arrête l'enregistreur de données du module.

power→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

power→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YPower.FindPower() YPower
yFindPower()YPower.FindPower()
YPower.FindPower()

Permet de retrouver un capteur de puissance electrique d'après un identifiant donné.

```
function FindPower( 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 electrique soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPower.isOnline()` pour tester si le capteur de puissance electrique 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le capteur de puissance electrique sans ambiguïté

Retourne :

un objet de classe `YPower` qui permet ensuite de contrôler le capteur de puissance electrique.

YPower.FindPowerInContext() YPower yFindPowerInContext()YPower.FindPowerInContext() YPower.FindPowerInContext()

Permet de retrouver un capteur de puissance electrique d'après un identifiant donné dans un Context YAPI.

function **FindPowerInContext**(**yctx**, **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 electrique soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPower.isOnline()` pour tester si le capteur de puissance electrique 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de puissance electrique sans ambiguïté

Retourne :

un objet de classe `YPower` qui permet ensuite de contrôler le capteur de puissance electrique.

YPower.FirstPower() YPower yFirstPower()YPower.FirstPower() YPower.FirstPower()

Commence l'énumération des capteurs de puissance electrique accessibles par la librairie.

function **FirstPower**( )

Utiliser la fonction `YPower.nextPower()` pour itérer sur les autres capteurs de puissance electrique.

Retourne :

un pointeur sur un objet `YPower`, correspondant au premier capteur de puissance electrique accessible en ligne, ou `null` si il n'y a pas de capteurs de puissance electrique disponibles.

YPower.FirstPowerInContext() YPower
yFirstPowerInContext()
YPower.FirstPowerInContext()
YPower.FirstPowerInContext()

Commence l'énumération des capteurs de puissance electrique accessibles par la librairie.

```
function FirstPowerInContext( yctx)
```

Utiliser la fonction `YPower.nextPower()` pour itérer sur les autres capteurs de puissance electrique.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YPower`, correspondant au premier capteur de puissance electrique accessible en ligne, ou `null` si il n'y a pas de capteurs de puissance electrique disponibles.

power→calibrateFromPoints() YPower
power.calibrateFromPoints()
power.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→clearCache()power.clearCache() YPower

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur de puissance electrique. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

power→describe()power.describe() YPower

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de puissance electrique au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le capteur de puissance electrique (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

power→get_advertisedValue() YPower
power→advertisedValue()
power.get_advertisedValue()
power.get_advertisedValue()

Retourne la valeur courante du capteur de puissance electrique (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de puissance electrique (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()
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()
power.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Watt, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en Watt, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

power→get_currentValue() YPower
power→currentValue()power.get_currentValue()
power.get_currentValue()

Retourne la valeur actuelle de la puissance électrique, en Watt, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle de la puissance électrique, en Watt, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

power→get_dataLogger() YPower
power→dataLogger()power.get_dataLogger()
power.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

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 electrique.

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 electrique.

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 electrique.

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 electrique.

power→get_friendlyName() YPower
power→friendlyName()power.get_friendlyName()

Retourne un identifiant global du capteur de puissance electrique au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du capteur de puissance electrique si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du capteur de puissance electrique (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le capteur de puissance electrique 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 electrique, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de puissance electrique (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 electrique 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 electrique (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le capteur de puissance electrique (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()
power.get_highestValue()

Retourne la valeur maximale observée pour la puissance électrique depuis le démarrage du module.

function **get_highestValue**( )

Retourne :
une valeur numérique représentant la valeur maximale observée pour la puissance électrique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

power→get_logFrequency() YPower
power→logFrequency()power.get_logFrequency()
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()
power.get_logicalName()

Retourne le nom logique du capteur de puissance electrique.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du capteur de puissance electrique.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

power→get_lowestValue() YPower
power→lowestValue()power.get_lowestValue()
power.get_lowestValue()

Retourne la valeur minimale observée pour la puissance électrique depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :
une valeur numérique représentant la valeur minimale observée pour la puissance électrique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

power→get_meter() YPower
power→meter()power.get_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()
power.get_meterTimer()

Retourne le temps écoulé depuis la dernière initilialisation du compteur d'énergie, en secondes

function **get_meterTimer**( )

Retourne :
un entier représentant le temps écoulé depuis la dernière initilialisation 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_recordedData() YPower
power→recordedData()power.get_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**)

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 find de l'intercalle 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()
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()
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_sensorState() YPower
power→sensorState()power.get_sensorState()
power.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :
un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

power→get_unit() YPower
power→unit()power.get_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 electrique est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du capteur de puissance electrique sont valides au moment de l'appel, le module est considéré joignable. 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 electrique est joignable, `false` sinon

power→load()power.load() YPower

Met en cache les valeurs courantes du capteur de puissance electrique, 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 occasionellement 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→loadAttribute()power.loadAttribute() YPower power.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

power→loadCalibrationPoints() YPower
power.loadCalibrationPoints()
power.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→muteValueCallbacks() YPower
power.muteValueCallbacks()
power.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→nextPower()power.nextPower() YPower power.nextPower()

Continue l'énumération des capteurs de puissance electrique 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() YPower
power.registerTimedReportCallback()
power.registerTimedReportCallback()

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.

power→registerValueCallback() YPower
power.registerValueCallback()
power.registerValueCallback()

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()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()
power.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.

power→set_logFrequency() YPower
power→setLogFrequency()power.set_logFrequency()
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()
power.set_logicalName()

Modifie le nom logique du capteur de puissance electrique.

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 electrique.

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()
power.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.

power→set_reportFrequency() YPower
power→setReportFrequency()
power.set_reportFrequency()
power.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.

power→set_resolution() YPower
power→setResolution()power.set_resolution()
power.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.

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→startDataLogger()power.startDataLogger() YPower power.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

power→stopDataLogger()power.stopDataLogger() YPower power.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

power→unmuteValueCallbacks() YPower
power.unmuteValueCallbacks()
power.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.46. Interface d'alimentation de sortie

La librairie de programmation Yoctopuce permet de contrôler l'alimentation mise a disposition sur certain module tels que le Yocto-Serial

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_poweroutput.js'></script>
cpp     #include "yocto_poweroutput.h"
m       #import "yocto_poweroutput.h"
pas     uses yocto_poweroutput;
vb      yocto_poweroutput.vb
cs      yocto_poweroutput.cs
java    import com.yoctopuce.YoctoAPI.YPowerOutput;
uwp     import com.yoctopuce.YoctoAPI.YPowerOutput;
py      from yocto_poweroutput import *
php     require_once('yocto_poweroutput.php');
es      in HTML: <script src="../../lib/yocto_poweroutput.js"></script>
        in node.js: require('yoctolib-es2017/yocto_poweroutput.js');
```

Fonction globales
yFindPowerOutput(**func**) Permet de retrouver une alimentation d'après un identifiant donné.
yFindPowerOutputInContext(**yctx**, **func**) Permet de retrouver une alimentation d'après un identifiant donné dans un Context YAPI.
yFirstPowerOutput() Commence l'énumération des alimentation accessibles par la librairie.
yFirstPowerOutputInContext(**yctx**) Commence l'énumération des alimentation accessibles par la librairie.
Méthodes des objets `YPowerOutput`
poweroutput→clearCache() Invalide le cache.
poweroutput→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de l'alimentation au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
poweroutput→get_advertisedValue() Retourne la valeur courante de l'alimentation (pas plus de 6 caractères).
poweroutput→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'alimentation.
poweroutput→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'alimentation.
poweroutput→get_friendlyName() Retourne un identifiant global de l'alimentation au format `NOM_MODULE.NOM_FONCTION`.
poweroutput→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
poweroutput→get_functionId() Retourne l'identifiant matériel de l'alimentation, sans référence au module.
poweroutput→get_hardwareId() Retourne l'identifiant matériel unique de l'alimentation au format `SERIAL.FUNCTIONID`.

poweroutput→get_logicalName()

Retourne le nom logique de l'alimentation.

poweroutput→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

poweroutput→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

poweroutput→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

poweroutput→get_voltage()

Retourne le voltage envoyé sur l'alimentation mise à disposition sur le module.

poweroutput→isOnline()

Vérifie si le module hébergeant l'alimentation est joignable, sans déclencher d'erreur.

poweroutput→isOnline_async(callback, context)

Vérifie si le module hébergeant l'alimentation est joignable, sans déclencher d'erreur.

poweroutput→load(msValidity)

Met en cache les valeurs courantes de l'alimentation, avec une durée de validité spécifiée.

poweroutput→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

poweroutput→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'alimentation, avec une durée de validité spécifiée.

poweroutput→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

poweroutput→nextPowerOutput()

Continue l'énumération des alimentation commencée à l'aide de `yFirstPowerOutput()`.

poweroutput→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

poweroutput→set_logicalName(newval)

Modifie le nom logique de l'alimentation.

poweroutput→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`.

poweroutput→set_voltage(newval)

Modifie le voltage de l'alimentation mise à disposition par le module.

poweroutput→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

poweroutput→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.

YPowerOutput.FindPowerOutput() YPowerOutput
yFindPowerOutput()
YPowerOutput.FindPowerOutput()
YPowerOutput.FindPowerOutput()

Permet de retrouver une alimentation d'après un identifiant donné.

```
function FindPowerOutput( 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'alimentation soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPowerOutput.isOnline()` pour tester si l'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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence l'alimentation sans ambiguïté

Retourne :

un objet de classe `YPowerOutput` qui permet ensuite de contrôler l'alimentation.

YPowerOutput.FindPowerOutputInContext() YPowerOutput yFindPowerOutputInContext() YPowerOutput.FindPowerOutputInContext() YPowerOutput.FindPowerOutputInContext()

Permet de retrouver une alimentation d'après un identifiant donné dans un Context YAPI.

function **FindPowerOutputInContext**(**yctx**, **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'alimentation soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPowerOutput.isOnline()` pour tester si l'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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'alimentation sans ambiguïté

Retourne :

un objet de classe `YPowerOutput` qui permet ensuite de contrôler l'alimentation.

YPowerOutput.FirstPowerOutput() YPowerOutput
yFirstPowerOutput()
YPowerOutput.FirstPowerOutput()
YPowerOutput.FirstPowerOutput()

Commence l'énumération des alimentation accessibles par la librairie.

function **FirstPowerOutput**( )

Utiliser la fonction `YPowerOutput.nextPowerOutput()` pour itérer sur les autres alimentation.

Retourne :

un pointeur sur un objet `YPowerOutput`, correspondant à la premiere alimentation accessible en ligne, ou `null` si il n'y a pas de alimentation disponibles.

YPowerOutput.FirstPowerOutputInContext() YPowerOutput yFirstPowerOutputInContext() YPowerOutput.FirstPowerOutputInContext() YPowerOutput.FirstPowerOutputInContext()

Commence l'énumération des alimentation accessibles par la librairie.

function **FirstPowerOutputInContext**(**yctx**)

Utiliser la fonction `YPowerOutput.nextPowerOutput()` pour itérer sur les autres alimentation.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YPowerOutput`, correspondant à la premiere alimentation accessible en ligne, ou `null` si il n'y a pas de alimentation disponibles.

poweroutput→clearCache()poweroutput.clearCache() YPowerOutput

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'alimentation. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

poweroutput→describe()poweroutput.describe() YPowerOutput

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'alimentation au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant l'alimentation (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

poweroutput→get_advertisedValue() YPowerOutput
poweroutput→advertisedValue()
poweroutput.get_advertisedValue()
poweroutput.get_advertisedValue()

Retourne la valeur courante de l'alimentation (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante de l'alimentation (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

poweroutput→get_errorMessage() YPowerOutput
poweroutput→errorMessage()
poweroutput.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'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 de l'alimentation.

poweroutput→get_errorType() YPowerOutput
poweroutput→errorType()
poweroutput.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'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 de l'alimentation.

poweroutput→get_friendlyName() YPowerOutput
poweroutput→friendlyName()
poweroutput.get_friendlyName()

Retourne un identifiant global de l'alimentation au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et de l'alimentation si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel de l'alimentation (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant l'alimentation en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

poweroutput→get_functionDescriptor() YPowerOutput
poweroutput→functionDescriptor()
poweroutput.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

poweroutput→get_functionId() YPowerOutput
poweroutput→functionId()
poweroutput.get_functionId()

Retourne l'identifiant matériel de l'alimentation, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant l'alimentation (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

poweroutput→get_hardwareId() YPowerOutput
poweroutput→hardwareId()
poweroutput.get_hardwareId()

Retourne l'identifiant matériel unique de l'alimentation 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'alimentation (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant l'alimentation (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

poweroutput→get_logicalName() YPowerOutput
poweroutput→logicalName()
poweroutput.get_logicalName()
poweroutput.get_logicalName()

Retourne le nom logique de l'alimentation.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique de l'alimentation.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

poweroutput→get_module() YPowerOutput
poweroutput→module()poweroutput.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

poweroutput→get_userData() YPowerOutput
poweroutput→userData()poweroutput.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.

poweroutput→get_voltage() YPowerOutput
poweroutput→voltage()poweroutput.get_voltage()
poweroutput.get_voltage()

Retourne le voltage envoyé sur l'alimentation mise à disposition sur le module.

function **get_voltage**( )

Retourne :

une valeur parmi Y_VOLTAGE_OFF, Y_VOLTAGE_OUT3V3 et Y_VOLTAGE_OUT5V représentant le voltage envoyé sur l'alimentation mise à disposition sur le module

En cas d'erreur, déclenche une exception ou retourne Y_VOLTAGE_INVALID.

poweroutput→isOnline()poweroutput.isOnline() YPowerOutput

Vérifie si le module hébergeant l'alimentation est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache de l'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 l'alimentation est joignable, `false` sinon

poweroutput→load()poweroutput.load() YPowerOutput

Met en cache les valeurs courantes de l'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 occasionellement 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.

poweroutput→loadAttribute() YPowerOutput
poweroutput.loadAttribute()
poweroutput.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

poweroutput→muteValueCallbacks() YPowerOutput
poweroutput.muteValueCallbacks()
poweroutput.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

poweroutput→nextPowerOutput() YPowerOutput
poweroutput.nextPowerOutput()
poweroutput.nextPowerOutput()

Continue l'énumération des alimentation commencée à l'aide de `yFirstPowerOutput()`.

function **nextPowerOutput**( )

Retourne :

un pointeur sur un objet `YPowerOutput` accessible en ligne, ou `null` lorsque l'énumération est terminée.

poweroutput→registerValueCallback() YPowerOutput
poweroutput.registerValueCallback()
poweroutput.registerValueCallback()

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.

poweroutput→set_logicalName() YPowerOutput
poweroutput→setLogicalName()
poweroutput.set_logicalName()
poweroutput.set_logicalName()

Modifie le nom logique de l'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 de l'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.

poweroutput→set_userData() YPowerOutput
poweroutput→setUserData()
poweroutput.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

poweroutput→set_voltage() YPowerOutput
poweroutput→setVoltage()poweroutput.set_voltage()
poweroutput.set_voltage()

Modifie le voltage de l'alimentation mise à disposition par le module.

function **set_voltage**(**newval**)

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur parmi `Y_VOLTAGE_OFF`, `Y_VOLTAGE_OUT3V3` et `Y_VOLTAGE_OUT5V` représentant le voltage de l'alimentation mise à disposition 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.

poweroutput→unmuteValueCallbacks() YPowerOutput
poweroutput.unmuteValueCallbacks()
poweroutput.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

poweroutput→wait_async()poweroutput.wait_async() YPowerOutput

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.

3.47. Interface de la fonction Pressure

La classe YPressure permet de lire et de configurer les capteurs de pression Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_pressure.js'></script>
cpp     #include "yocto_pressure.h"
m       #import "yocto_pressure.h"
pas     uses yocto_pressure;
vb      yocto_pressure.vb
cs      yocto_pressure.cs
java    import com.yoctopuce.YoctoAPI.YPressure;
uwp     import com.yoctopuce.YoctoAPI.YPressure;
py      from yocto_pressure import *
php     require_once('yocto_pressure.php');
es      in HTML: <script src="../../lib/yocto_pressure.js"></script>
        in node.js: require('yoctolib-es2017/yocto_pressure.js');
```

Fonction globales
yFindPressure(**func**) Permet de retrouver un capteur de pression d'après un identifiant donné.
yFindPressureInContext(**yctx**, **func**) Permet de retrouver un capteur de pression d'après un identifiant donné dans un Context YAPI.
yFirstPressure() Commence l'énumération des capteurs de pression accessibles par la librairie.
yFirstPressureInContext(**yctx**) 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→clearCache() Invalide le cache.
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), en millibar (hPa), sous forme de nombre à virgule.
pressure→get_currentValue() Retourne la valeur actuelle de la pression, en millibar (hPa), sous forme de nombre à virgule.
pressure→get_dataLogger() Retourne l'objet YDataLogger du module qui héberge le senseur.
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 depuis le démarrage du module.

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 depuis le démarrage du module.

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_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

pressure→load(**msValidity**)

Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.

pressure→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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.

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.

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→startDataLogger()

Démarre l'enregistreur de données du module.

pressure→stopDataLogger()

Arrête l'enregistreur de données du module.

pressure→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

pressure→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YPressure.FindPressure() YPressure yFindPressure()YPressure.FindPressure() YPressure.FindPressure()

Permet de retrouver un capteur de pression d'après un identifiant donné.

```
function FindPressure( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindPressureInContext() YPressure
yFindPressureInContext()
YPressure.FindPressureInContext()
YPressure.FindPressureInContext()

Permet de retrouver un capteur de pression d'après un identifiant donné dans un Context YAPI.

function **FindPressureInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de pression sans ambiguïté

Retourne :

un objet de classe `YPressure` qui permet ensuite de contrôler le capteur de pression.

YPressure.FirstPressure() YPressure yFirstPressure()YPressure.FirstPressure() YPressure.FirstPressure()

Commence l'énumération des capteurs de pression accessibles par la librairie.

function **FirstPressure**( )

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.

YPressure.FirstPressureInContext() YPressure
yFirstPressureInContext()
YPressure.FirstPressureInContext()
YPressure.FirstPressureInContext()

Commence l'énumération des capteurs de pression accessibles par la librairie.

function **FirstPressureInContext**(**yctx**)

Utiliser la fonction `YPressure.nextPressure()` pour itérer sur les autres capteurs de pression.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YPressure`, correspondant au premier capteur de pression accessible en ligne, ou `null` si il n'y a pas de capteurs de pression disponibles.

pressure→calibrateFromPoints() YPressure
pressure.calibrateFromPoints()
pressure.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→clearCache()pressure.clearCache() YPressure

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur de pression. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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()
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()
pressure.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en millibar (hPa), sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en millibar (hPa), sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

pressure→get_currentValue() YPressure
pressure→currentValue()pressure.get_currentValue()
pressure.get_currentValue()

Retourne la valeur actuelle de la pression, en millibar (hPa), sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle de la pression, en millibar (hPa), sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

pressure→get_dataLogger() YPressure
pressure→dataLogger()pressure.get_dataLogger()
pressure.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

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'identifant 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 example `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 example 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()
pressure.get_highestValue()

Retourne la valeur maximale observée pour la pression depuis le démarrage du module.

function **get_highestValue**( )

Retourne :

une valeur numérique représentant la valeur maximale observée pour la pression depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

pressure→get_logFrequency() YPressure
pressure→logFrequency()
pressure.get_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()
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()
pressure.get_lowestValue()

Retourne la valeur minimale observée pour la pression depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :
une valeur numérique représentant la valeur minimale observée pour la pression depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

pressure→get_module() YPressure
pressure→module()pressure.get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

function **get_module**( )

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :
une instance de YModule

pressure→get_recordedData() YPressure
pressure→recordedData()
pressure.get_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**)

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 find de l'intercalle 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()
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()
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_sensorState() YPressure
pressure→sensorState()pressure.get_sensorState()
pressure.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :

un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

pressure→get_unit() YPressure
pressure→unit()pressure.get_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→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 occasionellement 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→loadAttribute()pressure.loadAttribute() YPressure pressure.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

pressure→loadCalibrationPoints() YPressure
pressure.loadCalibrationPoints()
pressure.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→muteValueCallbacks() YPressure
pressure.muteValueCallbacks()
pressure.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→nextPressure()pressure.nextPressure() YPressure pressure.nextPressure()

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() YPressure
pressure.registerTimedReportCallback()
pressure.registerTimedReportCallback()

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() YPressure
pressure.registerValueCallback()
pressure.registerValueCallback()

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()
pressure.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.

pressure→set_logFrequency() YPressure
pressure→setLogFrequency()
pressure.set_logFrequency()
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()
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() YPressure
pressure→setLowestValue()
pressure.set_lowestValue()
pressure.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.

pressure→set_reportFrequency() YPressure
pressure→setReportFrequency()
pressure.set_reportFrequency()
pressure.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.

pressure→set_resolution() YPressure
pressure→setResolution()pressure.set_resolution()
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'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

pressure→startDataLogger() YPressure
pressure.startDataLogger()
pressure.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

pressure→stopDataLogger() YPressure
pressure.stopDataLogger()
pressure.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

pressure→unmuteValueCallbacks() YPressure
pressure.unmuteValueCallbacks()
pressure.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→wait_async()pressure.wait_async() YPressure

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

function **wait_async**(**callback**, **context**)

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout.

3.48. Interface de la fonction Proximity

La classe YProximity permet d'utiliser et de configurer les capteurs de proximité Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, elle permet d'effectuer facilement une calibration linéaire à un point pour compenser l'effet d'une vitre ou d'un filtre placé devant le capteur.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_proximity.js'></script>
cpp	#include "yocto_proximity.h"
m	#import "yocto_proximity.h"
pas	uses yocto_proximity;
vb	yocto_proximity.vb
cs	yocto_proximity.cs
java	import com.yoctopuce.YoctoAPI.YProximity;
uwp	import com.yoctopuce.YoctoAPI.YProximity;
py	from yocto_proximity import *
php	require_once('yocto_proximity.php');
es	in HTML: <script src="../../lib/yocto_proximity.js"></script> in node.js: require('yoctolib-es2017/yocto_proximity.js');

Fonction globales
yFindProximity(**func**) Permet de retrouver un capteur de proximité d'après un identifiant donné.
yFindProximityInContext(**yctx**, **func**) Permet de retrouver un capteur de proximité d'après un identifiant donné dans un Context YAPI.
yFirstProximity() Commence l'énumération des capteurs de proximité accessibles par la librairie.
yFirstProximityInContext(**yctx**) Commence l'énumération des capteurs de proximité accessibles par la librairie.
Méthodes des objets `YProximity`
proximity→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.
proximity→clearCache() Invalide le cache.
proximity→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de proximité au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
proximity→get_advertisedValue() Retourne la valeur courante du capteur de proximité (pas plus de 6 caractères).
proximity→get_currentRawValue() Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule.
proximity→get_currentValue() Retourne la valeur actuelle de la détection de proximité, en l'unité spécifiée, sous forme de nombre à virgule.
proximity→get_dataLogger() Retourne l'objet YDataLogger du module qui héberge le senseur.
proximity→get_detectionThreshold()

Retourne le seuil utilisé pour déterminer l'état logique de la détection de proximité, lorsqu'on la traite comme une entrée binaire (on/off).

proximity→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de proximité.

proximity→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de proximité.

proximity→get_friendlyName()

Retourne un identifiant global du capteur de proximité au format `NOM_MODULE.NOM_FONCTION`.

proximity→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

proximity→get_functionId()

Retourne l'identifiant matériel du capteur de proximité, sans référence au module.

proximity→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de proximité au format `SERIAL.FUNCTIONID`.

proximity→get_highestValue()

Retourne la valeur maximale observée pour la détection de proximité depuis le démarrage du module.

proximity→get_isPresent()

Retourne vrai si l'entrée (considérée comme binaire) est active (valeur de détection inférieure au seuil `threshold`), et faux sinon.

proximity→get_lastTimeApproached()

Retourne le temps absolu (nombre de millisecondes) entre la mise sous tension du module et la dernière détection observée (transition de absent à présent).

proximity→get_lastTimeRemoved()

Retourne le temps absolu (nombre de millisecondes) entre la mise sous tension du module et la dernière détection observée (transition de présent à absent).

proximity→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.

proximity→get_logicalName()

Retourne le nom logique du capteur de proximité.

proximity→get_lowestValue()

Retourne la valeur minimale observée pour la détection de proximité depuis le démarrage du module.

proximity→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

proximity→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

proximity→get_proximityReportMode()

Retourne le type de paramètre (valeur du capteur, présence ou compteur d'impulsion) renvoyé par la fonction get_currentValue et les callback.

proximity→get_pulseCounter()

Retourne la valeur du compteur d'impulsions.

proximity→get_pulseTimer()

Retourne le timer du compteur d'impulsions (ms).

proximity→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.

proximity→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

proximity→get_resolution()

Retourne la résolution des valeurs mesurées.

proximity→get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

proximity→get_signalValue()

Retourne la valeur actuelle du mesuré par le capteur de proximité.

proximity→get_unit()

Retourne l'unité dans laquelle la détection de proximité est exprimée.

proximity→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

proximity→isOnline()

Vérifie si le module hébergeant le capteur de proximité est joignable, sans déclencher d'erreur.

proximity→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le capteur de proximité est joignable, sans déclencher d'erreur.

proximity→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

proximity→load(**msValidity**)

Met en cache les valeurs courantes du capteur de proximité, avec une durée de validité spécifiée.

proximity→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

proximity→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

proximity→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du capteur de proximité, avec une durée de validité spécifiée.

proximity→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

proximity→nextProximity()

Continue l'énumération des capteurs de proximité commencée à l'aide de `yFirstProximity()`.

proximity→registerTimedReportCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

proximity→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

proximity→resetCounter()

Réinitialise le compteur d'impulsions et son timer.

proximity→set_detectionThreshold(**newval**)

Modifie le seuil utilisé pour déterminer l'état logique de la détection de proximité, lorsqu'on la traite comme une entrée binaire (on/off).

proximity→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

proximity→set_logFrequency(**newval**)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

proximity→set_logicalName(newval)

Modifie le nom logique du capteur de proximité.

proximity→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

proximity→set_proximityReportMode(newval)

Change le type de paramètre (valeur du capteur, présence ou compteur d'impulsion) renvoyé par la fonction get_currentValue et les callback.

proximity→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

proximity→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

proximity→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`.

proximity→startDataLogger()

Démarre l'enregistreur de données du module.

proximity→stopDataLogger()

Arrête l'enregistreur de données du module.

proximity→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

proximity→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.

YProximity.FindProximity() YProximity yFindProximity()YProximity.FindProximity() YProximity.FindProximity()

Permet de retrouver un capteur de proximité d'après un identifiant donné.

```
function FindProximity( 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 proximité soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YProximity.isOnline()` pour tester si le capteur de proximité 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le capteur de proximité sans ambiguïté

Retourne :

un objet de classe `YProximity` qui permet ensuite de contrôler le capteur de proximité.

YProximity.FindProximityInContext() YProximity
yFindProximityInContext()
YProximity.FindProximityInContext()
YProximity.FindProximityInContext()

Permet de retrouver un capteur de proximité d'après un identifiant donné dans un Context YAPI.

function **FindProximityInContext**(**yctx**, **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 proximité soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YProximity.isOnline()` pour tester si le capteur de proximité 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de proximité sans ambiguïté

Retourne :

un objet de classe `YProximity` qui permet ensuite de contrôler le capteur de proximité.

YProximity.FirstProximity() YProximity yFirstProximity()YProximity.FirstProximity() YProximity.FirstProximity()

Commence l'énumération des capteurs de proximité accessibles par la librairie.

function **FirstProximity**( )

Utiliser la fonction `YProximity.nextProximity()` pour itérer sur les autres capteurs de proximité.

Retourne :

un pointeur sur un objet `YProximity`, correspondant au premier capteur de proximité accessible en ligne, ou `null` si il n'y a pas de capteurs de proximité disponibles.

YProximity.FirstProximityInContext() YProximity yFirstProximityInContext() YProximity.FirstProximityInContext() YProximity.FirstProximityInContext()

Commence l'énumération des capteurs de proximité accessibles par la librairie.

function **FirstProximityInContext**(**yctx**)

Utiliser la fonction `YProximity.nextProximity()` pour itérer sur les autres capteurs de proximité.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YProximity`, correspondant au premier capteur de proximité accessible en ligne, ou `null` si il n'y a pas de capteurs de proximité disponibles.

proximity→calibrateFromPoints() YProximity
proximity.calibrateFromPoints()
proximity.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

proximity→clearCache()proximity.clearCache() YProximity

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur de proximité. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

proximity→describe()proximity.describe() YProximity

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de proximité au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le capteur de proximité (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

proximity→get_advertisedValue() YProximity
proximity→advertisedValue()
proximity.get_advertisedValue()
proximity.get_advertisedValue()

Retourne la valeur courante du capteur de proximité (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de proximité (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

proximity→get_currentRawValue() YProximity
proximity→currentRawValue()
proximity.get_currentRawValue()
proximity.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

proximity→get_currentValue() YProximity
proximity→currentValue()
proximity.get_currentValue()
proximity.get_currentValue()

Retourne la valeur actuelle de la détection de proximité, en l'unité spécifiée, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle de la détection de proximité, en l'unité spécifiée, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

proximity→get_dataLogger() YProximity
proximity→dataLogger()proximity.get_dataLogger()
proximity.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

proximity→get_detectionThreshold() YProximity
proximity→detectionThreshold()
proximity.get_detectionThreshold()
proximity.get_detectionThreshold()

Retourne le seuil utilisé pour déterminer l'état logique de la détection de proximité, lorsqu'on la traite comme une entrée binaire (on/off).

function **get_detectionThreshold**( )

Retourne :

un entier représentant le seuil utilisé pour déterminer l'état logique de la détection de proximité, lorsqu'on la traite comme une entrée binaire (on/off)

En cas d'erreur, déclenche une exception ou retourne `Y_DETECTIONTHRESHOLD_INVALID`.

proximity→get_errorMessage() YProximity
proximity→errorMessage()
proximity.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de proximité.

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 proximité.

proximity→get_errorType() YProximity
proximity→errorType()proximity.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de proximité.

```
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 proximité.

proximity→get_friendlyName() YProximity
proximity→friendlyName()
proximity.get_friendlyName()

Retourne un identifiant global du capteur de proximité au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du capteur de proximité si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du capteur de proximité (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le capteur de proximité en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

proximity→get_functionDescriptor() YProximity
proximity→functionDescriptor()
proximity.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

proximity→get_functionId() YProximity
proximity→functionId()proximity.get_functionId()

Retourne l'identifiant matériel du capteur de proximité, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de proximité (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

proximity→get_hardwareId() YProximity
proximity→hardwareId()proximity.get_hardwareId()

Retourne l'identifiant matériel unique du capteur de proximité 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 proximité (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le capteur de proximité (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

proximity→get_highestValue() YProximity
proximity→highestValue()
proximity.get_highestValue()
proximity.get_highestValue()

Retourne la valeur maximale observée pour la détection de proximité depuis le démarrage du module.

function **get_highestValue**( )

Retourne :

une valeur numérique représentant la valeur maximale observée pour la détection de proximité depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

proximity→get_isPresent() YProximity
proximity→isPresent()proximity.get_isPresent()
proximity.get_isPresent()

Retourne vrai si l'entrée (considérée comme binaire) est active (valeur de détection inférieure au seuil `threshold`), et faux sinon.

function **get_isPresent**( )

Retourne :

soit `Y_ISPRESENT_FALSE`, soit `Y_ISPRESENT_TRUE`, selon vrai si l'entrée (considérée comme binaire) est active (valeur de détection inférieure au seuil `threshold`), et faux sinon

En cas d'erreur, déclenche une exception ou retourne `Y_ISPRESENT_INVALID`.

proximity→get_lastTimeApproached() YProximity
proximity→lastTimeApproached()
proximity.get_lastTimeApproached()
proximity.get_lastTimeApproached()

Retourne le temps absolu (nombre de millisecondes) entre la mise sous tension du module et la dernière détection observée (transition de absent à présent).

function **get_lastTimeApproached**( )

Retourne :

un entier représentant le temps absolu (nombre de millisecondes) entre la mise sous tension du module et la dernière détection observée (transition de absent à présent)

En cas d'erreur, déclenche une exception ou retourne `Y_LASTTIMEAPPROACHED_INVALID`.

proximity→get_lastTimeRemoved() YProximity
proximity→lastTimeRemoved()
proximity.get_lastTimeRemoved()
proximity.get_lastTimeRemoved()

Retourne le temps absolu (nombre de millisecondes) entre la mise sous tension du module et la dernière détection observée (transition de présent à absent).

function **get_lastTimeRemoved**( )

Retourne :

un entier représentant le temps absolu (nombre de millisecondes) entre la mise sous tension du module et la dernière détection observée (transition de présent à absent)

En cas d'erreur, déclenche une exception ou retourne `Y_LASTTIMEREMOVED_INVALID`.

proximity→get_logFrequency() YProximity
proximity→logFrequency()
proximity.get_logFrequency()
proximity.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`.

proximity→get_logicalName() YProximity
proximity→logicalName()
proximity.get_logicalName()
proximity.get_logicalName()

Retourne le nom logique du capteur de proximité.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du capteur de proximité.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

proximity→get_lowestValue() YProximity
proximity→lowestValue()proximity.get_lowestValue()
proximity.get_lowestValue()

Retourne la valeur minimale observée pour la détection de proximité depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :

une valeur numérique représentant la valeur minimale observée pour la détection de proximité depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

proximity→get_module() YProximity
proximity→module()proximity.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

proximity→get_proximityReportMode() YProximity
proximity→proximityReportMode()
proximity.get_proximityReportMode()
proximity.get_proximityReportMode()

Retourne le type de paramètre (valeur du capteur, présence ou compteur d'impulsion) renvoyé par la fonction get_currentValue et les callback.

function **get_proximityReportMode**( )

Retourne :

une valeur parmi Y_PROXIMITYREPORTMODE_NUMERIC, Y_PROXIMITYREPORTMODE_PRESENCE et Y_PROXIMITYREPORTMODE_PULSECOUNT représentant le type de paramètre (valeur du capteur, présence ou compteur d'impulsion) renvoyé par la fonction get_currentValue et les callback

En cas d'erreur, déclenche une exception ou retourne Y_PROXIMITYREPORTMODE_INVALID.

proximity→get_pulseCounter() YProximity
proximity→pulseCounter()
proximity.get_pulseCounter()
proximity.get_pulseCounter()

Retourne la valeur du compteur d'impulsions.

function **get_pulseCounter**( )

La valeur est codée sur 32 bits. En cas de dépassement de capacité (>=2^32), le compteur repart à zéro. Le compteur peut être réinitialisé en appelant la méthode resetCounter().

Retourne :
un entier représentant la valeur du compteur d'impulsions

En cas d'erreur, déclenche une exception ou retourne `Y_PULSECOUNTER_INVALID`.

proximity→get_pulseTimer() YProximity
proximity→pulseTimer()proximity.get_pulseTimer()
proximity.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`.

proximity→get_recordedData() YProximity
proximity→recordedData()
proximity.get_recordedData()
proximity.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 find de l'intercalle 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.

proximity→get_reportFrequency() YProximity
proximity→reportFrequency()
proximity.get_reportFrequency()
proximity.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`.

proximity→get_resolution() YProximity
proximity→resolution()proximity.get_resolution()
proximity.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`.

proximity→get_sensorState() YProximity
proximity→sensorState()proximity.get_sensorState()
proximity.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :

un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

proximity→get_signalValue() YProximity
proximity→signalValue()proximity.get_signalValue()
proximity.get_signalValue()

Retourne la valeur actuelle du mesuré par le capteur de proximité.

function **get_signalValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle du mesuré par le capteur de proximité

En cas d'erreur, déclenche une exception ou retourne `Y_SIGNALVALUE_INVALID`.

proximity→get_unit() YProximity
proximity→unit()proximity.get_unit()
proximity.get_unit()

Retourne l'unité dans laquelle la détection de proximité est exprimée.

function **get_unit**( )

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la détection de proximité est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

proximity→get_userData() YProximity
proximity→userData()proximity.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.

proximity→isOnline()proximity.isOnline() YProximity

Vérifie si le module hébergeant le capteur de proximité est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache du capteur de proximité sont valides au moment de l'appel, le module est considéré joignable. 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 proximité est joignable, `false` sinon

proximity→load()proximity.load() YProximity

Met en cache les valeurs courantes du capteur de proximité, 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 occasionellement 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.

proximity→loadAttribute()proximity.loadAttribute() YProximity proximity.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

proximity→loadCalibrationPoints() YProximity
proximity.loadCalibrationPoints()
proximity.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

proximity→muteValueCallbacks() YProximity
proximity.muteValueCallbacks()
proximity.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

proximity→nextProximity()proximity.nextProximity() YProximity proximity.nextProximity()

Continue l'énumération des capteurs de proximité commencée à l'aide de `yFirstProximity()`.

function **nextProximity**( )

Retourne :

un pointeur sur un objet `YProximity` accessible en ligne, ou `null` lorsque l'énumération est terminée.

proximity→registerTimedReportCallback() YProximity
proximity.registerTimedReportCallback()
proximity.registerTimedReportCallback()

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.

proximity→registerValueCallback() YProximity
proximity.registerValueCallback()
proximity.registerValueCallback()

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.

proximity→resetCounter()proximity.resetCounter() YProximity proximity.resetCounter()

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.

proximity→set_detectionThreshold() YProximity
proximity→setDetectionThreshold()
proximity.set_detectionThreshold()
proximity.set_detectionThreshold()

Modifie le seuil utilisé pour déterminer l'état logique de la détection de proximité, lorsqu'on la traite comme une entrée binaire (on/off).

function **set_detectionThreshold**(**newval**)

Paramètres :

newval un entier représentant le seuil utilisé pour déterminer l'état logique de la détection de proximité, lorsqu'on la traite comme une entrée binaire (on/off)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

proximity→set_highestValue() YProximity
proximity→setHighestValue()
proximity.set_highestValue()
proximity.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.

proximity→set_logFrequency() YProximity
proximity→setLogFrequency()
proximity.set_logFrequency()
proximity.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.

proximity→set_logicalName() YProximity
proximity→setLogicalName()
proximity.set_logicalName()
proximity.set_logicalName()

Modifie le nom logique du capteur de proximité.

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 proximité.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

proximity→set_lowestValue() YProximity
proximity→setLowestValue()
proximity.set_lowestValue()
proximity.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.

proximity→set_proximityReportMode() YProximity
proximity→setProximityReportMode()
proximity.set_proximityReportMode()
proximity.set_proximityReportMode()

Change le type de paramètre (valeur du capteur, présence ou compteur d'impulsion) renvoyé par la fonction get_currentValue et les callback.

function **set_proximityReportMode**(**newval**)

Seuls les six digits de droite du nombre de changements d'état sont transmis, pour les valeurs plus grandes que un million, utiliser get_pulseCounter().

Paramètres :

newval une valeur parmi Y_PROXIMITYREPORTMODE_NUMERIC, Y_PROXIMITYREPORTMODE_PRESENCE et Y_PROXIMITYREPORTMODE_PULSECOUNT

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

proximity→set_reportFrequency() YProximity
proximity→setReportFrequency()
proximity.set_reportFrequency()
proximity.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.

proximity→set_resolution() YProximity
proximity→setResolution()proximity.set_resolution()
proximity.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.

proximity→set_userData() YProximity
proximity→setUserData()proximity.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

proximity→startDataLogger() YProximity
proximity.startDataLogger()
proximity.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

proximity→stopDataLogger() YProximity
proximity.stopDataLogger()
proximity.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

proximity→unmuteValueCallbacks() YProximity
proximity.unmuteValueCallbacks()
proximity.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

proximity→wait_async()proximity.wait_async() YProximity

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.

3.49. Interface de la fonction PwmInput

La classe YPwmInput permet de lire et de configurer les capteurs PWM Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, elle permet de configurer le paramètre du signal utilisé pour transmettre l'information: le duty cycle, le fréquence ou la longueur de la pulsation.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_pwminput.js'></script>
cpp	#include "yocto_pwminput.h"
m	#import "yocto_pwminput.h"
pas	uses yocto_pwminput;
vb	yocto_pwminput.vb
cs	yocto_pwminput.cs
java	import com.yoctopuce.YoctoAPI.YPwmInput;
uwp	import com.yoctopuce.YoctoAPI.YPwmInput;
py	from yocto_pwminput import *
php	require_once('yocto_pwminput.php');
es	in HTML: <script src="../../lib/yocto_pwminput.js"></script> in node.js: require('yoctolib-es2017/yocto_pwminput.js');

Fonction globales

yFindPwmInput(**func**)

Permet de retrouver une entrée PWM d'après un identifiant donné.

yFindPwmInputInContext(**yctx**, **func**)

Permet de retrouver une entrée PWM d'après un identifiant donné dans un Context YAPI.

yFirstPwmInput()

Commence l'énumération des Entrée PWM accessibles par la librairie.

yFirstPwmInputInContext(**yctx**)

Commence l'énumération des Entrée PWM accessibles par la librairie.

Méthodes des objets `YPwmInput`

pwminput→calibrateFromPoints(**rawValues**, **refValues**)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

pwminput→clearCache()

Invalide le cache.

pwminput→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'entrée PWM au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

pwminput→get_advertisedValue()

Retourne la valeur courante de l'entrée PWM (pas plus de 6 caractères).

pwminput→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Hz, sous forme de nombre à virgule.

pwminput→get_currentValue()

Retourne la valeur courante de la fonctionnalité PwmInput, sous forme de nombre à virgule.

pwminput→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

pwminput→get_dutyCycle()

Retourne le duty cycle du PWM, en pour cents.

pwminput→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'entrée PWM.

pwminput→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'entrée PWM.

pwminput→get_frequency()

Retourne la fréquence du PWM en Hz.

pwminput→get_friendlyName()

Retourne un identifiant global de l'entrée PWM au format `NOM_MODULE.NOM_FONCTION`.

pwminput→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

pwminput→get_functionId()

Retourne l'identifiant matériel de l'entrée PWM, sans référence au module.

pwminput→get_hardwareId()

Retourne l'identifiant matériel unique de l'entrée PWM au format `SERIAL.FUNCTIONID`.

pwminput→get_highestValue()

Retourne la valeur maximale observée pour le PWM depuis le démarrage du module.

pwminput→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.

pwminput→get_logicalName()

Retourne le nom logique de l'entrée PWM.

pwminput→get_lowestValue()

Retourne la valeur minimale observée pour le PWM depuis le démarrage du module.

pwminput→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

pwminput→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

pwminput→get_period()

Retourne la période du PWM en millisecondes.

pwminput→get_pulseCounter()

Retourne la valeur du compteur d'impulsions.

pwminput→get_pulseDuration()

Retourne la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule.

pwminput→get_pulseTimer()

Retourne le timer du compteur d'impulsions (ms).

pwminput→get_pwmReportMode()

Retourne le type de paramètre (fréquence, duty cycle , longueur d'impulsion ou nombre de changements d'état) renvoyé par la fonction get_currentValue et les callback.

pwminput→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.

pwminput→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

pwminput→get_resolution()

Retourne la résolution des valeurs mesurées.

pwminput→get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

pwminput→get_unit()

Retourne l'unité dans laquelle la valeur retournée par get_currentValue et les callback est exprimée.

pwminput→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

pwminput→isOnline()

Vérifie si le module hébergeant l'entrée PWM est joignable, sans déclencher d'erreur.

pwminput→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant l'entrée PWM est joignable, sans déclencher d'erreur.

pwminput→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

pwminput→load(**msValidity**)

Met en cache les valeurs courantes de l'entrée PWM, avec une durée de validité spécifiée.

pwminput→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

pwminput→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

pwminput→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes de l'entrée PWM, avec une durée de validité spécifiée.

pwminput→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

pwminput→nextPwmInput()

Continue l'énumération des Entrée PWM commencée à l'aide de `yFirstPwmInput()`.

pwminput→registerTimedReportCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

pwminput→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

pwminput→resetCounter()

Réinitialise le compteur d'impulsions et son timer.

pwminput→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

pwminput→set_logFrequency(**newval**)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

pwminput→set_logicalName(**newval**)

Modifie le nom logique de l'entrée PWM.

pwminput→set_lowestValue(**newval**)

Modifie la mémoire de valeur minimale observée.

pwminput→set_pwmReportMode(**newval**)

Change le type de paramètre (fréquence, duty cycle, longueur d'impulsion ou nombre de changement d'état) renvoyé par la fonction get_currentValue et les callback.

pwminput→set_reportFrequency(**newval**)

Modifie la fréquence de notification périodique des valeurs mesurées.

pwminput→set_resolution(**newval**)

Modifie la résolution des valeurs physique mesurées.

pwminput→set_userData(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

pwminput→startDataLogger()

Démarre l'enregistreur de données du module.

pwminput→stopDataLogger()

Arrête l'enregistreur de données du module.

pwminput→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

pwminput→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YPwmInput.FindPwmInput() YPwmInput yFindPwmInput()YPwmInput.FindPwmInput() YPwmInput.FindPwmInput()

Permet de retrouver une entrée PWM d'après un identifiant donné.

function **FindPwmInput**(**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 PWM soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmInput.isOnline()` pour tester si l'entrée 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence l'entrée PWM sans ambiguïté

Retourne :

un objet de classe `YPwmInput` qui permet ensuite de contrôler l'entrée PWM.

YPwmInput.FindPwmInputInContext() YPwmInput
yFindPwmInputInContext()
YPwmInput.FindPwmInputInContext()
YPwmInput.FindPwmInputInContext()

Permet de retrouver une entrée PWM d'après un identifiant donné dans un Context YAPI.

function **FindPwmInputInContext**(**yctx**, **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 PWM soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmInput.isOnline()` pour tester si l'entrée 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'entrée PWM sans ambiguïté

Retourne :

un objet de classe `YPwmInput` qui permet ensuite de contrôler l'entrée PWM.

YPwmInput.FirstPwmInput() YPwmInput yFirstPwmInput()YPwmInput.FirstPwmInput() YPwmInput.FirstPwmInput()

Commence l'énumération des Entrée PWM accessibles par la librairie.

function **FirstPwmInput**( )

Utiliser la fonction `YPwmInput.nextPwmInput()` pour itérer sur les autres Entrée PWM.

Retourne :

un pointeur sur un objet `YPwmInput`, correspondant à la première entrée PWM accessible en ligne, ou `null` si il n'y a pas de Entrée PWM disponibles.

YPwmInput.FirstPwmInputInContext() YPwmInput
yFirstPwmInputInContext()
YPwmInput.FirstPwmInputInContext()
YPwmInput.FirstPwmInputInContext()

Commence l'énumération des Entrée PWM accessibles par la librairie.

function **FirstPwmInputInContext**(**yctx**)

Utiliser la fonction YPwmInput.nextPwmInput() pour itérer sur les autres Entrée PWM.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet YPwmInput, correspondant à la première entrée PWM accessible en ligne, ou null si il n'y a pas de Entrée PWM disponibles.

pwminput→calibrateFromPoints() YPwmInput
pwminput.calibrateFromPoints()
pwminput.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→clearCache()pwminput.clearCache() YPwmInput

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'entrée PWM. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

pwminput→describe()pwminput.describe() YPwmInput

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'entrée PWM au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant l'entrée PWM (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

pwminput→get_advertisedValue() YPwmInput
pwminput→advertisedValue()
pwminput.get_advertisedValue()
pwminput.get_advertisedValue()

Retourne la valeur courante de l'entrée PWM (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante de l'entrée PWM (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

pwminput→get_currentRawValue() YPwmInput
pwminput→currentRawValue()
pwminput.get_currentRawValue()
pwminput.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Hz, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en Hz, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

pwminput→get_currentValue() YPwmInput
pwminput→currentValue()
pwminput.get_currentValue()
pwminput.get_currentValue()

Retourne la valeur courante de la fonctionnalité PwmInput, sous forme de nombre à virgule.

function **get_currentValue**( )

En fonction du réglage pwmReportMode, cela peut être soit la fréquence en Hz, le duty cycle en % ou encore la longueur d'impulsion en ms.

Retourne :

une valeur numérique représentant la valeur courante de la fonctionnalité PwmInput, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

pwminput→get_dataLogger() YPwmInput
pwminput→dataLogger()pwminput.get_dataLogger()
pwminput.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

pwminput→get_dutyCycle() YPwmInput
pwminput→dutyCycle()pwminput.get_dutyCycle()
pwminput.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`.

pwminput→get_errorMessage() YPwmInput
pwminput→errorMessage()
pwminput.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'entrée 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 de l'entrée PWM.

pwminput→get_errorType() YPwmInput
pwminput→errorType()pwminput.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'entrée 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 de l'entrée PWM.

pwminput→get_frequency() YPwmInput
pwminput→frequency()pwminput.get_frequency()
pwminput.get_frequency()

Retourne la fréquence du PWM en Hz.

function **get_frequency**( )

Retourne :
une valeur numérique représentant la fréquence du PWM en Hz

En cas d'erreur, déclenche une exception ou retourne `Y_FREQUENCY_INVALID`.

pwminput→get_friendlyName() YPwmInput
pwminput→friendlyName()
pwminput.get_friendlyName()

Retourne un identifiant global de l'entrée PWM au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et de l'entrée PWM si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel de l'entrée PWM (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant l'entrée PWM en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

pwminput→get_functionDescriptor() YPwmInput
pwminput→functionDescriptor()
pwminput.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

pwminput→get_functionId() YPwmInput
pwminput→functionId()pwminput.get_functionId()

Retourne l'identifiant matériel de l'entrée PWM, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant l'entrée PWM (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

pwminput→get_hardwareId() YPwmInput
pwminput→hardwareId()pwminput.get_hardwareId()

Retourne l'identifiant matériel unique de l'entrée PWM 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 PWM (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant l'entrée PWM (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

pwminput→get_highestValue() YPwmInput
pwminput→highestValue()
pwminput.get_highestValue()
pwminput.get_highestValue()

Retourne la valeur maximale observée pour le PWM depuis le démarrage du module.

function **get_highestValue**( )

Retourne :

une valeur numérique représentant la valeur maximale observée pour le PWM depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

pwminput→get_logFrequency() YPwmInput
pwminput→logFrequency()
pwminput.get_logFrequency()
pwminput.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`.

pwminput→get_logicalName() YPwmInput
pwminput→logicalName()
pwminput.get_logicalName()
pwminput.get_logicalName()

Retourne le nom logique de l'entrée PWM.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique de l'entrée PWM.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

pwminput→get_lowestValue() YPwmInput
pwminput→lowestValue()
pwminput.get_lowestValue()
pwminput.get_lowestValue()

Retourne la valeur minimale observée pour le PWM depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :

une valeur numérique représentant la valeur minimale observée pour le PWM depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

pwminput→get_module() YPwmInput
pwminput→module()pwminput.get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

function **get_module**( )

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :
une instance de YModule

pwminput→get_period() YPwmInput
pwminput→period()pwminput.get_period()
pwminput.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`.

pwminput→get_pulseCounter() YPwmInput
pwminput→pulseCounter()
pwminput.get_pulseCounter()
pwminput.get_pulseCounter()

Retourne la valeur du compteur d'impulsions.

function **get_pulseCounter**( )

Ce compteur est en réalité incrémenté deux fois par période. Ce compteur est limité à 1 milliard.

Retourne :
un entier représentant la valeur du compteur d'impulsions

En cas d'erreur, déclenche une exception ou retourne `Y_PULSECOUNTER_INVALID`.

pwminput→get_pulseDuration() YPwmInput
pwminput→pulseDuration()
pwminput.get_pulseDuration()
pwminput.get_pulseDuration()

Retourne la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule.

function **get_pulseDuration**( )

Retourne :

une valeur numérique représentant la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_PULSEDURATION_INVALID`.

pwminput→get_pulseTimer() YPwmInput
pwminput→pulseTimer()pwminput.get_pulseTimer()
pwminput.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`.

pwminput→get_pwmReportMode() YPwmInput
pwminput→pwmReportMode()
pwminput.get_pwmReportMode()
pwminput.get_pwmReportMode()

Retourne le type de paramètre (fréquence, duty cycle , longueur d'impulsion ou nombre de changements d'état) renvoyé par la fonction get_currentValue et les callback.

function **get_pwmReportMode**( )

Retourne :

une valeur parmi Y_PWMREPORTMODE_PWM_DUTYCYCLE, Y_PWMREPORTMODE_PWM_FREQUENCY, Y_PWMREPORTMODE_PWM_PULSEDURATION et Y_PWMREPORTMODE_PWM_EDGECOUNT représentant le type de paramètre (fréquence, duty cycle , longueur d'impulsion ou nombre de changements d'état) renvoyé par la fonction get_currentValue et les callback

En cas d'erreur, déclenche une exception ou retourne Y_PWMREPORTMODE_INVALID.

pwminput→get_recordedData() YPwmInput
pwminput→recordedData()
pwminput.get_recordedData()
pwminput.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 find de l'intercalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

pwminput→get_reportFrequency() YPwmInput
pwminput→reportFrequency()
pwminput.get_reportFrequency()
pwminput.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`.

pwminput→get_resolution() YPwmInput
pwminput→resolution()pwminput.get_resolution()
pwminput.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`.

pwminput→get_sensorState() YPwmInput
pwminput→sensorState()
pwminput.get_sensorState()
pwminput.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :

un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

pwminput→get_unit() YPwmInput
pwminput→unit()pwminput.get_unit()
pwminput.get_unit()

Retourne l'unité dans laquelle la valeur retournée par get_currentValue et les callback est exprimée.

function **get_unit**( )

Cette unité dépend du réglage pwmReportMode.

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la valeur retournée par get_currentValue et les callback est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

pwminput→get_userData() YPwmInput
pwminput→userData()pwminput.get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

function **get_userData**( )

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.

pwminput→isOnline()pwminput.isOnline() YPwmInput

Vérifie si le module hébergeant l'entrée PWM est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache de l'entrée 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 l'entrée PWM est joignable, `false` sinon

pwminput→load()pwminput.load() YPwmInput

Met en cache les valeurs courantes de l'entrée 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 occasionellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→loadAttribute()pwminput.loadAttribute() YPwmInput pwminput.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

pwminput→loadCalibrationPoints() YPwmInput
pwminput.loadCalibrationPoints()
pwminput.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→muteValueCallbacks() YPwmInput
pwminput.muteValueCallbacks()
pwminput.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→nextPwmInput() YPwmInput
pwminput.nextPwmInput()pwminput.nextPwmInput()

Continue l'énumération des Entrée PWM commencée à l'aide de `yFirstPwmInput()`.

function **nextPwmInput**( )

Retourne :

un pointeur sur un objet `YPwmInput` accessible en ligne, ou `null` lorsque l'énumération est terminée.

pwminput→registerTimedReportCallback() YPwmInput
pwminput.registerTimedReportCallback()
pwminput.registerTimedReportCallback()

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.

pwminput→registerValueCallback() YPwmInput
pwminput.registerValueCallback()
pwminput.registerValueCallback()

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.

pwminput→resetCounter()pwminput.resetCounter() YPwmInput pwminput.resetCounter()

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.

pwminput→set_highestValue() YPwmInput
pwminput→setHighestValue()
pwminput.set_highestValue()
pwminput.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.

pwminput→set_logFrequency() YPwmInput
pwminput→setLogFrequency()
pwminput.set_logFrequency()
pwminput.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.

pwminput→set_logicalName() YPwmInput
pwminput→setLogicalName()
pwminput.set_logicalName()
pwminput.set_logicalName()

Modifie le nom logique de l'entrée 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 de l'entrée 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.

pwminput→set_lowestValue() YPwmInput
pwminput→setLowestValue()
pwminput.set_lowestValue()
pwminput.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.

pwminput→set_pwmReportMode() YPwmInput
pwminput→setPwmReportMode()
pwminput.set_pwmReportMode()
pwminput.set_pwmReportMode()

Change le type de paramètre (fréquence, duty cycle, longueur d'impulsion ou nombre de changement d'état) renvoyé par la fonction get_currentValue et les callback.

function **set_pwmReportMode**(**newval**)

Seuls les six digits de droite du nombre de changements d'état sont transmis, pour les valeurs plus grandes que un million, utiliser get_pulseCounter().

Paramètres :

newval une valeur parmi Y_PWMREPORTMODE_PWM_DUTYCYCLE, Y_PWMREPORTMODE_PWM_FREQUENCY, Y_PWMREPORTMODE_PWM_PULSEDURATION et Y_PWMREPORTMODE_PWM_EDGECOUNT

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→set_reportFrequency() YPwmInput
pwminput→setReportFrequency()
pwminput.set_reportFrequency()
pwminput.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.

pwminput→set_resolution() YPwmInput
pwminput→setResolution()
pwminput.set_resolution()pwminput.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.

pwminput→set_userData() YPwmInput
pwminput→setUserData()pwminput.set_userData()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

```
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

pwminput→startDataLogger() YPwmInput
pwminput.startDataLogger()
pwminput.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

pwminput→stopDataLogger() YPwmInput
pwminput.stopDataLogger()
pwminput.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

pwminput→unmuteValueCallbacks() YPwmInput
pwminput.unmuteValueCallbacks()
pwminput.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→wait_async()pwminput.wait_async() YPwmInput

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.

3.50. Interface de la fonction PwmOutput

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>
cpp	#include "yocto_pwmoutput.h"
m	#import "yocto_pwmoutput.h"
pas	uses yocto_pwmoutput;
vb	yocto_pwmoutput.vb
cs	yocto_pwmoutput.cs
java	import com.yoctopuce.YoctoAPI.YPwmOutput;
uwp	import com.yoctopuce.YoctoAPI.YPwmOutput;
py	from yocto_pwmoutput import *
php	require_once('yocto_pwmoutput.php');
es	in HTML: <script src="../../lib/yocto_pwmoutput.js"></script> in node.js: require('yoctolib-es2017/yocto_pwmoutput.js');

Fonction globales

yFindPwmOutput(**func**)

Permet de retrouver un PWM d'après un identifiant donné.

yFindPwmOutputInContext(**yctx**, **func**)

Permet de retrouver un PWM d'après un identifiant donné dans un Context YAPI.

yFirstPwmOutput()

Commence l'énumération des PWM accessibles par la librairie.

yFirstPwmOutputInContext(**yctx**)

Commence l'énumération des PWM accessibles par la librairie.

Méthodes des objets YPwmOutput

pwmoutput→clearCache()

Invalide le cache.

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, sous forme d'un chiffre à virgule.

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→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

pwmoutput→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

pwmoutput→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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 en millisecondes.

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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

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() YPwmOutput yFindPwmOutput()YPwmOutput.FindPwmOutput() YPwmOutput.FindPwmOutput()

Permet de retrouver un PWM d'après un identifiant donné.

function **FindPwmOutput**(**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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindPwmOutputInContext() YPwmOutput yFindPwmOutputInContext()

YPwmOutput.FindPwmOutputInContext()
YPwmOutput.FindPwmOutputInContext()

Permet de retrouver un PWM d'après un identifiant donné dans un Context YAPI.

function **FindPwmOutputInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le PWM sans ambiguïté

Retourne :

un objet de classe `YPwmOutput` qui permet ensuite de contrôler le PWM.

YPwmOutput.FirstPwmOutput() YPwmOutput yFirstPwmOutput()YPwmOutput.FirstPwmOutput() YPwmOutput.FirstPwmOutput()

Commence l'énumération des PWM accessibles par la librairie.

function **FirstPwmOutput**( )

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.

YPwmOutput.FirstPwmOutputInContext() YPwmOutput yFirstPwmOutputInContext()

YPwmOutput.FirstPwmOutputInContext()
YPwmOutput.FirstPwmOutputInContext()

Commence l'énumération des PWM accessibles par la librairie.

function **FirstPwmOutputInContext**(**yctx**)

Utiliser la fonction YPwmOutput.nextPwmOutput() pour itérer sur les autres PWM.

Paramètres :

yctx un contexte YAPI.

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→clearCache()pwmoutput.clearCache() YPwmOutput

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du PWM. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le PWM (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

pwmoutput→dutyCycleMove() YPwmOutput
pwmoutput.dutyCycleMove()
pwmoutput.dutyCycleMove()

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()
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()
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()
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()
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()
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()
pwmoutput.get_frequency()

Retourne la fréquence du PWM en Hz.

function **get_frequency**( )

Retourne :
une valeur numérique représentant la fréquence du PWM en Hz

En cas d'erreur, déclenche une exception ou retourne `Y_FREQUENCY_INVALID`.

pwmoutput→get_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'identifant 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() YPwmOutput
pwmoutput→functionDescriptor()
pwmoutput.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

pwmoutput→get_functionId() YPwmOutput
pwmoutput→functionId()pwmoutput.get_functionId()

Retourne l'identifiant matériel du PWM, sans référence au module.

function **get_functionId**( )

Par example `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 example 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() YPwmOutput
pwmoutput→logicalName()
pwmoutput.get_logicalName()
pwmoutput.get_logicalName()

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_period() YPwmOutput
pwmoutput→period()pwmoutput.get_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()
pwmoutput.get_pulseDuration()

Retourne la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule.

function **get_pulseDuration**( )

Retourne :

une valeur numérique représentant la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_PULSEDURATION_INVALID`.

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→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 occasionellement 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→loadAttribute() YPwmOutput pwmoutput.loadAttribute()pwmoutput.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

pwmoutput→muteValueCallbacks() YPwmOutput
pwmoutput.muteValueCallbacks()
pwmoutput.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→nextPwmOutput() YPwmOutput
pwmoutput.nextPwmOutput()
pwmoutput.nextPwmOutput()

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() YPwmOutput
pwmoutput.pulseDurationMove()
pwmoutput.pulseDurationMove()

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() YPwmOutput
pwmoutput.registerValueCallback()
pwmoutput.registerValueCallback()

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() YPwmOutput
pwmoutput→setDutyCycle()
pwmoutput.set_dutyCycle()
pwmoutput.set_dutyCycle()

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() YPwmOutput
pwmoutput→setDutyCycleAtPowerOn()
pwmoutput.set_dutyCycleAtPowerOn()
pwmoutput.set_dutyCycleAtPowerOn()

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()
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() YPwmOutput
pwmoutput→setEnabledAtPowerOn()
pwmoutput.set_enabledAtPowerOn()
pwmoutput.set_enabledAtPowerOn()

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() YPwmOutput
pwmoutput→setFrequency()
pwmoutput.set_frequency()
pwmoutput.set_frequency()

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 une valeur numérique représentant la fréquence du PWM

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→set_logicalName() YPwmOutput
pwmoutput→setLogicalName()
pwmoutput.set_logicalName()
pwmoutput.set_logicalName()

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()
pwmoutput.set_period()

Modifie la période du PWM en millisecondes.

function **set_period**(**newval**)

Paramètres :

newval une valeur numérique représentant la période du PWM en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→set_pulseDuration() YPwmOutput
pwmoutput→setPulseDuration()
pwmoutput.set_pulseDuration()
pwmoutput.set_pulseDuration()

Modifie la longueur des impulsions du PWM, en millisecondes.

function **set_pulseDuration**(**newval**)

Attention, la longueur d'une impulsion ne peut pas être plus grande que la période, sinon la longueur sera automatiquement tronquée à la période.

Paramètres :

newval une valeur numérique représentant la longueur des impulsions du PWM, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→set_userData() YPwmOutput
pwmoutput→setUserData()
pwmoutput.set_userData()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

pwmoutput→unmuteValueCallbacks() YPwmOutput
pwmoutput.unmuteValueCallbacks()
pwmoutput.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.51. 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>
cpp	#include "yocto_pwmpowersource.h"
m	#import "yocto_pwmpowersource.h"
pas	uses yocto_pwmpowersource;
vb	yocto_pwmpowersource.vb
cs	yocto_pwmpowersource.cs
java	import com.yoctopuce.YoctoAPI.YPwmPowerSource;
uwp	import com.yoctopuce.YoctoAPI.YPwmPowerSource;
py	from yocto_pwmpowersource import *
php	require_once('yocto_pwmpowersource.php');
es	in HTML: <script src="../../lib/yocto_pwmpowersource.js"></script> in node.js: require('yoctolib-es2017/yocto_pwmpowersource.js');

Fonction globales

yFindPwmPowerSource(**func**)

Permet de retrouver une source de tension d'après un identifiant donné.

yFindPwmPowerSourceInContext(**yctx**, **func**)

Permet de retrouver une source de tension d'après un identifiant donné dans un Context YAPI.

yFirstPwmPowerSource()

Commence l'énumération des Source de tension accessibles par la librairie.

yFirstPwmPowerSourceInContext(**yctx**)

Commence l'énumération des Source de tension accessibles par la librairie.

Méthodes des objets YPwmPowerSource

pwmpowersource→clearCache()

Invalide le cache.

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→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

pwmpowersource→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

pwmpowersource→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

pwmpowersource→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YPwmPowerSource.FindPwmPowerSource() YPwmPowerSource
yFindPwmPowerSource()
YPwmPowerSource.FindPwmPowerSource()
YPwmPowerSource.FindPwmPowerSource()

Permet de retrouver une source de tension d'après un identifiant donné.

function **FindPwmPowerSource**(**func**)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la source de tension soit en ligne au moment 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindPwmPowerSourceInContext() YPwmPowerSource yFindPwmPowerSourceInContext() YPwmPowerSource.FindPwmPowerSourceInContext() YPwmPowerSource.FindPwmPowerSourceInContext()

Permet de retrouver une source de tension d'après un identifiant donné dans un Context YAPI.

function **FindPwmPowerSourceInContext**(**yctx**, **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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence la source de tension sans ambiguïté

Retourne :

un objet de classe `YPwmPowerSource` qui permet ensuite de contrôler la source de tension.

YPwmPowerSource.FirstPwmPowerSource() YPwmPowerSource
yFirstPwmPowerSource()
YPwmPowerSource.FirstPwmPowerSource()
YPwmPowerSource.FirstPwmPowerSource()

Commence l'énumération des Source de tension accessibles par la librairie.

function **FirstPwmPowerSource**( )

Utiliser la fonction `YPwmPowerSource.nextPwmPowerSource()` pour itérer sur les autres Source de tension.

Retourne :

un pointeur sur un objet `YPwmPowerSource`, correspondant à la première source de tension accessible en ligne, ou `null` si il n'y a pas de Source de tension disponibles.

YPwmPowerSource.FirstPwmPowerSourceInContext() YPwmPowerSource yFirstPwmPowerSourceInContext() YPwmPowerSource.FirstPwmPowerSourceInContext() YPwmPowerSource.FirstPwmPowerSourceInContext()

Commence l'énumération des Source de tension accessibles par la librairie.

function **FirstPwmPowerSourceInContext**(**yctx**)

Utiliser la fonction `YPwmPowerSource.nextPwmPowerSource()` pour itérer sur les autres Source de tension.

Paramètres :

yctx un contexte YAPI.

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→clearCache() YPwmPowerSource
pwmpowersource.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de la source de tension. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

pwmpowersource→describe() 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.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

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()
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'identifant 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() YPwmPowerSource
pwmpowersource→functionDescriptor()
pwmpowersource.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

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 example `relay1`.

Retourne :

une chaîne de caractères identifiant la source de tension (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

pwmpowersource→get_hardwareId() 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 example 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()
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_powerMode() YPwmPowerSource
pwmpowersource→powerMode()
pwmpowersource.get_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() YPwmPowerSource
pwmpowersource.isOnline()

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→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 occasionellement 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→loadAttribute() YPwmPowerSource
pwmpowersource.loadAttribute()
pwmpowersource.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

pwmpowersource→muteValueCallbacks() YPwmPowerSource
pwmpowersource.muteValueCallbacks()
pwmpowersource.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmpowersource→nextPwmPowerSource() YPwmPowerSource
pwmpowersource.nextPwmPowerSource()
pwmpowersource.nextPwmPowerSource()

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() YPwmPowerSource
pwmpowersource.registerValueCallback()
pwmpowersource.registerValueCallback()

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()
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() YPwmPowerSource
pwmpowersource→setPowerMode()
pwmpowersource.set_powerMode()
pwmpowersource.set_powerMode()

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→unmuteValueCallbacks() YPwmPowerSource
pwmpowersource.unmuteValueCallbacks()
pwmpowersource.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→wait_async() YPwmPowerSource
pwmpowersource.wait_async()

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.

3.52. Interface du quaternion

La class YQt de la librairie Yoctopuce permet d'accéder à l'estimation de l'orientation tridimentionnelle du Yocto-3D sous forme d'un quaternion. Il n'est en général pas nécessaire d'y accéder directement, la classe YGyro offrant une abstraction de plus haut niveau.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_gyro.js'></script>
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;
uwp	import com.yoctopuce.YoctoAPI.YGyro;
py	from yocto_gyro import *
php	require_once('yocto_gyro.php');
es	in HTML: <script src="../../lib/yocto_gyro.js"></script> in node.js: require('yoctolib-es2017/yocto_gyro.js');

Fonction globales

yFindQt(**func**)

Permet de retrouver un élément de quaternion d'après un identifiant donné.

yFindQtInContext(**yctx**, **func**)

Permet de retrouver un élément de quaternion d'après un identifiant donné dans un Context YAPI.

yFirstQt()

Commence l'énumération des éléments de quaternion accessibles par la librairie.

yFirstQtInContext(**yctx**)

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→clearCache()

Invalide le cache.

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), en unités, sous forme de nombre à virgule.

qt→get_currentValue()

Retourne la valeur actuelle de la coordonnée, en unités, sous forme de nombre à virgule.

qt→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

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.

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_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

qt→load(**msValidity**)

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

qt→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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→startDataLogger()

Démarre l'enregistreur de données du module.

qt→stopDataLogger()

Arrête l'enregistreur de données du module.

qt→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

qt→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YQt.FindQt() YQt yFindQt()YQt.FindQt()YQt.FindQt()

Permet de retrouver un élément de quaternion d'après un identifiant donné.

```
function FindQt( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindQtInContext() YQt yFindQtInContext()YQt.FindQtInContext() YQt.FindQtInContext()

Permet de retrouver un élément de quaternion d'après un identifiant donné dans un Context YAPI.

function **FindQtInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'élément de quaternion sans ambiguïté

Retourne :

un objet de classe `YQt` qui permet ensuite de contrôler l'élément de quaternion.

YQt.FirstQt() YQt yFirstQt()YQt.FirstQt()YQt.FirstQt()

Commence l'énumération des éléments de quaternion accessibles par la librairie.

```
function FirstQt( )
```

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.

YQt.FirstQtInContext() YQt yFirstQtInContext()YQt.FirstQtInContext() YQt.FirstQtInContext()

Commence l'énumération des éléments de quaternion accessibles par la librairie.

function **FirstQtInContext**(**yctx**)

Utiliser la fonction `YQt.nextQt()` pour itérer sur les autres éléments de quaternion.

Paramètres :

yctx un contexte YAPI.

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 qt.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→clearCache()qt.clearCache() YQt

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'élément de quaternion. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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()
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() qt.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en unités, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en unités, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

qt→get_currentValue() YQt
qt→currentValue()qt.get_currentValue()
qt.get_currentValue()

Retourne la valeur actuelle de la coordonnée, en unités, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle de la coordonnée, en unités, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

qt→get_dataLogger() YQt
qt→dataLogger()qt.get_dataLogger()
qt.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

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'identifant 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 example `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 example 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()
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()
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()
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()
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_recordedData() YQt
qt→recordedData()qt.get_recordedData()
qt.get_recordedData()

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

function **get_recordedData**(**startTime**, **endTime**)

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la find de l'intercalle 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()
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()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_sensorState() YQt
qt→sensorState()qt.get_sensorState()
qt.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :
un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

qt→get_unit() YQt
qt→unit()qt.get_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.

qt→isOnline()qt.isOnline() YQt

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache de l'élément de quaternion sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'élément de quaternion est joignable, `false` sinon

qt→isSensorReady() YQt

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

Retourne faux si le module n'est pas joignable, ou que le capteur n'a pas de mesure actuelle à communiquer. 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 dispose d'une mesure actuelle, `false` sinon

qt→load()qt.load() YQt

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

function **load**(**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 occasionellement 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→loadAttribute()qt.loadAttribute()qt.loadAttribute() YQt

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

qt→loadCalibrationPoints()qt.loadCalibrationPoints() YQt qt.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→muteValueCallbacks()qt.muteValueCallbacks() YQt qt.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→nextQt()qt.nextQt()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() YQt
qt.registerTimedReportCallback()
qt.registerTimedReportCallback()

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() YQt qt.registerValueCallback()qt.registerValueCallback()

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()
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()
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()
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()
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()
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()
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'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

qt→startDataLogger()qt.startDataLogger() YQt qt.startDataLogger()

Démarre l'enregistreur de données du module.

```
function startDataLogger( )
```

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

qt→stopDataLogger()qt.stopDataLogger() YQt qt.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

qt→unmuteValueCallbacks() YQt
qt.unmuteValueCallbacks()
qt.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.53. Interface de la fonction QuadratureDecoder

La classe YQuadratureDecoder permet de décoder un signal produit par un encodeur en quadrature. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_quadraturedecoder.js'></script>
cpp	#include "yocto_quadraturedecoder.h"
m	#import "yocto_quadraturedecoder.h"
pas	uses yocto_quadraturedecoder;
vb	yocto_quadraturedecoder.vb
cs	yocto_quadraturedecoder.cs
java	import com.yoctopuce.YoctoAPI.YQuadratureDecoder;
uwp	import com.yoctopuce.YoctoAPI.YQuadratureDecoder;
py	from yocto_quadraturedecoder import *
php	require_once('yocto_quadraturedecoder.php');
es	in HTML: <script src="../../lib/yocto_quadraturedecoder.js"></script> in node.js: require('yoctolib-es2017/yocto_quadraturedecoder.js');

Fonction globales

yFindQuadratureDecoder(**func**)

Permet de retrouver un décodeut de quadrature d'après un identifiant donné.

yFindQuadratureDecoderInContext(**yctx**, **func**)

Permet de retrouver un décodeut de quadrature d'après un identifiant donné dans un Context YAPI.

yFirstQuadratureDecoder()

Commence l'énumération des Décodeur de quadrature accessibles par la librairie.

yFirstQuadratureDecoderInContext(**yctx**)

Commence l'énumération des Décodeur de quadrature accessibles par la librairie.

Méthodes des objets `YQuadratureDecoder`

quadraturedecoder→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.

quadraturedecoder→clearCache()

Invalide le cache.

quadraturedecoder→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du décodeur de quadrature au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

quadraturedecoder→get_advertisedValue()

Retourne la valeur courante du décodeur de quadrature (pas plus de 6 caractères).

quadraturedecoder→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en pas, sous forme de nombre à virgule.

quadraturedecoder→get_currentValue()

Retourne la valeur actuelle de la position, en pas, sous forme de nombre à virgule.

quadraturedecoder→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

quadraturedecoder→get_decoding()

Retourne l'état d'activation du décodeur de quadrature.

quadraturedecoder→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du décodeur de quadrature.

quadraturedecoder→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du décodeur de quadrature.

quadraturedecoder→get_friendlyName()

Retourne un identifiant global du décodeur de quadrature au format `NOM_MODULE.NOM_FONCTION`.

quadraturedecoder→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

quadraturedecoder→get_functionId()

Retourne l'identifiant matériel du décodeur de quadrature, sans référence au module.

quadraturedecoder→get_hardwareId()

Retourne l'identifiant matériel unique du décodeur de quadrature au format `SERIAL.FUNCTIONID`.

quadraturedecoder→get_highestValue()

Retourne la valeur maximale observée pour la position depuis le démarrage du module.

quadraturedecoder→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.

quadraturedecoder→get_logicalName()

Retourne le nom logique du décodeur de quadrature.

quadraturedecoder→get_lowestValue()

Retourne la valeur minimale observée pour la position depuis le démarrage du module.

quadraturedecoder→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

quadraturedecoder→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

quadraturedecoder→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.

quadraturedecoder→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

quadraturedecoder→get_resolution()

Retourne la résolution des valeurs mesurées.

quadraturedecoder→get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

quadraturedecoder→get_speed()

Retourne la fréquence des incréments, en Hz.

quadraturedecoder→get_unit()

Retourne l'unité dans laquelle la position est exprimée.

quadraturedecoder→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

quadraturedecoder→isOnline()

Vérifie si le module hébergeant le décodeur de quadrature est joignable, sans déclencher d'erreur.

quadraturedecoder→isOnline_async(callback, context)

Vérifie si le module hébergeant le décodeur de quadrature est joignable, sans déclencher d'erreur.

quadraturedecoder→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

quadraturedecoder→load(**msValidity**)

Met en cache les valeurs courantes du décodeur de quadrature, avec une durée de validité spécifiée.

quadraturedecoder→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

quadraturedecoder→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

quadraturedecoder→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du décodeur de quadrature, avec une durée de validité spécifiée.

quadraturedecoder→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

quadraturedecoder→nextQuadratureDecoder()

Continue l'énumération des Décodeur de quadrature commencée à l'aide de `yFirstQuadratureDecoder()`.

quadraturedecoder→registerTimedReportCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

quadraturedecoder→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

quadraturedecoder→set_currentValue(**newval**)

Modifie la position actuelle supposée par le décodeur de quadrature.

quadraturedecoder→set_decoding(**newval**)

Modifie l'état d'activation du décodeur de quadrature.

quadraturedecoder→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

quadraturedecoder→set_logFrequency(**newval**)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

quadraturedecoder→set_logicalName(**newval**)

Modifie le nom logique du décodeur de quadrature.

quadraturedecoder→set_lowestValue(**newval**)

Modifie la mémoire de valeur minimale observée.

quadraturedecoder→set_reportFrequency(**newval**)

Modifie la fréquence de notification périodique des valeurs mesurées.

quadraturedecoder→set_resolution(**newval**)

Modifie la résolution des valeurs physique mesurées.

quadraturedecoder→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`.

quadraturedecoder→startDataLogger()

Démarre l'enregistreur de données du module.

quadraturedecoder→stopDataLogger()

Arrête l'enregistreur de données du module.

quadraturedecoder→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

quadraturedecoder→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.

YQuadratureDecoder.FindQuadratureDecoder() YQuadratureDecoder yFindQuadratureDecoder() YQuadratureDecoder.FindQuadratureDecoder() YQuadratureDecoder.FindQuadratureDecoder()

Permet de retrouver un décodeut de quadrature d'après un identifiant donné.

```
function FindQuadratureDecoder( 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 décodeur de quadrature soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YQuadratureDecoder.isOnline()` pour tester si le décodeur de quadrature 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le décodeur de quadrature sans ambiguïté

Retourne :

un objet de classe `YQuadratureDecoder` qui permet ensuite de contrôler le décodeur de quadrature.

YQuadratureDecoder.FindQuadratureDecoderInContext() YQuadratureDecoder yFindQuadratureDecoderInContext() YQuadratureDecoder.FindQuadratureDecoderInContext() YQuadratureDecoder.FindQuadratureDecoderInContext()

Permet de retrouver un décodeut de quadrature d'après un identifiant donné dans un Context YAPI.

function **FindQuadratureDecoderInContext**(**yctx**, **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 décodeur de quadrature soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YQuadratureDecoder.isOnline()` pour tester si le décodeur de quadrature 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le décodeur de quadrature sans ambiguïté

Retourne :

un objet de classe `YQuadratureDecoder` qui permet ensuite de contrôler le décodeur de quadrature.

YQuadratureDecoder.FirstQuadratureDecoder() YQuadratureDecoder yFirstQuadratureDecoder() YQuadratureDecoder.FirstQuadratureDecoder() YQuadratureDecoder.FirstQuadratureDecoder()

Commence l'énumération des Décodeur de quadrature accessibles par la librairie.

function **FirstQuadratureDecoder**( )

Utiliser la fonction `YQuadratureDecoder.nextQuadratureDecoder()` pour itérer sur les autres Décodeur de quadrature.

Retourne :

un pointeur sur un objet `YQuadratureDecoder`, correspondant au premier décodeur de quadrature PWM accessible en ligne, ou `null` si il n'y a pas de Décodeur de quadrature disponibles.

YQuadratureDecoder.FirstQuadratureDecoderInContext() YQuadratureDecoder yFirstQuadratureDecoderInContext() YQuadratureDecoder.FirstQuadratureDecoderInContext() YQuadratureDecoder.FirstQuadratureDecoderInContext()

Commence l'énumération des Décodeur de quadrature accessibles par la librairie.

function **FirstQuadratureDecoderInContext**(**yctx**)

Utiliser la fonction `YQuadratureDecoder.nextQuadratureDecoder()` pour itérer sur les autres Décodeur de quadrature.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YQuadratureDecoder`, correspondant au premier décodeur de quadrature PWM accessible en ligne, ou `null` si il n'y a pas de Décodeur de quadrature disponibles.

quadraturedecoder→calibrateFromPoints() YQuadratureDecoder
quadraturedecoder.calibrateFromPoints()
quadraturedecoder.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

quadraturedecoder→clearCache() YQuadratureDecoder
quadraturedecoder.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du décodeur de quadrature. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

quadraturedecoder→describe() YQuadratureDecoder
quadraturedecoder.describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du décodeur de quadrature au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le décodeur de quadrature (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

quadraturedecoder→get_advertisedValue() YQuadratureDecoder
quadraturedecoder→advertisedValue()
quadraturedecoder.get_advertisedValue()
quadraturedecoder.get_advertisedValue()

Retourne la valeur courante du décodeur de quadrature (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :

une chaîne de caractères représentant la valeur courante du décodeur de quadrature (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

quadraturedecoder→get_currentRawValue() YQuadratureDecoder
quadraturedecoder→currentRawValue()
quadraturedecoder.get_currentRawValue()
quadraturedecoder.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en pas, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en pas, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

quadraturedecoder→get_currentValue() YQuadratureDecoder
quadraturedecoder→currentValue()
quadraturedecoder.get_currentValue()
quadraturedecoder.get_currentValue()

Retourne la valeur actuelle de la position, en pas, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle de la position, en pas, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

quadraturedecoder→get_dataLogger() YQuadratureDecoder
quadraturedecoder→dataLogger()
quadraturedecoder.get_dataLogger()
quadraturedecoder.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

quadraturedecoder→get_decoding() YQuadratureDecoder
quadraturedecoder→decoding()
quadraturedecoder.get_decoding()
quadraturedecoder.get_decoding()

Retourne l'état d'activation du décodeur de quadrature.

function **get_decoding**( )

Retourne :

soit `Y_DECODING_OFF`, soit `Y_DECODING_ON`, selon l'état d'activation du décodeur de quadrature

En cas d'erreur, déclenche une exception ou retourne `Y_DECODING_INVALID`.

quadraturedecoder→get_errorMessage() YQuadratureDecoder
quadraturedecoder→errorMessage()
quadraturedecoder.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du décodeur de quadrature.

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 décodeur de quadrature.

quadraturedecoder→get_errorType() YQuadratureDecoder
quadraturedecoder→errorType()
quadraturedecoder.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du décodeur de quadrature.

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 décodeur de quadrature.

quadraturedecoder→get_friendlyName() YQuadratureDecoder
quadraturedecoder→friendlyName()
quadraturedecoder.get_friendlyName()

Retourne un identifiant global du décodeur de quadrature au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du décodeur de quadrature si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du décodeur de quadrature (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le décodeur de quadrature en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

quadraturedecoder→get_functionDescriptor() YQuadratureDecoder
quadraturedecoder→functionDescriptor()
quadraturedecoder.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

quadraturedecoder→get_functionId() YQuadratureDecoder
quadraturedecoder→functionId()
quadraturedecoder.get_functionId()

Retourne l'identifiant matériel du décodeur de quadrature, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le décodeur de quadrature (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

quadraturedecoder→get_hardwareId() YQuadratureDecoder
quadraturedecoder→hardwareId()
quadraturedecoder.get_hardwareId()

Retourne l'identifiant matériel unique du décodeur de quadrature au format SERIAL.FUNCTIONID.

function **get_hardwareId**( )

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du décodeur de quadrature (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le décodeur de quadrature (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

quadraturedecoder→get_highestValue() YQuadratureDecoder
quadraturedecoder→highestValue()
quadraturedecoder.get_highestValue()
quadraturedecoder.get_highestValue()

Retourne la valeur maximale observée pour la position depuis le démarrage du module.

function **get_highestValue**( )

Retourne :

une valeur numérique représentant la valeur maximale observée pour la position depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

quadraturedecoder→get_logFrequency() YQuadratureDecoder
quadraturedecoder→logFrequency()
quadraturedecoder.get_logFrequency()
quadraturedecoder.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`.

quadraturedecoder→get_logicalName() YQuadratureDecoder
quadraturedecoder→logicalName()
quadraturedecoder.get_logicalName()
quadraturedecoder.get_logicalName()

Retourne le nom logique du décodeur de quadrature.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du décodeur de quadrature.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

quadraturedecoder→get_lowestValue() YQuadratureDecoder
quadraturedecoder→lowestValue()
quadraturedecoder.get_lowestValue()
quadraturedecoder.get_lowestValue()

Retourne la valeur minimale observée pour la position depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :

une valeur numérique représentant la valeur minimale observée pour la position depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

quadraturedecoder→get_module() YQuadratureDecoder
quadraturedecoder→module()
quadraturedecoder.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

quadraturedecoder→get_recordedData() YQuadratureDecoder
quadraturedecoder→recordedData()
quadraturedecoder.get_recordedData()
quadraturedecoder.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 find de l'intercalle 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.

quadraturedecoder→get_reportFrequency() YQuadratureDecoder
quadraturedecoder→reportFrequency()
quadraturedecoder.get_reportFrequency()
quadraturedecoder.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`.

quadraturedecoder→get_resolution() YQuadratureDecoder
quadraturedecoder→resolution()
quadraturedecoder.get_resolution()
quadraturedecoder.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`.

quadraturedecoder→get_sensorState() YQuadratureDecoder
quadraturedecoder→sensorState()
quadraturedecoder.get_sensorState()
quadraturedecoder.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :

un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

quadraturedecoder→get_speed() YQuadratureDecoder
quadraturedecoder→speed()
quadraturedecoder.get_speed()
quadraturedecoder.get_speed()

Retourne la fréquence des incréments, en Hz.

function **get_speed**( )

Retourne :
une valeur numérique représentant la fréquence des incréments, en Hz

En cas d'erreur, déclenche une exception ou retourne `Y_SPEED_INVALID`.

quadraturedecoder→get_unit()
quadraturedecoder→unit()
quadraturedecoder.get_unit()
quadraturedecoder.get_unit()

Retourne l'unité dans laquelle la position est exprimée.

function **get_unit**( )

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la position est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

quadraturedecoder→get_userData() YQuadratureDecoder
quadraturedecoder→userData()
quadraturedecoder.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.

quadraturedecoder→isOnline() YQuadratureDecoder
quadraturedecoder.isOnline()

Vérifie si le module hébergeant le décodeur de quadrature est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du décodeur de quadrature sont valides au moment de l'appel, le module est considéré joignable. 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 décodeur de quadrature est joignable, `false` sinon

quadraturedecoder→load()quadraturedecoder.load() YQuadratureDecoder

Met en cache les valeurs courantes du décodeur de quadrature, 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 occasionellement 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.

quadraturedecoder→loadAttribute() YQuadratureDecoder
quadraturedecoder.loadAttribute()
quadraturedecoder.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

quadraturedecoder→loadCalibrationPoints() YQuadratureDecoder
quadraturedecoder.loadCalibrationPoints()
quadraturedecoder.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

quadraturedecoder→muteValueCallbacks() YQuadratureDecoder
quadraturedecoder.muteValueCallbacks()
quadraturedecoder.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

quadraturedecoder→nextQuadratureDecoder() YQuadratureDecoder
quadraturedecoder.nextQuadratureDecoder()
quadraturedecoder.nextQuadratureDecoder()

Continue l'énumération des Décodeur de quadrature commencée à l'aide de `yFirstQuadratureDecoder()`.

function **nextQuadratureDecoder**( )

Retourne :

un pointeur sur un objet `YQuadratureDecoder` accessible en ligne, ou `null` lorsque l'énumération est terminée.

quadraturedecoder→registerTimedReportCallback() YQuadratureDecoder
quadraturedecoder.registerTimedReportCallback()
quadraturedecoder.registerTimedReportCallback()

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.

quadraturedecoder→registerValueCallback() YQuadratureDecoder
quadraturedecoder.registerValueCallback()
quadraturedecoder.registerValueCallback()

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.

quadraturedecoder→set_currentValue() YQuadratureDecoder
quadraturedecoder→setCurrentValue()
quadraturedecoder.set_currentValue()
quadraturedecoder.set_currentValue()

Modifie la position actuelle supposée par le décodeur de quadrature.

function **set_currentValue**(**newval**)

L'appel à cette fonction active implitement le décodeur de quadrature.

Paramètres :

newval une valeur numérique représentant la position actuelle supposée par le décodeur de quadrature

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

quadraturedecoder→set_decoding() YQuadratureDecoder
quadraturedecoder→setDecoding()
quadraturedecoder.set_decoding()
quadraturedecoder.set_decoding()

Modifie l'état d'activation du décodeur de quadrature.

function **set_decoding**(**newval**)

Paramètres :

newval soit `Y_DECODING_OFF`, soit `Y_DECODING_ON`, selon l'état d'activation du décodeur de quadrature

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

quadraturedecoder→set_highestValue() YQuadratureDecoder
quadraturedecoder→setHighestValue()
quadraturedecoder.set_highestValue()
quadraturedecoder.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.

quadraturedecoder→set_logFrequency() YQuadratureDecoder
quadraturedecoder→setLogFrequency()
quadraturedecoder.set_logFrequency()
quadraturedecoder.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.

quadraturedecoder→set_logicalName() YQuadratureDecoder
quadraturedecoder→setLogicalName()
quadraturedecoder.set_logicalName()
quadraturedecoder.set_logicalName()

Modifie le nom logique du décodeur de quadrature.

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 décodeur de quadrature.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

quadraturedecoder→set_lowestValue() YQuadratureDecoder
quadraturedecoder→setLowestValue()
quadraturedecoder.set_lowestValue()
quadraturedecoder.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.

quadraturedecoder→set_reportFrequency() YQuadratureDecoder
quadraturedecoder→setReportFrequency()
quadraturedecoder.set_reportFrequency()
quadraturedecoder.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.

quadraturedecoder→set_resolution() YQuadratureDecoder
quadraturedecoder→setResolution()
quadraturedecoder.set_resolution()
quadraturedecoder.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.

quadraturedecoder→set_userData() YQuadratureDecoder
quadraturedecoder→setUserData()
quadraturedecoder.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

quadraturedecoder→startDataLogger() YQuadratureDecoder
quadraturedecoder.startDataLogger()
quadraturedecoder.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

quadraturedecoder→stopDataLogger() YQuadratureDecoder
quadraturedecoder.stopDataLogger()
quadraturedecoder.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

quadraturedecoder→unmuteValueCallbacks() YQuadratureDecoder
quadraturedecoder.unmuteValueCallbacks()
quadraturedecoder.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

quadraturedecoder→wait_async() YQuadratureDecoder
quadraturedecoder.wait_async()

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.

3.54. Interface de la fonction RangeFinder

La classe YRangeFinder permet d'utiliser et de configurer les capteurs de distance Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, elle permet d'effectuer facilement une calibration linéaire à un point pour compenser l'effet d'une vitre ou d'un filtre placé devant le capteur.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_rangefinder.js'></script>
cpp     #include "yocto_rangefinder.h"
m       #import "yocto_rangefinder.h"
pas     uses yocto_rangefinder;
vb      yocto_rangefinder.vb
cs      yocto_rangefinder.cs
java    import com.yoctopuce.YoctoAPI.YRangeFinder;
uwp     import com.yoctopuce.YoctoAPI.YRangeFinder;
py      from yocto_rangefinder import *
php     require_once('yocto_rangefinder.php');
es      in HTML: <script src="../../lib/yocto_rangefinder.js"></script>
        in node.js: require('yoctolib-es2017/yocto_rangefinder.js');
```

Fonction globales	
yFindRangeFinder(**func**)	Permet de retrouver un capteur de distance d'après un identifiant donné.
yFindRangeFinderInContext(**yctx**, **func**)	Permet de retrouver un capteur de distance d'après un identifiant donné dans un Context YAPI.
yFirstRangeFinder()	Commence l'énumération des capteurs de distance accessibles par la librairie.
yFirstRangeFinderInContext(**yctx**)	Commence l'énumération des capteurs de distance accessibles par la librairie.
Méthodes des objets YRangeFinder	
rangefinder→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.
rangefinder→cancelCoverGlassCalibrations()	Annule l'effet des calibrations effectuées précédemment pour compenser une vitre de protection, et rétablit les paramètres d'usine.
rangefinder→clearCache()	Invalide le cache.
rangefinder→describe()	Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de distance au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
rangefinder→get_advertisedValue()	Retourne la valeur courante du capteur de distance (pas plus de 6 caractères).
rangefinder→get_currentRawValue()	Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en mm, sous forme de nombre à virgule.
rangefinder→get_currentTemperature()	Retourne la température actuelle du capteur, sous forme de nombre à virgule.

rangefinder→get_currentValue()

Retourne la valeur actuelle de la distance, en mm, sous forme de nombre à virgule.

rangefinder→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

rangefinder→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de distance.

rangefinder→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de distance.

rangefinder→get_friendlyName()

Retourne un identifiant global du capteur de distance au format `NOM_MODULE.NOM_FONCTION`.

rangefinder→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

rangefinder→get_functionId()

Retourne l'identifiant matériel du capteur de distance, sans référence au module.

rangefinder→get_hardwareCalibrationTemperature()

Retourne la température à laquelle la dernière calibration a été effectuée.

rangefinder→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de distance au format `SERIAL.FUNCTIONID`.

rangefinder→get_highestValue()

Retourne la valeur maximale observée pour la distance depuis le démarrage du module.

rangefinder→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.

rangefinder→get_logicalName()

Retourne le nom logique du capteur de distance.

rangefinder→get_lowestValue()

Retourne la valeur minimale observée pour la distance depuis le démarrage du module.

rangefinder→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

rangefinder→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

rangefinder→get_rangeFinderMode()

Retourne le mode de fonctionnement du capteur de distance.

rangefinder→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.

rangefinder→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

rangefinder→get_resolution()

Retourne la résolution des valeurs mesurées.

rangefinder→get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

rangefinder→get_unit()

Retourne l'unité dans laquelle la distance est exprimée.

rangefinder→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

rangefinder→isOnline()

Vérifie si le module hébergeant le capteur de distance est joignable, sans déclencher d'erreur.

rangefinder→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le capteur de distance est joignable, sans déclencher d'erreur.

rangefinder→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

rangefinder→load(**msValidity**)

Met en cache les valeurs courantes du capteur de distance, avec une durée de validité spécifiée.

rangefinder→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

rangefinder→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

rangefinder→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du capteur de distance, avec une durée de validité spécifiée.

rangefinder→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

rangefinder→nextRangeFinder()

Continue l'énumération des capteurs de distance commencée à l'aide de `yFirstRangeFinder()`.

rangefinder→registerTimedReportCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

rangefinder→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

rangefinder→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

rangefinder→set_logFrequency(**newval**)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

rangefinder→set_logicalName(**newval**)

Modifie le nom logique du capteur de distance.

rangefinder→set_lowestValue(**newval**)

Modifie la mémoire de valeur minimale observée.

rangefinder→set_rangeFinderMode(**newval**)

Modifie le mode de fonctionnement du capteur de distance, permettant ainsi de mettre la priorité sur la précision, la vitesse ou la distance maximale détectable.

rangefinder→set_reportFrequency(**newval**)

Modifie la fréquence de notification périodique des valeurs mesurées.

rangefinder→set_resolution(**newval**)

Modifie la résolution des valeurs physique mesurées.

rangefinder→set_unit(**newval**)

Change l'unité dans laquelle la distance mesurée est exprimée.

rangefinder→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`.

rangefinder→startDataLogger()

Démarre l'enregistreur de données du module.

rangefinder→stopDataLogger()

Arrête l'enregistreur de données du module.

rangefinder→triggerOffsetCalibration(**targetDist**)

Lance la calibration matérielle d'offset du capteur de distance.

rangefinder→triggerSpadCalibration()

Lance la calibration des détecteurs de photons.

rangefinder→triggerTemperatureCalibration()

Lance une calibration par rapport à la température ambiante.

rangefinder→triggerXTalkCalibration(**targetDist**)

Lance la calibration matérielle d'interférence de réflexion du capteur de distance.

rangefinder→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

rangefinder→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.

YRangeFinder.FindRangeFinder() YRangeFinder
yFindRangeFinder()
YRangeFinder.FindRangeFinder()
YRangeFinder.FindRangeFinder()

Permet de retrouver un capteur de distance d'après un identifiant donné.

```
function FindRangeFinder( 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 distance soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRangeFinder.isOnline()` pour tester si le capteur de distance 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le capteur de distance sans ambiguïté

Retourne :

un objet de classe `YRangeFinder` qui permet ensuite de contrôler le capteur de distance.

YRangeFinder.FindRangeFinderInContext() YRangeFinder yFindRangeFinderInContext() YRangeFinder.FindRangeFinderInContext() YRangeFinder.FindRangeFinderInContext()

Permet de retrouver un capteur de distance d'après un identifiant donné dans un Context YAPI.

function **FindRangeFinderInContext**(**yctx**, **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 distance soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRangeFinder.isOnline()` pour tester si le capteur de distance 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de distance sans ambiguïté

Retourne :

un objet de classe `YRangeFinder` qui permet ensuite de contrôler le capteur de distance.

YRangeFinder.FirstRangeFinder() YRangeFinder
yFirstRangeFinder()
YRangeFinder.FirstRangeFinder()
YRangeFinder.FirstRangeFinder()

Commence l'énumération des capteurs de distance accessibles par la librairie.

function **FirstRangeFinder**( )

Utiliser la fonction `YRangeFinder.nextRangeFinder()` pour itérer sur les autres capteurs de distance.

Retourne :

un pointeur sur un objet `YRangeFinder`, correspondant au premier capteur de distance accessible en ligne, ou `null` si il n'y a pas de capteurs de distance disponibles.

YRangeFinder.FirstRangeFinderInContext() YRangeFinder yFirstRangeFinderInContext() YRangeFinder.FirstRangeFinderInContext() YRangeFinder.FirstRangeFinderInContext()

Commence l'énumération des capteurs de distance accessibles par la librairie.

function **FirstRangeFinderInContext**(**yctx**)

Utiliser la fonction `YRangeFinder.nextRangeFinder()` pour itérer sur les autres capteurs de distance.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YRangeFinder`, correspondant au premier capteur de distance accessible en ligne, ou `null` si il n'y a pas de capteurs de distance disponibles.

rangefinder→calibrateFromPoints() YRangeFinder
rangefinder.calibrateFromPoints()
rangefinder.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

rangefinder→cancelCoverGlassCalibrations() YRangeFinder
rangefinder.cancelCoverGlassCalibrations()
rangefinder.cancelCoverGlassCalibrations()

Annule l'effet des calibrations effectuées précédemment pour compenser une vitre de protection, et rétablit les paramètres d'usine.

function **cancelCoverGlassCalibrations**( )

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

rangefinder→clearCache()rangefinder.clearCache() YRangeFinder

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur de distance. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

rangefinder→describe()rangefinder.describe() YRangeFinder

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de distance au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le capteur de distance (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

rangefinder→get_advertisedValue() YRangeFinder
rangefinder→advertisedValue()
rangefinder.get_advertisedValue()
rangefinder.get_advertisedValue()

Retourne la valeur courante du capteur de distance (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante du capteur de distance (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

rangefinder→get_currentRawValue() YRangeFinder
rangefinder→currentRawValue()
rangefinder.get_currentRawValue()
rangefinder.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en mm, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en mm, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

rangefinder→get_currentTemperature() YRangeFinder
rangefinder→currentTemperature()
rangefinder.get_currentTemperature()
rangefinder.get_currentTemperature()

Retourne la température actuelle du capteur, sous forme de nombre à virgule.

function **get_currentTemperature**( )

Retourne :
une valeur numérique représentant la température actuelle du capteur, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTTEMPERATURE_INVALID`.

rangefinder→get_currentValue() YRangeFinder
rangefinder→currentValue()
rangefinder.get_currentValue()
rangefinder.get_currentValue()

Retourne la valeur actuelle de la distance, en mm, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle de la distance, en mm, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

rangefinder→get_dataLogger() YRangeFinder
rangefinder→dataLogger()
rangefinder.get_dataLogger()
rangefinder.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

rangefinder→get_errorMessage() YRangeFinder
rangefinder→errorMessage()
rangefinder.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de distance.

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 distance.

rangefinder→get_errorType() YRangeFinder
rangefinder→errorType()rangefinder.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de distance.

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 distance.

rangefinder→get_friendlyName() YRangeFinder
rangefinder→friendlyName()
rangefinder.get_friendlyName()

Retourne un identifiant global du capteur de distance au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du capteur de distance si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du capteur de distance (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le capteur de distance en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

rangefinder→get_functionDescriptor() YRangeFinder
rangefinder→functionDescriptor()
rangefinder.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

rangefinder→get_functionId() YRangeFinder
rangefinder→functionId()rangefinder.get_functionId()

Retourne l'identifiant matériel du capteur de distance, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de distance (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

rangefinder→get_hardwareCalibrationTemperature() YRangeFinder
rangefinder→hardwareCalibrationTemperature()
rangefinder.get_hardwareCalibrationTemperature()
rangefinder.get_hardwareCalibrationTemperature()

Retourne la température à laquelle la dernière calibration a été effectuée.

function **get_hardwareCalibrationTemperature**( )

Cette fonction permet de déterminer si une nouvelle calibration par rapport à la température ambiante est nécessaire.

Retourne :

une température, sous forme de nombre à virgule. En cas d'erreur, déclenche une exception ou retourne YAPI_INVALID_DOUBLE.

rangefinder→get_hardwareId() YRangeFinder
rangefinder→hardwareId()
rangefinder.get_hardwareId()

Retourne l'identifiant matériel unique du capteur de distance 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 distance (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le capteur de distance (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

rangefinder→get_highestValue() YRangeFinder
rangefinder→highestValue()
rangefinder.get_highestValue()
rangefinder.get_highestValue()

Retourne la valeur maximale observée pour la distance depuis le démarrage du module.

function **get_highestValue**( )

Retourne :
une valeur numérique représentant la valeur maximale observée pour la distance depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

rangefinder→get_logFrequency() YRangeFinder
rangefinder→logFrequency()
rangefinder.get_logFrequency()
rangefinder.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`.

rangefinder→get_logicalName() YRangeFinder
rangefinder→logicalName()
rangefinder.get_logicalName()
rangefinder.get_logicalName()

Retourne le nom logique du capteur de distance.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du capteur de distance.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

rangefinder→get_lowestValue() YRangeFinder
rangefinder→lowestValue()
rangefinder.get_lowestValue()
rangefinder.get_lowestValue()

Retourne la valeur minimale observée pour la distance depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :

une valeur numérique représentant la valeur minimale observée pour la distance depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

rangefinder→get_module() YRangeFinder
rangefinder→module()rangefinder.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

rangefinder→get_rangeFinderMode() YRangeFinder
rangefinder→rangeFinderMode()
rangefinder.get_rangeFinderMode()
rangefinder.get_rangeFinderMode()

Retourne le mode de fonctionnement du capteur de distance.

function **get_rangeFinderMode**( )

Le choix du mode permet de favoriser la précision, la vitesse, ou la distance maximale détectable.

Retourne :

une valeur parmi `Y_RANGEFINDERMODE_DEFAULT`, `Y_RANGEFINDERMODE_LONG_RANGE`, `Y_RANGEFINDERMODE_HIGH_ACCURACY` et `Y_RANGEFINDERMODE_HIGH_SPEED` représentant le mode de fonctionnement du capteur de distance

En cas d'erreur, déclenche une exception ou retourne `Y_RANGEFINDERMODE_INVALID`.

rangefinder→get_recordedData() YRangeFinder
rangefinder→recordedData()
rangefinder.get_recordedData()
rangefinder.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 find de l'intercalle 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.

rangefinder→get_reportFrequency() YRangeFinder
rangefinder→reportFrequency()
rangefinder.get_reportFrequency()
rangefinder.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`.

rangefinder→get_resolution() YRangeFinder
rangefinder→resolution()rangefinder.get_resolution()
rangefinder.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`.

rangefinder→get_sensorState() YRangeFinder
rangefinder→sensorState()
rangefinder.get_sensorState()
rangefinder.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :

un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

rangefinder→get_unit() YRangeFinder
rangefinder→unit()rangefinder.get_unit()
rangefinder.get_unit()

Retourne l'unité dans laquelle la distance est exprimée.

function **get_unit**( )

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la distance est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

rangefinder→get_userData() YRangeFinder
rangefinder→userData()rangefinder.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.

rangefinder→isOnline()rangefinder.isOnline() YRangeFinder

Vérifie si le module hébergeant le capteur de distance est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du capteur de distance sont valides au moment de l'appel, le module est considéré joignable. 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 distance est joignable, `false` sinon

rangefinder→load()rangefinder.load() YRangeFinder

Met en cache les valeurs courantes du capteur de distance, 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 occasionellement 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.

rangefinder→loadAttribute() YRangeFinder
rangefinder.loadAttribute()rangefinder.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

rangefinder→loadCalibrationPoints() YRangeFinder
rangefinder.loadCalibrationPoints()
rangefinder.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

rangefinder→muteValueCallbacks() YRangeFinder
rangefinder.muteValueCallbacks()
rangefinder.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

rangefinder→nextRangeFinder() YRangeFinder
rangefinder.nextRangeFinder()
rangefinder.nextRangeFinder()

Continue l'énumération des capteurs de distance commencée à l'aide de `yFirstRangeFinder()`.

function **nextRangeFinder**( )

Retourne :

un pointeur sur un objet `YRangeFinder` accessible en ligne, ou `null` lorsque l'énumération est terminée.

rangefinder→registerTimedReportCallback() YRangeFinder
rangefinder.registerTimedReportCallback()
rangefinder.registerTimedReportCallback()

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.

rangefinder→registerValueCallback() YRangeFinder
rangefinder.registerValueCallback()
rangefinder.registerValueCallback()

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.

rangefinder→set_highestValue() YRangeFinder
rangefinder→setHighestValue()
rangefinder.set_highestValue()
rangefinder.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.

rangefinder→set_logFrequency() YRangeFinder
rangefinder→setLogFrequency()
rangefinder.set_logFrequency()
rangefinder.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.

rangefinder→set_logicalName() YRangeFinder
rangefinder→setLogicalName()
rangefinder.set_logicalName()
rangefinder.set_logicalName()

Modifie le nom logique du capteur de distance.

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 distance.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

rangefinder→set_lowestValue() YRangeFinder
rangefinder→setLowestValue()
rangefinder.set_lowestValue()
rangefinder.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.

rangefinder→set_rangeFinderMode() YRangeFinder
rangefinder→setRangeFinderMode()
rangefinder.set_rangeFinderMode()
rangefinder.set_rangeFinderMode()

Modifie le mode de fonctionnement du capteur de distance, permettant ainsi de mettre la priorité sur la précision, la vitesse ou la distance maximale détectable.

function **set_rangeFinderMode**(**newval**)

Paramètres :

newval une valeur parmi Y_RANGEFINDERMODE_DEFAULT, Y_RANGEFINDERMODE_LONG_RANGE, Y_RANGEFINDERMODE_HIGH_ACCURACY et Y_RANGEFINDERMODE_HIGH_SPEED représentant le mode de fonctionnement du capteur de distance, permettant ainsi de mettre la priorité sur la précision, la vitesse ou la distance maximale détectable

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

rangefinder→set_reportFrequency() YRangeFinder
rangefinder→setReportFrequency()
rangefinder.set_reportFrequency()
rangefinder.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.

rangefinder→set_resolution() YRangeFinder
rangefinder→setResolution()
rangefinder.set_resolution()
rangefinder.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.

rangefinder→set_unit() YRangeFinder
rangefinder→setUnit()rangefinder.set_unit()
rangefinder.set_unit()

Change l'unité dans laquelle la distance mesurée est exprimée.

```
function set_unit( newval)
```

Cette unité est une chaîne de charactère qui peut être `"` ou `mm`. Toute autre valeur sera ignorée. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé. Attention: si une calibration spécifique est définie pour la fonction rangeFinder un changement d'unité a toutes les chances de la fausser.

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.

rangefinder→set_userData() YRangeFinder
rangefinder→setUserData()
rangefinder.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

rangefinder→startDataLogger() YRangeFinder
rangefinder.startDataLogger()
rangefinder.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

rangefinder→stopDataLogger() YRangeFinder
rangefinder.stopDataLogger()
rangefinder.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

rangefinder→triggerOffsetCalibration() YRangeFinder
rangefinder.triggerOffsetCalibration()
rangefinder.triggerOffsetCalibration()

Lance la calibration matérielle d'offset du capteur de distance.

function **triggerOffsetCalibration**(**targetDist**)

Cette fonction fait partie de la procédure de calibration pour tenir compte de la présence d'une vitre de protection devant le capteur. Référez-vous au chapitre de la documentation concernant la calibration pour compenser l'effet d'une vitre pour les détails sur la marche à suivre exacte.

Paramètres :

targetDist distance réelle de la cible, en mm ou en pouce selon l'unité configurée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

rangefinder→triggerSpadCalibration() YRangeFinder
rangefinder.triggerSpadCalibration()
rangefinder.triggerSpadCalibration()

Lance la calibration des détecteurs de photons.

function **triggerSpadCalibration**( )

Cette fonction fait partie de la procédure de calibration pour tenir compte de la présence d'une vitre de protection devant le capteur. Référez-vous au chapitre de la documentation concernant la calibration pour compenser l'effet d'une vitre pour les détails sur la marche à suivre exacte.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

rangefinder→triggerTemperatureCalibration() YRangeFinder
rangefinder.triggerTemperatureCalibration()
rangefinder.triggerTemperatureCalibration()

Lance une calibration par rapport à la température ambiante.

function **triggerTemperatureCalibration**( )

Cette calibration ne nécessite aucune interaction physique avec le capteur. Elle est effectuée automatiquement au démarrage du module, mais il est recommandé de la relancer si la variation de température depuis la dernière calibration dépasse 8°C.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

rangefinder→triggerXTalkCalibration() YRangeFinder
rangefinder.triggerXTalkCalibration()
rangefinder.triggerXTalkCalibration()

Lance la calibration matérielle d'interférence de réflexion du capteur de distance.

function **triggerXTalkCalibration**(**targetDist**)

Cette fonction fait partie de la procédure de calibration pour tenir compte de la présence d'une vitre de protection devant le capteur. Référez-vous au chapitre de la documentation concernant la calibration pour compenser l'effet d'une vitre pour les détails sur la marche à suivre exacte.

Paramètres :

targetDist distance réelle de la cible, en mm ou en pouce selon l'unité configurée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

rangefinder→unmuteValueCallbacks() YRangeFinder
rangefinder.unmuteValueCallbacks()
rangefinder.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

rangefinder→wait_async()rangefinder.wait_async() YRangeFinder

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.

3.55. Interface de la fonction Horloge Temps Real

La fonction RealTimeClock fourni 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 fait au changement d'heure été/hiver.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_realtimeclock.js'></script>
cpp	#include "yocto_realtimeclock.h"
m	#import "yocto_realtimeclock.h"
pas	uses yocto_realtimeclock;
vb	yocto_realtimeclock.vb
cs	yocto_realtimeclock.cs
java	import com.yoctopuce.YoctoAPI.YRealTimeClock;
uwp	import com.yoctopuce.YoctoAPI.YRealTimeClock;
py	from yocto_realtimeclock import *
php	require_once('yocto_realtimeclock.php');
es	in HTML: <script src="../../lib/yocto_realtimeclock.js"></script> in node.js: require('yoctolib-es2017/yocto_realtimeclock.js');

Fonction globales

yFindRealTimeClock(**func**)

Permet de retrouver une horloge d'après un identifiant donné.

yFindRealTimeClockInContext(**yctx**, **func**)

Permet de retrouver une horloge d'après un identifiant donné dans un Context YAPI.

yFirstRealTimeClock()

Commence l'énumération des horloge accessibles par la librairie.

yFirstRealTimeClockInContext(**yctx**)

Commence l'énumération des horloge accessibles par la librairie.

Méthodes des objets `YRealTimeClock`

realtimeclock→clearCache()

Invalide le cache.

realtimeclock→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'horloge au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

realtimeclock→get_advertisedValue()

Retourne la valeur courante de l'horloge (pas plus de 6 caractères).

realtimeclock→get_dateTime()

Retourne l'heure courante au format "AAAA/MM/JJ hh:mm:ss".

realtimeclock→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

realtimeclock→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

realtimeclock→get_friendlyName()

Retourne un identifiant global de l'horloge au format `NOM_MODULE.NOM_FONCTION`.

realtimeclock→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

realtimeclock→get_functionId()

Retourne l'identifiant matériel de l'horloge, sans référence au module.

realtimeclock→get_hardwareId()

Retourne l'identifiant matériel unique de l'horloge au format `SERIAL.FUNCTIONID`.

realtimeclock→get_logicalName()

Retourne le nom logique de l'horloge.

realtimeclock→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

realtimeclock→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

realtimeclock→get_timeSet()

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

realtimeclock→get_unixTime()

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

realtimeclock→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

realtimeclock→get_utcOffset()

Retourne le nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone).

realtimeclock→isOnline()

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

realtimeclock→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

realtimeclock→load(**msValidity**)

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

realtimeclock→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

realtimeclock→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

realtimeclock→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

realtimeclock→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YRealTimeClock.FindRealTimeClock() YRealTimeClock
yFindRealTimeClock()
YRealTimeClock.FindRealTimeClock()
YRealTimeClock.FindRealTimeClock()

Permet de retrouver une horloge d'après un identifiant donné.

function **FindRealTimeClock**(**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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindRealTimeClockInContext() YRealTimeClock yFindRealTimeClockInContext() YRealTimeClock.FindRealTimeClockInContext() YRealTimeClock.FindRealTimeClockInContext()

Permet de retrouver une horloge d'après un identifiant donné dans un Context YAPI.

function **FindRealTimeClockInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'horloge sans ambiguïté

Retourne :

un objet de classe `YRealTimeClock` qui permet ensuite de contrôler l'horloge.

YRealTimeClock.FirstRealTimeClock() YRealTimeClock
yFirstRealTimeClock()
YRealTimeClock.FirstRealTimeClock()
YRealTimeClock.FirstRealTimeClock()

Commence l'énumération des horloge accessibles par la librairie.

function **FirstRealTimeClock**( )

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.

YRealTimeClock.FirstRealTimeClockInContext() YRealTimeClock
yFirstRealTimeClockInContext()
YRealTimeClock.FirstRealTimeClockInContext()
YRealTimeClock.FirstRealTimeClockInContext()

Commence l'énumération des horloge accessibles par la librairie.

function **FirstRealTimeClockInContext**(**yctx**)

Utiliser la fonction `YRealTimeClock.nextRealTimeClock()` pour itérer sur les autres horloge.

Paramètres :

yctx un contexte YAPI.

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→clearCache() YRealTimeClock
realtimeclock.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'horloge. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant l'horloge (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

realtimeclock→get_advertisedValue() YRealTimeClock
realtimeclock→advertisedValue()
realtimeclock.get_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()
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'identifant 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() YRealTimeClock
realtimeclock→functionDescriptor()
realtimeclock.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

realtimeclock→get_functionId() YRealTimeClock
realtimeclock→functionId()
realtimeclock.get_functionId()

Retourne l'identifiant matériel de l'horloge, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant l'horloge (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

realtimeclock→get_hardwareId() YRealTimeClock
realtimeclock→hardwareId()
realtimeclock.get_hardwareId()

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 example 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()
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_timeSet() YRealTimeClock
realtimeclock→timeSet()realtimeclock.get_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()
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'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.

realtimeclock→get_utcOffset() YRealTimeClock
realtimeclock→utcOffset()
realtimeclock.get_utcOffset()
realtimeclock.get_utcOffset()

Retourne le nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone).

function **get_utcOffset**( )

Retourne :

un entier représentant le nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone)

En cas d'erreur, déclenche une exception ou retourne `Y_UTCOFFSET_INVALID`.

realtimeclock→isOnline()realtimeclock.isOnline() YRealTimeClock

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache de l'horloge sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si l'horloge est joignable, `false` sinon

realtimeclock→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 occasionellement 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→loadAttribute() YRealTimeClock
realtimeclock.loadAttribute()
realtimeclock.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

realtimeclock→muteValueCallbacks() YRealTimeClock
realtimeclock.muteValueCallbacks()
realtimeclock.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

realtimeclock→nextRealTimeClock() YRealTimeClock
realtimeclock.nextRealTimeClock()
realtimeclock.nextRealTimeClock()

Continue l'énumération des horloge commencée à l'aide de `yFirstRealTimeClock()`.

function **nextRealTimeClock**( )

Retourne :

un pointeur sur un objet `YRealTimeClock` accessible en ligne, ou `null` lorsque l'énumération est terminée.

realtimeclock→registerValueCallback() YRealTimeClock
realtimeclock.registerValueCallback()
realtimeclock.registerValueCallback()

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() YRealTimeClock
realtimeclock→setLogicalName()
realtimeclock.set_logicalName()
realtimeclock.set_logicalName()

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() YRealTimeClock
realtimeclock→setUnixTime()
realtimeclock.set_unixTime()
realtimeclock.set_unixTime()

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).

Paramètres :

newval un entier représentant l'heure courante

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

realtimeclock→set_userData() YRealTimeClock
realtimeclock→setUserData()
realtimeclock.set_userData()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

realtimeclock→set_utcOffset() YRealTimeClock
realtimeclock→setUtcOffset()
realtimeclock.set_utcOffset()
realtimeclock.set_utcOffset()

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

function **set_utcOffset**(**newval**)

Le décallage est automatiquement arrondi au quart d'heure le plus proche.

Paramètres :

newval un entier représentant le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone)

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

realtimeclock→unmuteValueCallbacks() YRealTimeClock
realtimeclock.unmuteValueCallbacks()
realtimeclock.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

realtimeclock→wait_async() YRealTimeClock
realtimeclock.wait_async()

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.

3.56. 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>
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;
uwp	import com.yoctopuce.YoctoAPI.YRefFrame;
py	from yocto_refframe import *
php	require_once('yocto_refframe.php');
es	in HTML: <script src="../../lib/yocto_refframe.js"></script> in node.js: require('yoctolib-es2017/yocto_refframe.js');

Fonction globales

yFindRefFrame(**func**)

Permet de retrouver un référentiel d'après un identifiant donné.

yFindRefFrameInContext(**yctx**, **func**)

Permet de retrouver un référentiel d'après un identifiant donné dans un Context YAPI.

yFirstRefFrame()

Commence l'énumération des référentiels accessibles par la librairie.

yFirstRefFrameInContext(**yctx**)

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→clearCache()

Invalide le cache.

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`.

reframe→get_3DCalibrationStageProgress()

Retourne l'avancement de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.

reframe→get_advertisedValue()

Retourne la valeur courante du référentiel (pas plus de 6 caractères).

reframe→get_bearing()

Retourne le cap de référence utilisé par le compas.

reframe→get_calibrationState()

Retourne l'état de calibration des capteurs 3D (Yocto-3D-V2 seulement).

reframe→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

reframe→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

reframe→get_friendlyName()

Retourne un identifiant global du référentiel au format `NOM_MODULE.NOM_FONCTION`.

reframe→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

reframe→get_functionId()

Retourne l'identifiant matériel du référentiel, sans référence au module.

reframe→get_hardwareId()

Retourne l'identifiant matériel unique du référentiel au format `SERIAL.FUNCTIONID`.

reframe→get_logicalName()

Retourne le nom logique du référentiel.

reframe→get_measureQuality()

Retourne l'estimation de qualité de la mesure d'orientation (Yocto-3D-V2 seulement).

reframe→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

reframe→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

reframe→get_mountOrientation()

Retourne l'orientation à l'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

reframe→get_mountPosition()

Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

reframe→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

reframe→isOnline()

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

reframe→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

reframe→load(**msValidity**)

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

reframe→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

reframe→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

reframe→more3DCalibration()

Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode `start3DCalibration`.

reframe→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

reframe→nextRefFrame()

Continue l'énumération des référentiels commencée à l'aide de `yFirstRefFrame()`.

reframe→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

reframe→save3DCalibration()

Applique les paramètres de calibration tridimensionnelle précédemment calculés.

reframe→set_bearing(newval)

Modifie le cap de référence utilisé par le compas.

reframe→set_logicalName(newval)

Modifie le nom logique du référentiel.

reframe→set_mountPosition(position, orientation)

Modifie le référentiel de la boussole et des inclinomètres.

reframe→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`.

reframe→start3DCalibration()

Initie le processus de calibration tridimensionnelle des capteurs.

reframe→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

reframe→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YRefFrame.FindRefFrame() YRefFrame yFindRefFrame()YRefFrame.FindRefFrame() YRefFrame.FindRefFrame()

Permet de retrouver un référentiel d'après un identifiant donné.

```
function FindRefFrame( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindRefFrameInContext() YRefFrame
yFindRefFrameInContext()
YRefFrame.FindRefFrameInContext()
YRefFrame.FindRefFrameInContext()

Permet de retrouver un référentiel d'après un identifiant donné dans un Context YAPI.

function **FindRefFrameInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le référentiel sans ambiguïté

Retourne :

un objet de classe `YRefFrame` qui permet ensuite de contrôler le référentiel.

YRefFrame.FirstRefFrame() YRefFrame yFirstRefFrame()YRefFrame.FirstRefFrame() YRefFrame.FirstRefFrame()

Commence l'énumération des référentiels accessibles par la librairie.

function **FirstRefFrame**( )

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.

YRefFrame.FirstRefFrameInContext() YRefFrame
yFirstRefFrameInContext()
YRefFrame.FirstRefFrameInContext()
YRefFrame.FirstRefFrameInContext()

Commence l'énumération des référentiels accessibles par la librairie.

function **FirstRefFrameInContext**(**yctx**)

Utiliser la fonction `YRefFrame.nextRefFrame()` pour itérer sur les autres référentiels.

Paramètres :

yctx un contexte YAPI.

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() YRefFrame
refframe.cancel3DCalibration()
refframe.cancel3DCalibration()

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→clearCache()refframe.clearCache() YRefFrame

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du référentiel. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

reframe→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 utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le référentiel (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

refframe→get_3DCalibrationHint() YRefFrame
refframe→3DCalibrationHint()
refframe.get_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() YRefFrame
refframe→3DCalibrationLogMsg()
refframe.get_3DCalibrationLogMsg()
refframe.get_3DCalibrationLogMsg()

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()
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()
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()
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()
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()
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_calibrationState() YRefFrame
refframe→calibrationState()
refframe.get_calibrationState()
refframe.get_calibrationState()

Retourne l'état de calibration des capteurs 3D (Yocto-3D-V2 seulement).

function **get_calibrationState**( )

Cette fonction retourne un entier représentant l'état de calibration des trois capteurs inertiels du chip BNO055, présent dans le Yocto-3D-V2. Les centaines indiquent l'état de calibration de l'accéléromètre, les dizaines indiquent l'état de calibration du magnétomètre, et les unités indiquent l'état de calibration du gyroscope. Pour chaque capteur, la valeur 0 indique l'absence de calibration tandis que la valeur 3 indique une calibration complète.

Retourne :

un entier représentant l'état de calibration du Yocto-3D-V2: 333 quand il est entièrement calibré, 0 lorsqu'il n'est pas calibré du tout.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif. Pour le Yocto-3D (V1), la valeur retournée est toujours -3 (fonction non supportée).

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.

reframe→get_friendlyName() YRefFrame
reframe→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'identifant 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() YRefFrame
refframe→functionDescriptor()
refframe.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

refframe→get_functionId() YRefFrame
refframe→functionId()refframe.get_functionId()

Retourne l'identifiant matériel du référentiel, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le référentiel (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

reframe→get_hardwareId() YRefFrame
reframe→hardwareId()reframe.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 example 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()
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_measureQuality() YRefFrame
refframe→measureQuality()
refframe.get_measureQuality()
refframe.get_measureQuality()

Retourne l'estimation de qualité de la mesure d'orientation (Yocto-3D-V2 seulement).

function **get_measureQuality**( )

Cette fonction retourne un entier entre 0 et 3 représentant le degré de confiance de l'estimation de position. Lorsque la valeur est 3, l'estimation est fiable. En dessous de 3, il faut s'attendre à des corrections d'orientations ultérieures, en particulier sur la boussole (fonction `compass`). Les causes les plus fréquentes pour une qualité inférieure à 3 sont les interférences magnétiques et les accélérations our rotations en delà des capacités du capteur.

Retourne :
un entier entre 0 et 3 (3 quand la mesure est fiable)

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif. Pour le Yocto-3D (V1), la valeur retournée est toujours -3 (fonction non supportée).

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_mountOrientation() YRefFrame
refframe→mountOrientation()
refframe.get_mountOrientation()
refframe.get_mountOrientation()

Retourne l'orientation à l'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

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()
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

reframe→load()reframe.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 occasionellement 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→loadAttribute()refframe.loadAttribute() YRefFrame refframe.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

refframe→more3DCalibration() YRefFrame
refframe.more3DCalibration()
refframe.more3DCalibration()

Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode `start3DCalibration`.

function **more3DCalibration**( )

Cette méthode doit être appelée environ 5 fois par secondes après avoir positionné le module selon les instructions fournies par la méthode `get_3DCalibrationHint` (les instructions changent pendant la procédure de calibration).

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

reframe→muteValueCallbacks() YRefFrame
reframe.muteValueCallbacks()
reframe.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→nextRefFrame()refframe.nextRefFrame() YRefFrame refframe.nextRefFrame()

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() YRefFrame
refframe.registerValueCallback()
refframe.registerValueCallback()

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() YRefFrame
refframe.save3DCalibration()
refframe.save3DCalibration()

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()
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()
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()
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.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→set_userData() YRefFrame
refframe→setUserData()reframe.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

refframe→start3DCalibration() YRefFrame
refframe.start3DCalibration()
refframe.start3DCalibration()

Initie le processus de calibration tridimensionnelle des capteurs.

function **start3DCalibration**( )

Cette calibration est utilisée à bas niveau pour l'estimation innertielle de position et pour améliorer la précision des mesures d'inclinaison.

Après avoir appelé cette méthode, il faut positionner le module selon les instructions fournies par la méthode `get_3DCalibrationHint` et appeler `more3DCalibration` environ 5 fois par secondes. La procédure de calibration est terminée lorsque la méthode `get_3DCalibrationProgress` retourne 100. Il est alors possible d'appliquer les paramètres calculés, à l'aide de la méthode `save3DCalibration`. A tout moment, la calibration peut être abandonnée à l'aide de `cancel3DCalibration`.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→unmuteValueCallbacks() YRefFrame
refframe.unmuteValueCallbacks()
refframe.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.57. 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 a 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>
cpp	#include "yocto_relay.h"
m	#import "yocto_relay.h"
pas	uses yocto_relay;
vb	yocto_relay.vb
cs	yocto_relay.cs
java	import com.yoctopuce.YoctoAPI.YRelay;
uwp	import com.yoctopuce.YoctoAPI.YRelay;
py	from yocto_relay import *
php	require_once('yocto_relay.php');
es	in HTML: <script src="../../lib/yocto_relay.js"></script> in node.js: require('yoctolib-es2017/yocto_relay.js');

Fonction globales

yFindRelay(**func**)

Permet de retrouver un relais d'après un identifiant donné.

yFindRelayInContext(**yctx**, **func**)

Permet de retrouver un relais d'après un identifiant donné dans un Context YAPI.

yFirstRelay()

Commence l'énumération des relais accessibles par la librairie.

yFirstRelayInContext(**yctx**)

Commence l'énumération des relais accessibles par la librairie.

Méthodes des objets YRelay

relay→clearCache()

Invalide le cache.

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.

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→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

relay→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

relay→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

relay→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YRelay.FindRelay() YRelay yFindRelay()YRelay.FindRelay()YRelay.FindRelay()

Permet de retrouver un relais d'après un identifiant donné.

```
function FindRelay( func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le relais soit en ligne au moment ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindRelayInContext() YRelay yFindRelayInContext()YRelay.FindRelayInContext() YRelay.FindRelayInContext()

Permet de retrouver un relais d'après un identifiant donné dans un Context YAPI.

```
function FindRelayInContext( yctx, 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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le relais sans ambiguïté

Retourne :

un objet de classe `YRelay` qui permet ensuite de contrôler le relais.

YRelay.FirstRelay() YRelay yFirstRelay()YRelay.FirstRelay()YRelay.FirstRelay()

Commence l'énumération des relais accessibles par la librairie.

function **FirstRelay**( )

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.

YRelay.FirstRelayInContext() YRelay yFirstRelayInContext()YRelay.FirstRelayInContext() YRelay.FirstRelayInContext()

Commence l'énumération des relais accessibles par la librairie.

function **FirstRelayInContext**(**yctx**)

Utiliser la fonction `YRelay.nextRelay()` pour itérer sur les autres relais.

Paramètres :

yctx un contexte YAPI.

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→clearCache()relay.clearCache() YRelay

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du relais. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

Retourne :

une chaîne de caractères décrivant le relais (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

relay→get_advertisedValue() YRelay
relay→advertisedValue()relay.get_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()
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'identifant 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 example `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 example 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()
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()
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()
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() YRelay
relay→module()relay.get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

function **get_module**( )

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :
une instance de YModule

relay→get_output() YRelay
relay→output()relay.get_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()
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()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()
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.

function **isOnline**( )

Si les valeurs des attributs en cache du relais sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si le relais est joignable, `false` sinon

relay→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 occasionellement 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→loadAttribute()relay.loadAttribute() YRelay relay.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

relay→muteValueCallbacks() YRelay
relay.muteValueCallbacks()
relay.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→nextRelay()relay.nextRelay()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()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() YRelay
relay.registerValueCallback()
relay.registerValueCallback()

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()
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()
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()
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()
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()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()
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→unmuteValueCallbacks() YRelay
relay.unmuteValueCallbacks()
relay.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.58. Interface de la fonction SegmentedDisplay

La classe SegmentedDisplay permet de gérer des afficheurs à segments

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_segmenteddisplay.js'></script>
cpp	#include "yocto_segmenteddisplay.h"
m	#import "yocto_segmenteddisplay.h"
pas	uses yocto_segmenteddisplay;
vb	yocto_segmenteddisplay.vb
cs	yocto_segmenteddisplay.cs
java	import com.yoctopuce.YoctoAPI.YSegmentedDisplay;
uwp	import com.yoctopuce.YoctoAPI.YSegmentedDisplay;
py	from yocto_segmenteddisplay import *
php	require_once('yocto_segmenteddisplay.php');
es	in HTML: <script src="../../lib/yocto_segmenteddisplay.js"></script> in node.js: require('yoctolib-es2017/yocto_segmenteddisplay.js');

Fonction globales

yFindSegmentedDisplay(**func**)

Permet de retrouver un afficheur d'après un identifiant donné.

yFindSegmentedDisplayInContext(**yctx**, **func**)

Permet de retrouver un afficheur d'après un identifiant donné dans un Context YAPI.

yFirstSegmentedDisplay()

Commence l'énumération des un afficheur accessibles par la librairie.

yFirstSegmentedDisplayInContext(**yctx**)

Commence l'énumération des un afficheur accessibles par la librairie.

Méthodes des objets YSegmentedDisplay

segmenteddisplay→clearCache()

Invalide le cache.

segmenteddisplay→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'afficheur au format TYPE(NAME)=SERIAL.FUNCTIONID.

segmenteddisplay→get_advertisedValue()

Retourne la valeur courante de l'afficheur (pas plus de 6 caractères).

segmenteddisplay→get_displayedText()

Retourne le texte actuellement affiché à l'écran.

segmenteddisplay→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'afficheur.

segmenteddisplay→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'afficheur.

segmenteddisplay→get_friendlyName()

Retourne un identifiant global de l'afficheur au format NOM_MODULE.NOM_FONCTION.

segmenteddisplay→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

segmenteddisplay→get_functionId()

Retourne l'identifiant matériel de l'afficheur, sans référence au module.

segmenteddisplay→get_hardwareId()

Retourne l'identifiant matériel unique de l'afficheur au format `SERIAL.FUNCTIONID`.

segmenteddisplay→get_logicalName()

Retourne le nom logique de l'afficheur.

segmenteddisplay→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

segmenteddisplay→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

segmenteddisplay→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

segmenteddisplay→isOnline()

Vérifie si le module hébergeant l'afficheur est joignable, sans déclencher d'erreur.

segmenteddisplay→isOnline_async(callback, context)

Vérifie si le module hébergeant l'afficheur est joignable, sans déclencher d'erreur.

segmenteddisplay→load(msValidity)

Met en cache les valeurs courantes de l'afficheur, avec une durée de validité spécifiée.

segmenteddisplay→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

segmenteddisplay→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'afficheur, avec une durée de validité spécifiée.

segmenteddisplay→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

segmenteddisplay→nextSegmentedDisplay()

Continue l'énumération des un afficheur commencée à l'aide de `yFirstSegmentedDisplay()`.

segmenteddisplay→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

segmenteddisplay→set_displayedText(newval)

Modifie le texte actuellement affiché à l'écran.

segmenteddisplay→set_logicalName(newval)

Modifie le nom logique de l'afficheur.

segmenteddisplay→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`.

segmenteddisplay→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

segmenteddisplay→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.

YSegmentedDisplay.FindSegmentedDisplay() YSegmentedDisplay yFindSegmentedDisplay() YSegmentedDisplay.FindSegmentedDisplay() YSegmentedDisplay.FindSegmentedDisplay()

Permet de retrouver un afficheur d'après un identifiant donné.

function **FindSegmentedDisplay**(**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'afficheur soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YSegmentedDisplay.isOnline()` pour tester si l'afficheur 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence l'afficheur sans ambiguïté

Retourne :

un objet de classe `YSegmentedDisplay` qui permet ensuite de contrôler l'afficheur.

YSegmentedDisplay.FindSegmentedDisplayInContext() YSegmentedDisplay yFindSegmentedDisplayInContext() YSegmentedDisplay.FindSegmentedDisplayInContext() YSegmentedDisplay.FindSegmentedDisplayInContext()

Permet de retrouver un afficheur d'après un identifiant donné dans un Context YAPI.

function **FindSegmentedDisplayInContext**(**yctx**, **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'afficheur soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YSegmentedDisplay.isOnline()` pour tester si l'afficheur 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'afficheur sans ambiguïté

Retourne :

un objet de classe `YSegmentedDisplay` qui permet ensuite de contrôler l'afficheur.

YSegmentedDisplay.FirstSegmentedDisplay() YSegmentedDisplay yFirstSegmentedDisplay()
YSegmentedDisplay.FirstSegmentedDisplay()
YSegmentedDisplay.FirstSegmentedDisplay()

Commence l'énumération des un afficheur accessibles par la librairie.

function **FirstSegmentedDisplay**( )

Utiliser la fonction `YSegmentedDisplay.nextSegmentedDisplay()` pour itérer sur les autres un afficheur.

Retourne :

un pointeur sur un objet `YSegmentedDisplay`, correspondant au premier afficheur accessible en ligne, ou `null` si il n'y a pas de un afficheur disponibles.

YSegmentedDisplay.FirstSegmentedDisplayInContext() YSegmentedDisplay.yFirstSegmentedDisplayInContext() YSegmentedDisplay.FirstSegmentedDisplayInContext() YSegmentedDisplay.FirstSegmentedDisplayInContext()

Commence l'énumération des un afficheur accessibles par la librairie.

function **FirstSegmentedDisplayInContext**(**yctx**)

Utiliser la fonction `YSegmentedDisplay.nextSegmentedDisplay()` pour itérer sur les autres un afficheur.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YSegmentedDisplay`, correspondant au premier afficheur accessible en ligne, ou `null` si il n'y a pas de un afficheur disponibles.

segmenteddisplay→clearCache() YSegmentedDisplay
segmenteddisplay.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'afficheur. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

segmenteddisplay→describe() YSegmentedDisplay
segmenteddisplay.describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'afficheur au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant l'afficheur (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

segmenteddisplay→get_advertisedValue() YSegmentedDisplay
segmenteddisplay→advertisedValue()
segmenteddisplay.get_advertisedValue()
segmenteddisplay.get_advertisedValue()

Retourne la valeur courante de l'afficheur (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante de l'afficheur (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

segmenteddisplay→get_displayedText() YSegmentedDisplay
segmenteddisplay→displayedText()
segmenteddisplay.get_displayedText()
segmenteddisplay.get_displayedText()

Retourne le texte actuellement affiché à l'écran.

function **get_displayedText**( )

Retourne :
une chaîne de caractères représentant le texte actuellement affiché à l'écran

En cas d'erreur, déclenche une exception ou retourne `Y_DISPLAYEDTEXT_INVALID`.

segmenteddisplay→get_errorMessage() YSegmentedDisplay
segmenteddisplay→errorMessage()
segmenteddisplay.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'afficheur.

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'afficheur.

segmenteddisplay→get_errorType() YSegmentedDisplay
segmenteddisplay→errorType()
segmenteddisplay.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'afficheur.

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'afficheur.

segmenteddisplay→get_friendlyName() YSegmentedDisplay
segmenteddisplay→friendlyName()
segmenteddisplay.get_friendlyName()

Retourne un identifiant global de l'afficheur au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et de l'afficheur si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel de l'afficheur (par exemple: MyCustomName.relay1)

Retourne :
une chaîne de caractères identifiant l'afficheur en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

segmenteddisplay→get_functionDescriptor() YSegmentedDisplay
segmenteddisplay→functionDescriptor()
segmenteddisplay.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

segmenteddisplay→get_functionId() YSegmentedDisplay
segmenteddisplay→functionId()
segmenteddisplay.get_functionId()

Retourne l'identifiant matériel de l'afficheur, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant l'afficheur (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

segmenteddisplay→get_hardwareId() YSegmentedDisplay
segmenteddisplay→hardwareId()
segmenteddisplay.get_hardwareId()

Retourne l'identifiant matériel unique de l'afficheur 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'afficheur (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant l'afficheur (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

segmenteddisplay→get_logicalName() YSegmentedDisplay
segmenteddisplay→logicalName()
segmenteddisplay.get_logicalName()
segmenteddisplay.get_logicalName()

Retourne le nom logique de l'afficheur.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique de l'afficheur.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

segmenteddisplay→get_module() YSegmentedDisplay
segmenteddisplay→module()
segmenteddisplay.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

segmenteddisplay→get_userData() YSegmentedDisplay
segmenteddisplay→userData()
segmenteddisplay.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.

segmenteddisplay→isOnline() YSegmentedDisplay
segmenteddisplay.isOnline()

Vérifie si le module hébergeant l'afficheur est joignable, sans déclencher d'erreur.

```
function isOnline( )
```

Si les valeurs des attributs en cache de l'afficheur sont valides au moment de l'appel, le module est considéré joignable. 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'afficheur est joignable, `false` sinon

segmenteddisplay→load()segmenteddisplay.load() YSegmentedDisplay

Met en cache les valeurs courantes de l'afficheur, 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 occasionellement 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.

segmenteddisplay→loadAttribute() YSegmentedDisplay
segmenteddisplay.loadAttribute()
segmenteddisplay.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

segmenteddisplay→muteValueCallbacks() YSegmentedDisplay
segmenteddisplay.muteValueCallbacks()
segmenteddisplay.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

segmenteddisplay→nextSegmentedDisplay() YSegmentedDisplay
segmenteddisplay.nextSegmentedDisplay()
segmenteddisplay.nextSegmentedDisplay()

Continue l'énumération des un afficheur commencée à l'aide de `yFirstSegmentedDisplay()`.

function **nextSegmentedDisplay**( )

Retourne :

un pointeur sur un objet `YSegmentedDisplay` accessible en ligne, ou `null` lorsque l'énumération est terminée.

segmenteddisplay→registerValueCallback() YSegmentedDisplay
segmenteddisplay.registerValueCallback()
segmenteddisplay.registerValueCallback()

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.

segmenteddisplay→set_displayedText() YSegmentedDisplay
segmenteddisplay→setDisplayedText()
segmenteddisplay.set_displayedText()
segmenteddisplay.set_displayedText()

Modifie le texte actuellement affiché à l'écran.

function **set_displayedText**(**newval**)

Paramètres :

newval une chaîne de caractères représentant le texte actuellement affiché à 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.

segmenteddisplay→set_logicalName() YSegmentedDisplay
segmenteddisplay→setLogicalName()
segmenteddisplay.set_logicalName()
segmenteddisplay.set_logicalName()

Modifie le nom logique de l'afficheur.

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'afficheur.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

segmenteddisplay→set_userData() YSegmentedDisplay
segmenteddisplay→setUserData()
segmenteddisplay.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

segmenteddisplay→unmuteValueCallbacks() YSegmentedDisplay
segmenteddisplay.unmuteValueCallbacks()
segmenteddisplay.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

segmenteddisplay→wait_async() YSegmentedDisplay
segmenteddisplay.wait_async()

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.

3.59. Interface des fonctions de type senseur

La classe YSensor est la classe parente de tous les senseurs Yoctopuce. Elle permet de lire la valeur courante et l'unité de n'importe quel capteur, de lire les valeurs min/max, de configurer la fréquence d'enregistrement autonome des données et de récupérer les mesures enregistrées. Elle permet aussi d'enregistrer un callback appelé lorsque la valeur mesurée change ou à intervalle prédéfini. L'utilisation de cette classe plutôt qu'une de ces sous-classes permet de créer des application génériques, compatibles même avec les capteurs Yoctopuce futurs. Note: la classe YAnButton est le seul type d'entrée analogique qui n'hérite pas de YSensor.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_api.js'></script>
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;
uwp	import com.yoctopuce.YoctoAPI.YModule;
py	from yocto_api import *
php	require_once('yocto_api.php');
es	in HTML: <script src="../../lib/yocto_api.js"></script> in node.js: require('yoctolib-es2017/yocto_api.js');

Fonction globales

yFindSensor(**func**)

Permet de retrouver un senseur d'après un identifiant donné.

yFindSensorInContext(**yctx**, **func**)

Permet de retrouver un senseur d'après un identifiant donné dans un Context YAPI.

yFirstSensor()

Commence l'énumération des senseurs accessibles par la librairie.

yFirstSensorInContext(**yctx**)

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→clearCache()

Invalide le cache.

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), en l'unité spécifiée, sous forme de nombre à virgule.

sensor→get_currentValue()

Retourne la valeur actuelle de la measure, en l'unité spécifiée, sous forme de nombre à virgule.

sensor→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

sensor→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

sensor→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

sensor→get_friendlyName()

Retourne un identifiant global du senseur au format `NOM_MODULE.NOM_FONCTION`.

sensor→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

sensor→get_functionId()

Retourne l'identifiant matériel du senseur, sans référence au module.

sensor→get_hardwareId()

Retourne l'identifiant matériel unique du senseur au format `SERIAL.FUNCTIONID`.

sensor→get_highestValue()

Retourne la valeur maximale observée pour la measure depuis le démarrage du module.

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.

sensor→get_logicalName()

Retourne le nom logique du senseur.

sensor→get_lowestValue()

Retourne la valeur minimale observée pour la measure depuis le démarrage du module.

sensor→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

sensor→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

sensor→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.

sensor→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

sensor→get_resolution()

Retourne la résolution des valeurs mesurées.

sensor→get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

sensor→get_unit()

Retourne l'unité dans laquelle la measure est exprimée.

sensor→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

sensor→isOnline()

Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.

sensor→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.

sensor→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

sensor→load(msValidity)

Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.

sensor→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

sensor→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

sensor→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.

sensor→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

sensor→nextSensor()

Continue l'énumération des senseurs commencée à l'aide de `yFirstSensor()`.

sensor→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

sensor→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

sensor→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

sensor→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

sensor→set_logicalName(newval)

Modifie le nom logique du senseur.

sensor→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

sensor→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

sensor→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

sensor→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

sensor→startDataLogger()

Démarre l'enregistreur de données du module.

sensor→stopDataLogger()

Arrête l'enregistreur de données du module.

sensor→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

sensor→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YSensor.FindSensor() YSensor yFindSensor()YSensor.FindSensor() YSensor.FindSensor()

Permet de retrouver un senseur d'après un identifiant donné.

```
function FindSensor( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindSensorInContext() YSensor
yFindSensorInContext()
YSensor.FindSensorInContext()
YSensor.FindSensorInContext()

Permet de retrouver un senseur d'après un identifiant donné dans un Context YAPI.

function **FindSensorInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le senseur sans ambiguïté

Retourne :

un objet de classe `YSensor` qui permet ensuite de contrôler le senseur.

YSensor.FirstSensor() YSensor yFirstSensor()YSensor.FirstSensor() YSensor.FirstSensor()

Commence l'énumération des senseurs accessibles par la librairie.

function **FirstSensor**( )

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.

YSensor.FirstSensorInContext() YSensor
yFirstSensorInContext()
YSensor.FirstSensorInContext()
YSensor.FirstSensorInContext()

Commence l'énumération des senseurs accessibles par la librairie.

function **FirstSensorInContext**(**yctx**)

Utiliser la fonction `YSensor.nextSensor()` pour itérer sur les autres senseurs.

Paramètres :

yctx un contexte YAPI.

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() YSensor
sensor.calibrateFromPoints()
sensor.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→clearCache()sensor.clearCache() YSensor

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du senseur. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le senseur (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

sensor→get_advertisedValue() YSensor
sensor→advertisedValue()
sensor.get_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()
sensor.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

sensor→get_currentValue() YSensor
sensor→currentValue()sensor.get_currentValue()
sensor.get_currentValue()

Retourne la valeur actuelle de la measure, en l'unité spécifiée, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle de la measure, en l'unité spécifiée, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

sensor→get_dataLogger() YSensor
sensor→dataLogger()sensor.get_dataLogger()
sensor.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

sensor→get_errorMessage() YSensor
sensor→errorMessage()sensor.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

function **get_errorMessage**( )

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du senseur.

sensor→get_errorType() YSensor
sensor→errorType()sensor.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

function **get_errorType**( )

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :
un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du senseur.

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'identifant 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 example `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 example 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()
sensor.get_highestValue()

Retourne la valeur maximale observée pour la measure depuis le démarrage du module.

function **get_highestValue**( )

Retourne :
une valeur numérique représentant la valeur maximale observée pour la measure 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()
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()
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()
sensor.get_lowestValue()

Retourne la valeur minimale observée pour la measure depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :
une valeur numérique représentant la valeur minimale observée pour la measure 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_recordedData() YSensor
sensor→recordedData()sensor.get_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**)

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 find de l'intercalle 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()
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()
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_sensorState() YSensor
sensor→sensorState()sensor.get_sensorState()
sensor.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :
un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

sensor→get_unit() YSensor
sensor→unit()sensor.get_unit()sensor.get_unit()

Retourne l'unité dans laquelle la measure est exprimée.

function **get_unit**( )

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la measure 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→isSensorReady()sensor.isSensorReady() YSensor sensor.isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

function **isSensorReady**( )

Retourne faux si le module n'est pas joignable, ou que le capteur n'a pas de mesure actuelle à communiquer. 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 dispose d'une mesure actuelle, `false` sinon

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 occasionellement 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→loadAttribute()sensor.loadAttribute() YSensor sensor.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

sensor→loadCalibrationPoints() YSensor
sensor.loadCalibrationPoints()
sensor.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→muteValueCallbacks() YSensor
sensor.muteValueCallbacks()
sensor.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→nextSensor()sensor.nextSensor() YSensor sensor.nextSensor()

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() YSensor
sensor.registerTimedReportCallback()
sensor.registerTimedReportCallback()

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() YSensor
sensor.registerValueCallback()
sensor.registerValueCallback()

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()
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() YSensor
sensor→setLogFrequency()
sensor.set_logFrequency()
sensor.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.

sensor→set_logicalName() YSensor
sensor→setLogicalName()sensor.set_logicalName()
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()
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()
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()
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'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

sensor→startDataLogger()sensor.startDataLogger() YSensor sensor.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

sensor→stopDataLogger()sensor.stopDataLogger() YSensor sensor.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

sensor→unmuteValueCallbacks() YSensor
sensor.unmuteValueCallbacks()
sensor.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.60. Interface de la fonction SerialPort

La fonction SerialPort permet de piloter entièrement un module d'interface série Yoctopuce, pour envoyer et recevoir des données et configurer les paramètres de transmission (vitesse, nombre de bits, parité, contrôle de flux et protocole). Notez que les interfaces série Yoctopuce ne sont pas des visibles comme des ports COM virtuels. Ils sont faits pour être utilisés comme tous les autres modules Yoctopuce.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_serialport.js'></script>
cpp	#include "yocto_serialport.h"
m	#import "yocto_serialport.h"
pas	uses yocto_serialport;
vb	yocto_serialport.vb
cs	yocto_serialport.cs
java	import com.yoctopuce.YoctoAPI.YSerialPort;
uwp	import com.yoctopuce.YoctoAPI.YSerialPort;
py	from yocto_serialport import *
php	require_once('yocto_serialport.php');
es	in HTML: <script src="../../lib/yocto_serialport.js"></script> in node.js: require('yoctolib-es2017/yocto_serialport.js');

Fonction globales
yFindSerialPort(**func**) Permet de retrouver une port série d'après un identifiant donné.
yFindSerialPortInContext(**yctx**, **func**) Permet de retrouver une port série d'après un identifiant donné dans un Context YAPI.
yFirstSerialPort() Commence l'énumération des le port série accessibles par la librairie.
yFirstSerialPortInContext(**yctx**) Commence l'énumération des le port série accessibles par la librairie.
Méthodes des objets `YSerialPort`
serialport→clearCache() Invalide le cache.
serialport→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du port série au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
serialport→get_CTS() Lit l'état de la ligne CTS.
serialport→get_advertisedValue() Retourne la valeur courante du port série (pas plus de 6 caractères).
serialport→get_currentJob() Retourne le nom du fichier de tâches actif en ce moment.
serialport→get_errCount() Retourne le nombre d'erreurs de communication détectées depuis la dernière mise à zéro.
serialport→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port série.
serialport→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port série.

serialport→get_friendlyName()

Retourne un identifiant global du port série au format `NOM_MODULE.NOM_FONCTION`.

serialport→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

serialport→get_functionId()

Retourne l'identifiant matériel du port série, sans référence au module.

serialport→get_hardwareId()

Retourne l'identifiant matériel unique du port série au format `SERIAL.FUNCTIONID`.

serialport→get_lastMsg()

Retourne le dernier message reçu (pour les protocoles de type Line, Frame et Modbus).

serialport→get_logicalName()

Retourne le nom logique du port série.

serialport→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

serialport→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

serialport→get_protocol()

Retourne le type de protocole utilisé sur la communication série, sous forme d'une chaîne de caractères.

serialport→get_rxCount()

Retourne le nombre d'octets reçus depuis la dernière mise à zéro.

serialport→get_rxMsgCount()

Retourne le nombre de messages reçus depuis la dernière mise à zéro.

serialport→get_serialMode()

Retourne les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1".

serialport→get_startupJob()

Retourne le nom du fichier de tâches à exécuter au démarrage du module.

serialport→get_txCount()

Retourne le nombre d'octets transmis depuis la dernière mise à zéro.

serialport→get_txMsgCount()

Retourne le nombre de messages envoyés depuis la dernière mise à zéro.

serialport→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

serialport→get_voltageLevel()

Retourne le niveau de tension utilisé par le module sur le port série.

serialport→isOnline()

Vérifie si le module hébergeant le port série est joignable, sans déclencher d'erreur.

serialport→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le port série est joignable, sans déclencher d'erreur.

serialport→load(**msValidity**)

Met en cache les valeurs courantes du port série, avec une durée de validité spécifiée.

serialport→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

serialport→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du port série, avec une durée de validité spécifiée.

serialport→modbusReadBits(slaveNo, pduAddr, nBits)

Lit un ou plusieurs bits contigus depuis un périphérique MODBUS.

serialport→modbusReadInputBits(slaveNo, pduAddr, nBits)

Lit un ou plusieurs bits contigus depuis un périphérique MODBUS.

serialport→modbusReadInputRegisters(slaveNo, pduAddr, nWords)

Lit un ou plusieurs registres d'entrée (registre enlecture seule) depuis un périphérique MODBUS.

serialport→modbusReadRegisters(slaveNo, pduAddr, nWords)

Lit un ou plusieurs registres interne depuis un périphérique MODBUS.

serialport→modbusWriteAndReadRegisters(slaveNo, pduWriteAddr, values, pduReadAddr, nReadWords)

Modifie l'état de plusieurs bits (ou relais) contigus sur un périphérique MODBUS.

serialport→modbusWriteBit(slaveNo, pduAddr, value)

Modifie l'état d'un seul bit (ou relais) sur un périphérique MODBUS.

serialport→modbusWriteBits(slaveNo, pduAddr, bits)

Modifie l'état de plusieurs bits (ou relais) contigus sur un périphérique MODBUS.

serialport→modbusWriteRegister(slaveNo, pduAddr, value)

Modifie la valeur d'un registre interne 16 bits sur un périphérique MODBUS.

serialport→modbusWriteRegisters(slaveNo, pduAddr, values)

Modifie l'état de plusieurs registres internes 16 bits contigus sur un périphérique MODBUS.

serialport→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

serialport→nextSerialPort()

Continue l'énumération des le port série commencée à l'aide de `yFirstSerialPort()`.

serialport→queryLine(query, maxWait)

Envoie un message sous forme de ligne de texte sur le port série, et lit la réponse reçue.

serialport→queryMODBUS(slaveNo, pduBytes)

Envoie un message à un périphérique MODBUS esclave connecté au port série, et lit la réponse reçue.

serialport→readArray(nChars)

Lit le contenu du tampon de réception sous forme de liste d'octets, à partir de la position courante dans le flux de donnée.

serialport→readBin(nChars)

Lit le contenu du tampon de réception sous forme d'objet binaire, à partir de la position courante dans le flux de donnée.

serialport→readByte()

Lit le prochain byte dans le tampon de réception, à partir de la position courante dans le flux de donnée.

serialport→readHex(nBytes)

Lit le contenu du tampon de réception sous forme hexadécimale, à partir de la position courante dans le flux de donnée.

serialport→readLine()

Lit la prochaine ligne (ou le prochain message) du tampon de réception, à partir de la position courante dans le flux de donnée.

serialport→readMessages(pattern, maxWait)

Cherche les messages entrants dans le tampon de réception correspondant à un format donné, à partir de la position courante.

serialport→readStr(nChars)

Lit le contenu du tampon de réception sous forme de string, à partir de la position courante dans le flux de donnée.

serialport→read_avail()

Retourne le nombre de bytes prêts à être lus dans le tampon de réception, depuis la position courante dans le flux de donnée utilisé par l'objet d'API.

serialport→read_seek(**absPos**)

Change le pointeur de position courante dans le flux de donnée à la valeur spécifiée.

serialport→read_tell()

Retourne la valeur actuelle du pointeur de position courante dans le flux de donnée utilisé par l'objet d'API.

serialport→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

serialport→reset()

Remet à zéro tous les compteurs et efface les tampons.

serialport→selectJob(**jobfile**)

Charge et execute le fichier de tâche spécifié.

serialport→set_RTS(**val**)

Change manuellement l'état de la ligne RTS.

serialport→set_currentJob(**newval**)

Modifie le nom du job à exécuter au démarrage du module.

serialport→set_logicalName(**newval**)

Modifie le nom logique du port série.

serialport→set_protocol(**newval**)

Modifie le type de protocol utilisé sur la communication série.

serialport→set_serialMode(**newval**)

Modifie les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1".

serialport→set_startupJob(**newval**)

Modifie le nom du job à exécuter au démarrage du module.

serialport→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`.

serialport→set_voltageLevel(**newval**)

Modifie le niveau de tension utilisé par le module sur le port série.

serialport→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

serialport→uploadJob(**jobfile**, **jsonDef**)

Sauvegarde une définition de tâche (au format JSON) dans un fichier.

serialport→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.

serialport→writeArray(**byteList**)

Envoie une séquence d'octets (fournie sous forme d'une liste) sur le port série.

serialport→writeBin(**buff**)

Envoie un objet binaire tel quel sur le port série.

serialport→writeByte(**code**)

Envoie un unique byte sur le port série.

serialport→writeHex(**hexString**)

Envoie une séquence d'octets (fournie sous forme de chaîne hexadécimale) sur le port série.

serialport→writeLine(**text**)

Envoie une chaîne de caractères sur le port série, suivie d'un saut de ligne (CR LF).

serialport→writeMODBUS(hexString)

Envoie une commande MODBUS (fournie sous forme de chaîne hexadécimale) sur le port série.

serialport→writeStr(text)

Envoie une chaîne de caractères telle quelle sur le port série.

YSerialPort.FindSerialPort() YSerialPort yFindSerialPort()YSerialPort.FindSerialPort() YSerialPort.FindSerialPort()

Permet de retrouver une port série d'après un identifiant donné.

```
function FindSerialPort( 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 série soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YSerialPort.isOnline()` pour tester si le port série est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le port série sans ambiguïté

Retourne :

un objet de classe `YSerialPort` qui permet ensuite de contrôler le port série.

YSerialPort.FindSerialPortInContext() YSerialPort
yFindSerialPortInContext()
YSerialPort.FindSerialPortInContext()
YSerialPort.FindSerialPortInContext()

Permet de retrouver une port série d'après un identifiant donné dans un Context YAPI.

function **FindSerialPortInContext**(**yctx**, **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 série soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YSerialPort.isOnline()` pour tester si le port série est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence le port série sans ambiguïté

Retourne :

un objet de classe `YSerialPort` qui permet ensuite de contrôler le port série.

YSerialPort.FirstSerialPort() YSerialPort yFirstSerialPort()YSerialPort.FirstSerialPort() YSerialPort.FirstSerialPort()

Commence l'énumération des le port série accessibles par la librairie.

function **FirstSerialPort**( )

Utiliser la fonction `YSerialPort.nextSerialPort()` pour itérer sur les autres le port série.

Retourne :

un pointeur sur un objet `YSerialPort`, correspondant au premier port série accessible en ligne, ou `null` si il n'y a pas du port série disponibles.

YSerialPort.FirstSerialPortInContext() YSerialPort
yFirstSerialPortInContext()
YSerialPort.FirstSerialPortInContext()
YSerialPort.FirstSerialPortInContext()

Commence l'énumération des le port série accessibles par la librairie.

function **FirstSerialPortInContext**(**yctx**)

Utiliser la fonction `YSerialPort.nextSerialPort()` pour itérer sur les autres le port série.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YSerialPort`, correspondant au premier port série accessible en ligne, ou `null` si il n'y a pas du port série disponibles.

serialport→clearCache()serialport.clearCache() YSerialPort

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du port série. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

serialport→describe()serialport.describe() YSerialPort

Retourne un court texte décrivant de manière non-ambigüe l'instance du port série au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le port série (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

serialport→get_CTS() YSerialPort
serialport→CTS()serialport.get_CTS()
serialport.get_CTS()

Lit l'état de la ligne CTS.

function **get_CTS**( )

La ligne CTS est habituellement pilotée par le signal RTS du périphérique série connecté.

Retourne :
1 si le CTS est signalé (niveau haut), 0 si le CTS n'est pas actif (niveau bas).

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→get_advertisedValue() YSerialPort
serialport→advertisedValue()
serialport.get_advertisedValue()
serialport.get_advertisedValue()

Retourne la valeur courante du port série (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante du port série (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

serialport→get_currentJob() YSerialPort
serialport→currentJob()serialport.get_currentJob()
serialport.get_currentJob()

Retourne le nom du fichier de tâches actif en ce moment.

function **get_currentJob**( )

Retourne :
une chaîne de caractères représentant le nom du fichier de tâches actif en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTJOB_INVALID`.

serialport→get_errCount() YSerialPort
serialport→errCount()serialport.get_errCount()
serialport.get_errCount()

Retourne le nombre d'erreurs de communication détectées depuis la dernière mise à zéro.

function **get_errCount**( )

Retourne :
un entier représentant le nombre d'erreurs de communication détectées depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne `Y_ERRCOUNT_INVALID`.

serialport→get_errorMessage() YSerialPort
serialport→errorMessage()
serialport.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port série.

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 série.

serialport→get_errorType() YSerialPort
serialport→errorType()serialport.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port série.

```
function get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :
un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du port série.

serialport→get_friendlyName() YSerialPort
serialport→friendlyName()
serialport.get_friendlyName()

Retourne un identifiant global du port série au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du port série si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du port série (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le port série en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

serialport→get_functionDescriptor() YSerialPort
serialport→functionDescriptor()
serialport.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

serialport→get_functionId() YSerialPort
serialport→functionId()serialport.get_functionId()

Retourne l'identifiant matériel du port série, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le port série (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

serialport→get_hardwareId() YSerialPort
serialport→hardwareId()serialport.get_hardwareId()

Retourne l'identifiant matériel unique du port série au format SERIAL.FUNCTIONID.

function **get_hardwareId**( )

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du port série (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le port série (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

serialport→get_lastMsg() YSerialPort
serialport→lastMsg()serialport.get_lastMsg()
serialport.get_lastMsg()

Retourne le dernier message reçu (pour les protocoles de type Line, Frame et Modbus).

function **get_lastMsg**( )

Retourne :
une chaîne de caractères représentant le dernier message reçu (pour les protocoles de type Line, Frame et Modbus)

En cas d'erreur, déclenche une exception ou retourne `Y_LASTMSG_INVALID`.

serialport→get_logicalName() YSerialPort
serialport→logicalName()
serialport.get_logicalName()
serialport.get_logicalName()

Retourne le nom logique du port série.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du port série.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

serialport→get_module() YSerialPort
serialport→module()serialport.get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

function **get_module**( )

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :
une instance de YModule

serialport→get_protocol() YSerialPort
serialport→protocol()serialport.get_protocol()
serialport.get_protocol()

Retourne le type de protocole utilisé sur la communication série, sous forme d'une chaîne de caractères.

function **get_protocol**( )

Les valeurs possibles sont "Line" pour des messages ASCII séparés par des retours de ligne, "Frame:[timeout]ms" pour des messages binaires séparés par une temporisation, "Modbus-ASCII" pour des messages MODBUS en mode ASCII, "Modbus-RTU" pour des messages MODBUS en mode RTU, "Wiegand-ASCII" pour des messages Wiegand en mode ASCII, "Wiegand-26","Wiegand-34", etc pour des messages Wiegand en mode octet, "Char" pour un flux ASCII continu ou "Byte" pour un flux binaire continue.

Retourne :

une chaîne de caractères représentant le type de protocole utilisé sur la communication série, sous forme d'une chaîne de caractères

En cas d'erreur, déclenche une exception ou retourne `Y_PROTOCOL_INVALID`.

serialport→get_rxCount() YSerialPort
serialport→rxCount()serialport.get_rxCount()
serialport.get_rxCount()

Retourne le nombre d'octets reçus depuis la dernière mise à zéro.

function **get_rxCount**( )

Retourne :
un entier représentant le nombre d'octets reçus depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne `Y_RXCOUNT_INVALID`.

serialport→get_rxMsgCount() YSerialPort
serialport→rxMsgCount()
serialport.get_rxMsgCount()
serialport.get_rxMsgCount()

Retourne le nombre de messages reçus depuis la dernière mise à zéro.

function **get_rxMsgCount**( )

Retourne :
un entier représentant le nombre de messages reçus depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne `Y_RXMSGCOUNT_INVALID`.

serialport→get_serialMode() YSerialPort
serialport→serialMode()serialport.get_serialMode()
serialport.get_serialMode()

Retourne les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1".

function **get_serialMode**( )

La chaîne contient le taux de transfert, le nombre de bits de données, la parité parité et le nombre de bits d'arrêt. Un suffixe supplémentaire optionnel est inclus si une option de contrôle de flux est active: "CtsRts" pour le contrôle de flux matériel, "XOnXOff" pour le contrôle de flux logique et "Simplex" pour l'utilisation du signal RTS pour l'acquisition d'un bus partagé (tel qu'utilisé pour certains adaptateurs RS485 par exemple).

Retourne :
une chaîne de caractères représentant les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1"

En cas d'erreur, déclenche une exception ou retourne `Y_SERIALMODE_INVALID`.

serialport→get_startupJob() YSerialPort
serialport→startupJob()serialport.get_startupJob()
serialport.get_startupJob()

Retourne le nom du fichier de tâches à exécuter au démarrage du module.

function **get_startupJob**( )

Retourne :
une chaîne de caractères représentant le nom du fichier de tâches à exécuter au démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_STARTUPJOB_INVALID`.

serialport→get_txCount() YSerialPort
serialport→txCount()serialport.get_txCount()
serialport.get_txCount()

Retourne le nombre d'octets transmis depuis la dernière mise à zéro.

function **get_txCount**( )

Retourne :
un entier représentant le nombre d'octets transmis depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne `Y_TXCOUNT_INVALID`.

serialport→get_txMsgCount() YSerialPort
serialport→txMsgCount()serialport.get_txMsgCount()
serialport.get_txMsgCount()

Retourne le nombre de messages envoyés depuis la dernière mise à zéro.

function **get_txMsgCount**( )

Retourne :
un entier représentant le nombre de messages envoyés depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne `Y_TXMSGCOUNT_INVALID`.

serialport→get_userData() YSerialPort
serialport→userData()serialport.get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

function **get_userData**( )

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.

serialport→get_voltageLevel() YSerialPort
serialport→voltageLevel()
serialport.get_voltageLevel()
serialport.get_voltageLevel()

Retourne le niveau de tension utilisé par le module sur le port série.

function **get_voltageLevel**( )

Retourne :

une valeur parmi Y_VOLTAGELEVEL_OFF, Y_VOLTAGELEVEL_TTL3V, Y_VOLTAGELEVEL_TTL3VR, Y_VOLTAGELEVEL_TTL5V, Y_VOLTAGELEVEL_TTL5VR, Y_VOLTAGELEVEL_RS232 et Y_VOLTAGELEVEL_RS485 représentant le niveau de tension utilisé par le module sur le port série

En cas d'erreur, déclenche une exception ou retourne Y_VOLTAGELEVEL_INVALID.

serialport→isOnline()serialport.isOnline() YSerialPort

Vérifie si le module hébergeant le port série est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du port série sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si le port série est joignable, `false` sinon

serialport→load()serialport.load() YSerialPort

Met en cache les valeurs courantes du port série, avec une durée de validité spécifiée.

function **load**(**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 occasionellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→loadAttribute()serialport.loadAttribute() YSerialPort serialport.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

serialport→modbusReadBits() YSerialPort
serialport.modbusReadBits()
serialport.modbusReadBits()

Lit un ou plusieurs bits contigus depuis un périphérique MODBUS.

function **modbusReadBits**(**slaveNo**, **pduAddr**, **nBits**)

Cette méthode utilise le code de fonction MODBUS 0x01 (Read Coils).

Paramètres :

slaveNo	addresse du périphérique MODBUS esclave à interroger
pduAddr	adresse relative du premier bit à lire (indexé à partir de zéro).
nBits	nombre de bits à lire

Retourne :

un vecteur d'entiers, correspondant chacun à un bit.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

serialport→modbusReadInputBits() YSerialPort
serialport.modbusReadInputBits()
serialport.modbusReadInputBits()

Lit un ou plusieurs bits contigus depuis un périphérique MODBUS.

function **modbusReadInputBits**(**slaveNo**, **pduAddr**, **nBits**)

Cette méthode utilise le code de fonction MODBUS 0x02 (Read Discrete Inputs).

Paramètres :

slaveNo addresse du périphérique MODBUS esclave à interroger

pduAddr adresse relative du premier bit à lire (indexé à partir de zéro).

nBits nombre de bits à lire

Retourne :

un vecteur d'entiers, correspondant chacun à un bit.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

serialport→modbusReadInputRegisters() YSerialPort
serialport.modbusReadInputRegisters()
serialport.modbusReadInputRegisters()

Lit un ou plusieurs registres d'entrée (registre enlecture seule) depuis un périphérique MODBUS.

function **modbusReadInputRegisters**(**slaveNo**, **pduAddr**, **nWords**)

Cette méthode utilise le code de fonction MODBUS 0x04 (Read Input Registers).

Paramètres :

slaveNo addresse du périphérique MODBUS esclave à interroger

pduAddr adresse relative du premier registre d'entrée à lire (indexé à partir de zéro).

nWords nombre de registres d'entrée à lire

Retourne :

un vecteur d'entiers, correspondant chacun à une valeur d'entrée (16 bits).

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

serialport→modbusReadRegisters() YSerialPort
serialport.modbusReadRegisters()
serialport.modbusReadRegisters()

Lit un ou plusieurs registres interne depuis un périphérique MODBUS.

function **modbusReadRegisters**(**slaveNo**, **pduAddr**, **nWords**)

Cette méthode utilise le code de fonction MODBUS 0x03 (Read Holding Registers).

Paramètres :

slaveNo addresse du périphérique MODBUS esclave à interroger

pduAddr adresse relative du premier registre interne à lire (indexé à partir de zéro).

nWords nombre de registres internes à lire

Retourne :

un vecteur d'entiers, correspondant chacun à une valeur de registre (16 bits).

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

serialport→modbusWriteAndReadRegisters() YSerialPort
serialport.modbusWriteAndReadRegisters()
serialport.modbusWriteAndReadRegisters()

Modifie l'état de plusieurs bits (ou relais) contigus sur un périphérique MODBUS.

function **modbusWriteAndReadRegisters**(**slaveNo**, **pduWriteAddr**, **values**, **pduReadAddr**, **nReadWords**)

Cette méthode utilise le code de fonction MODBUS 0x17 (Read/Write Multiple Registers).

Paramètres :

slaveNo	addresse du périphérique MODBUS esclave à piloter
pduWriteAddr	adresse relative du premier registre interne à modifier (indexé à partir de zéro).
values	vecteur de valeurs 16 bits à appliquer
pduReadAddr	adresse relative du premier registre interne à lire (indexé à partir de zéro).
nReadWords	nombre de registres internes à lire

Retourne :

un vecteur d'entiers, correspondant chacun à une valeur de registre (16 bits) lue.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

serialport→modbusWriteBit() YSerialPort
serialport.modbusWriteBit()
serialport.modbusWriteBit()

Modifie l'état d'un seul bit (ou relais) sur un périphérique MODBUS.

function **modbusWriteBit**(**slaveNo**, **pduAddr**, **value**)

Cette méthode utilise le code de fonction MODBUS 0x05 (Write Single Coil).

Paramètres :

slaveNo addresse du périphérique MODBUS esclave à piloter

pduAddr adresse relative du bit à modifier (indexé à partir de zéro).

value la valeur à appliquer (0 pour l'état OFF, non-zéro pour l'état ON)

Retourne :

le nombre de bits affectés sur le périphérique (1)

En cas d'erreur, déclenche une exception ou retourne zéro.

serialport→modbusWriteBits() YSerialPort
serialport.modbusWriteBits()
serialport.modbusWriteBits()

Modifie l'état de plusieurs bits (ou relais) contigus sur un périphérique MODBUS.

function **modbusWriteBits**(**slaveNo**, **pduAddr**, **bits**)

Cette méthode utilise le code de fonction MODBUS 0x0f (Write Multiple Coils).

Paramètres :

slaveNo	addresse du périphérique MODBUS esclave à piloter
pduAddr	adresse relative du premier bit à modifier (indexé à partir de zéro).
bits	vecteur de bits à appliquer (un entier par bit)

Retourne :

le nombre de bits affectés sur le périphérique

En cas d'erreur, déclenche une exception ou retourne zéro.

serialport→modbusWriteRegister() YSerialPort
serialport.modbusWriteRegister()
serialport.modbusWriteRegister()

Modifie la valeur d'un registre interne 16 bits sur un périphérique MODBUS.

function **modbusWriteRegister**(**slaveNo**, **pduAddr**, **value**)

Cette méthode utilise le code de fonction MODBUS 0x06 (Write Single Register).

Paramètres :

slaveNo addresse du périphérique MODBUS esclave à piloter

pduAddr adresse relative du registre à modifier (indexé à partir de zéro).

value la valeur 16 bits à appliquer

Retourne :

le nombre de registres affectés sur le périphérique (1)

En cas d'erreur, déclenche une exception ou retourne zéro.

serialport→modbusWriteRegisters() YSerialPort
serialport.modbusWriteRegisters()
serialport.modbusWriteRegisters()

Modifie l'état de plusieurs registres internes 16 bits contigus sur un périphérique MODBUS.

function **modbusWriteRegisters**(**slaveNo**, **pduAddr**, **values**)

Cette méthode utilise le code de fonction MODBUS 0x10 (Write Multiple Registers).

Paramètres :

slaveNo addresse du périphérique MODBUS esclave à piloter

pduAddr adresse relative du premier registre interne à modifier (indexé à partir de zéro).

values vecteur de valeurs 16 bits à appliquer

Retourne :

le nombre de registres affectés sur le périphérique

En cas d'erreur, déclenche une exception ou retourne zéro.

serialport→muteValueCallbacks() YSerialPort
serialport.muteValueCallbacks()
serialport.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→nextSerialPort()serialport.nextSerialPort() YSerialPort serialport.nextSerialPort()

Continue l'énumération des le port série commencée à l'aide de `yFirstSerialPort()`.

function **nextSerialPort**( )

Retourne :

un pointeur sur un objet `YSerialPort` accessible en ligne, ou `null` lorsque l'énumération est terminée.

serialport→queryLine()serialport.queryLine() YSerialPort serialport.queryLine()

Envoie un message sous forme de ligne de texte sur le port série, et lit la réponse reçue.

```
function queryLine( query, maxWait)
```

Cette fonction est prévue pour être utilisée lorsque le module est configuré en protocole 'Line'.

Paramètres :

query le message à envoyer (sans le retour de chariot)

maxWait le temps maximum d'attente pour obtenir une réponse (en millisecondes).

Retourne :

la première ligne de texte reçue après l'envoi du message. Les lignes suivantes peuvent être obtenues avec des appels à readLine ou readMessages.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→queryMODBUS() YSerialPort
serialport.queryMODBUS()serialport.queryMODBUS()

Envoie un message à un périphérique MODBUS esclave connecté au port série, et lit la réponse reçue.

function **queryMODBUS**(**slaveNo**, **pduBytes**)

Le contenu du message est le PDU, fourni sous forme de vecteur d'octets.

Paramètres :

slaveNo addresse du périphérique MODBUS esclave

pduBytes message à envoyer (PDU), sous forme de vecteur d'octets. Le premier octet du PDU est le code de fonction MODBUS.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un tableau vide (ou une réponse d'erreur).

serialport→readArray()serialport.readArray() YSerialPort serialport.readArray()

Lit le contenu du tampon de réception sous forme de liste d'octets, à partir de la position courante dans le flux de donnée.

```
function readArray( nChars)
```

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction ne retournera que les données disponibles.

Paramètres :

nChars le nombre maximum de bytes à lire

Retourne :

une liste de bytes avec le contenu du tampon de réception.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→readBin()serialport.readBin() YSerialPort serialport.readBin()

Lit le contenu du tampon de réception sous forme d'objet binaire, à partir de la position courante dans le flux de donnée.

```
function readBin( nChars)
```

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction ne retournera que les données disponibles.

Paramètres :

nChars le nombre maximum de bytes à lire

Retourne :

un objet binaire avec le contenu du tampon de réception.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→readByte()serialport.readByte() YSerialPort serialport.readByte()

Lit le prochain byte dans le tampon de réception, à partir de la position courante dans le flux de donnée.

function **readByte**( )

Si le contenu à la position n'est plus disponible dans le tampon de réception, ou si aucun octet n'est disponible pour l'instant, la fonction retourne YAPI_NO_MORE_DATA.

Retourne :
le prochain byte

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→readHex()serialport.readHex() YSerialPort serialport.readHex()

Lit le contenu du tampon de réception sous forme hexadécimale, à partir de la position courante dans le flux de donnée.

```
function readHex( nBytes)
```

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction ne retournera que les données disponibles.

Paramètres :

nBytes le nombre maximal d'octets à lire

Retourne :

une chaîne de caractère avec le contenu du tampon de réception, encodé en hexadécimal

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→readLine()serialport.readLine() YSerialPort serialport.readLine()

Lit la prochaine ligne (ou le prochain message) du tampon de réception, à partir de la position courante dans le flux de donnée.

```
function readLine( )
```

Cette fonction est destinée à être utilisée lorsque le module est configuré pour un protocole basé message, comme en mode 'Line' ou en protocole 'Frame'.

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction retournera la plus ancienne ligne disponible et déplacera le pointeur de position juste après. Si aucune nouvelle ligne entière n'est disponible, la fonction retourne un chaîne vide.

Retourne :
une chaîne de caractère avec une ligne de texte

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→readMessages() YSerialPort
serialport.readMessages()serialport.readMessages()

Cherche les messages entrants dans le tampon de réception correspondant à un format donné, à partir de la position courante.

function **readMessages**(**pattern**, **maxWait**)

Cette fonction ne compare et ne retourne que les caractères imprimables. Les protocoles binaires sont gérés sous forme de représentation hexadécimale.

La recherche retourne tous les messages trouvés qui correspondent au format. Tant qu'aucun message adéquat n'est trouvé, la fonction attendra, au maximum pour le temps spécifié en argument (en millisecondes).

Paramètres :

pattern une expression régulière limitée décrivant le format de message désiré, ou une chaîne vide si aucun filtrage des messages n'est désiré. Pour les protocoles binaires, le format est appliqué à la représentation hexadécimale du message.

maxWait le temps maximum d'attente pour obtenir un message, tant qu'aucun n'est trouvé dans le tampon de réception (en millisecondes).

Retourne :

un tableau de chaînes de caractères contenant les messages trouvés. Les messages binaires sont convertis automatiquement en représentation hexadécimale.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

serialport→readStr()serialport.readStr() YSerialPort serialport.readStr()

Lit le contenu du tampon de réception sous forme de string, à partir de la position courante dans le flux de donnée.

```
function readStr( nChars)
```

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction ne retournera que les données disponibles.

Paramètres :

nChars le nombre maximum de caractères à lire

Retourne :

une chaîne de caractère avec le contenu du tampon de réception.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→read_avail()serialport.read_avail() YSerialPort serialport.read_avail()

Retourne le nombre de bytes prêts à être lus dans le tampon de réception, depuis la position courante dans le flux de donnée utilisé par l'objet d'API.

```
function read_avail( )
```

Retourne :
le nombre d'octets prêts à être lus

serialport→read_seek()serialport.read_seek() YSerialPort serialport.read_seek()

Change le pointeur de position courante dans le flux de donnée à la valeur spécifiée.

```
function read_seek( absPos)
```

Cette fonction n'a pas d'effet sur le module, elle ne fait que changer la valeur stockée dans l'objet d'API qui sera utilisée pour les prochaines operations de lecture.

Paramètres :

absPos index de position absolue pour les opérations de lecture suivantes.

Retourne :

rien du tout.

serialport→read_tell()serialport.read_tell() YSerialPort serialport.read_tell()

Retourne la valeur actuelle du pointeur de position courante dans le flux de donnée utilisé par l'objet d'API.

function **read_tell**( )

Retourne :
l'index de position absolue pour les prochaines opérations de lecture.

serialport→registerValueCallback() YSerialPort
serialport.registerValueCallback()
serialport.registerValueCallback()

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.

serialport→reset()serialport.reset()serialport.reset() YSerialPort

Remet à zéro tous les compteurs et efface les tampons.

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.

serialport→selectJob()serialport.selectJob() YSerialPort serialport.selectJob()

Charge et execute le fichier de tâche spécifié.

function **selectJob**(**jobfile**)

Le fichier doit avoir été préalablement créé en utilisant l'interface graphique, ou téléchargé sur le module à l'aide de la fonction `uploadJob()`.

Paramètres :

jobfile nom du fichier de tâche (fichier sur 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.

serialport→set_RTS() YSerialPort
serialport→setRTS()serialport.set_RTS()
serialport.set_RTS()

Change manuellement l'état de la ligne RTS.

function **set_RTS**(**val**)

Cette fonction n'a pas d'effet lorsque le contrôle du flux par CTS/RTS est actif, car la ligne RTS est alors pilotée automatiquement.

Paramètres :

val 1 pour activer la ligne RTS, 0 pour la désactiver

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→set_currentJob() YSerialPort
serialport→setCurrentJob()
serialport.set_currentJob()
serialport.set_currentJob()

Modifie le nom du job à exécuter au démarrage du module.

function **set_currentJob**(**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 du job à exécuter 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.

serialport→set_logicalName() YSerialPort
serialport→setLogicalName()
serialport.set_logicalName()
serialport.set_logicalName()

Modifie le nom logique du port série.

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 série.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→set_protocol() YSerialPort
serialport→setProtocol()serialport.set_protocol()
serialport.set_protocol()

Modifie le type de protocol utilisé sur la communication série.

function **set_protocol**(**newval**)

Les valeurs possibles sont "Line" pour des messages ASCII séparés par des retours de ligne, "Frame:[timeout]ms" pour des messages binaires séparés par une temporisation, "Modbus-ASCII" pour des messages MODBUS en mode ASCII, "Modbus-RTU" pour des messages MODBUS en mode RTU, "Wiegand-ASCII" pour des messages Wiegand en mode ASCII, "Wiegand-26","Wiegand-34", etc pour des messages Wiegand en mode octet, "Char" pour un flux ASCII continu ou "Byte" pour un flux binaire continue. Le suffixe "/[wait]ms" peut être ajouté pour réduire la cadence d'émission de sorte à ce qu'il y ait au minimum le nombre spécifié de millisecondes d'intervalle entre l'envoi de chaque byte.

Paramètres :

newval une chaîne de caractères représentant le type de protocol utilisé sur la communication série

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→set_serialMode() YSerialPort
serialport→setSerialMode()
serialport.set_serialMode()
serialport.set_serialMode()

Modifie les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1".

function **set_serialMode**(**newval**)

La chaîne contient le taux de transfert, le nombre de bits de données, la parité parité et le nombre de bits d'arrêt. Un suffixe supplémentaire optionnel peut être inclus pour activer une option de contrôle de flux: "CtsRts" pour le contrôle de flux matériel, "XOnXOff" pour le contrôle de flux logique et "Simplex" pour l'utilisation du signal RTS pour l'acquisition d'un bus partagé (tel qu'utilisé pour certains adaptateurs RS485 par exemple).

Paramètres :

newval une chaîne de caractères représentant les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1"

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→set_startupJob() YSerialPort
serialport→setStartupJob()
serialport.set_startupJob()serialport.set_startupJob()

Modifie le nom du job à exécuter au démarrage du module.

function **set_startupJob**(**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 du job à exécuter 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.

serialport→set_userData() YSerialPort
serialport→setUserData()serialport.set_userData()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

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

serialport→set_voltageLevel() YSerialPort
serialport→setVoltageLevel()
serialport.set_voltageLevel()
serialport.set_voltageLevel()

Modifie le niveau de tension utilisé par le module sur le port série.

function **set_voltageLevel**(**newval**)

Les valeurs valides dépendent du modèle de module Yoctopuce hébergeant le port série. Consultez la documentation de votre module pour savoir quelles valeurs sont supportées. Affecter une valeur invalide n'aura aucun effet.

Paramètres :

newval une valeur parmi Y_VOLTAGELEVEL_OFF, Y_VOLTAGELEVEL_TTL3V, Y_VOLTAGELEVEL_TTL3VR, Y_VOLTAGELEVEL_TTL5V, Y_VOLTAGELEVEL_TTL5VR, Y_VOLTAGELEVEL_RS232 et Y_VOLTAGELEVEL_RS485 représentant le niveau de tension utilisé par le module sur le port série

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→unmuteValueCallbacks() YSerialPort
serialport.unmuteValueCallbacks()
serialport.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→uploadJob()serialport.uploadJob() YSerialPort serialport.uploadJob()

Sauvegarde une définition de tâche (au format JSON) dans un fichier.

function **uploadJob**(**jobfile**, **jsonDef**)

Le fichier peut ensuite être activé à l'aide de la méthode `selectJob()`.

Paramètres :

jobfile nom du fichier de tâche sur le module

jsonDef une chaîne de caractères contenant la définition du job en JSON

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→wait_async()serialport.wait_async() YSerialPort

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.

serialport→writeArray()serialport.writeArray() YSerialPort serialport.writeArray()

Envoie une séquence d'octets (fournie sous forme d'une liste) sur le port série.

function **writeArray**(**byteList**)

Paramètres :

byteList la liste d'octets à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→writeBin()serialport.writeBin() YSerialPort serialport.writeBin()

Envoie un objet binaire tel quel sur le port série.

```
function writeBin( buff)
```

Paramètres :

buff l'objet binaire à envoyer

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→writeByte()serialport.writeByte() YSerialPort serialport.writeByte()

Envoie un unique byte sur le port série.

```
function writeByte( code)
```

Paramètres :

code le byte à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→writeHex()serialport.writeHex() YSerialPort serialport.writeHex()

Envoie une séquence d'octets (fournie sous forme de chaîne hexadécimale) sur le port série.

```
function writeHex( hexString)
```

Paramètres :

hexString la chaîne hexadécimale à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→writeLine()serialport.writeLine() YSerialPort serialport.writeLine()

Envoie une chaîne de caractères sur le port série, suivie d'un saut de ligne (CR LF).

function **writeLine**(**text**)

Paramètres :

text la chaîne de caractères à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→writeMODBUS()serialport.writeMODBUS() YSerialPort serialport.writeMODBUS()

Envoie une commande MODBUS (fournie sous forme de chaîne hexadécimale) sur le port série.

function **writeMODBUS**(**hexString**)

Le message doit commencer par l'adresse de destination. Le CRC (ou LRC) MODBUS est ajouté automatiquement par la fonction. Cette fonction n'attend pas de réponse.

Paramètres :

hexString le message à envoyer, en hexadécimal, sans le CRC/LRC

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→writeStr()serialport.writeStr() YSerialPort serialport.writeStr()

Envoie une chaîne de caractères telle quelle sur le port série.

function **writeStr**(**text**)

Paramètres :

text la chaîne de caractères à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.61. 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'interval de temps dans lequel le mouvement doit être fait, de sorte à pouvoir synchroniser un mouvement sur plusieurs servos.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_servo.js'></script>
cpp	#include "yocto_servo.h"
m	#import "yocto_servo.h"
pas	uses yocto_servo;
vb	yocto_servo.vb
cs	yocto_servo.cs
java	import com.yoctopuce.YoctoAPI.YServo;
uwp	import com.yoctopuce.YoctoAPI.YServo;
py	from yocto_servo import *
php	require_once('yocto_servo.php');
es	in HTML: <script src="../../lib/yocto_servo.js"></script> in node.js: require('yoctolib-es2017/yocto_servo.js');

Fonction globales
yFindServo(**func**) Permet de retrouver un servo d'après un identifiant donné.
yFindServoInContext(**yctx**, **func**) Permet de retrouver un servo d'après un identifiant donné dans un Context YAPI.
yFirstServo() Commence l'énumération des servo accessibles par la librairie.
yFirstServoInContext(**yctx**) Commence l'énumération des servo accessibles par la librairie.
Méthodes des objets YServo
servo→clearCache() Invalide le cache.
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→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

servo→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.

YServo.FindServo() YServo yFindServo()YServo.FindServo()YServo.FindServo()

Permet de retrouver un servo d'après un identifiant donné.

```
function FindServo( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindServoInContext() YServo yFindServoInContext()YServo.FindServoInContext() YServo.FindServoInContext()

Permet de retrouver un servo d'après un identifiant donné dans un Context YAPI.

```
function FindServoInContext( yctx, 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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le servo sans ambiguïté

Retourne :

un objet de classe `YServo` qui permet ensuite de contrôler le servo.

YServo.FirstServo() YServo yFirstServo()YServo.FirstServo()YServo.FirstServo()

Commence l'énumération des servo accessibles par la librairie.

function **FirstServo**( )

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.

YServo.FirstServoInContext() YServo yFirstServoInContext()YServo.FirstServoInContext() YServo.FirstServoInContext()

Commence l'énumération des servo accessibles par la librairie.

function **FirstServoInContext**(**yctx**)

Utiliser la fonction `YServo.nextServo()` pour itérer sur les autres servo.

Paramètres :

yctx un contexte YAPI.

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→clearCache()servo.clearCache() YServo

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du servo. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le servo (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

servo→get_advertisedValue() YServo
servo→advertisedValue()servo.get_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()
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()
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'identifant 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 example `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 example 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()
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_neutral() YServo
servo→neutral()servo.get_neutral()
servo.get_neutral()

Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.

function **get_neutral**( )

Retourne :
un entier représentant la durée en microsecondes de l'impulsion correspondant au neutre du servo

En cas d'erreur, déclenche une exception ou retourne `Y_NEUTRAL_INVALID`.

servo→get_position() YServo
servo→position()servo.get_position()
servo.get_position()

Retourne la position courante du servo.

function **get_position**( )

Retourne :
un entier représentant la position courante du servo

En cas d'erreur, déclenche une exception ou retourne `Y_POSITION_INVALID`.

servo→get_positionAtPowerOn() YServo
servo→positionAtPowerOn()
servo.get_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() YServo
servo→range()servo.get_range()servo.get_range()

Retourne la plage d'utilisation du servo.

function **get_range**( )

Retourne :
un entier représentant la plage d'utilisation du servo

En cas d'erreur, déclenche une exception ou retourne `Y_RANGE_INVALID`.

servo→get_userData() YServo
servo→userData()servo.get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

function **get_userData**( )

Cet attribut n'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→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 occasionellement 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→loadAttribute()servo.loadAttribute() YServo servo.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

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→muteValueCallbacks() YServo
servo.muteValueCallbacks()
servo.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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 servo.nextServo()

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() YServo
servo.registerValueCallback()
servo.registerValueCallback()

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()
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() YServo
servo→setEnabledAtPowerOn()
servo.set_enabledAtPowerOn()
servo.set_enabledAtPowerOn()

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()
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()
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. N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

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()
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() YServo
servo→setPositionAtPowerOn()
servo.set_positionAtPowerOn()
servo.set_positionAtPowerOn()

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()
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. N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

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→unmuteValueCallbacks() YServo
servo.unmuteValueCallbacks()
servo.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.62. Interface de la fonction SpiPort

La fonction SpiPort permet de piloter entièrement un module d'interface SPI Yoctopuce, pour envoyer et recevoir des données et configurer les paramètres de transmission (vitesse, nombre de bits, parité, contrôle de flux et protocole). Notez que les interfaces SPI Yoctopuce ne sont pas des visibles comme des ports COM virtuels. Ils sont faits pour être utilisés comme tous les autres modules Yoctopuce.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_spiport.js'></script>
cpp	#include "yocto_spiport.h"
m	#import "yocto_spiport.h"
pas	uses yocto_spiport;
vb	yocto_spiport.vb
cs	yocto_spiport.cs
java	import com.yoctopuce.YoctoAPI.YSpiPort;
uwp	import com.yoctopuce.YoctoAPI.YSpiPort;
py	from yocto_spiport import *
php	require_once('yocto_spiport.php');
es	in HTML: <script src="../../lib/yocto_spiport.js"></script> in node.js: require('yoctolib-es2017/yocto_spiport.js');

Fonction globales
yFindSpiPort(**func**) Permet de retrouver une port SPI d'après un identifiant donné.
yFindSpiPortInContext(**yctx**, **func**) Permet de retrouver une port SPI d'après un identifiant donné dans un Context YAPI.
yFirstSpiPort() Commence l'énumération des le port SPI accessibles par la librairie.
yFirstSpiPortInContext(**yctx**) Commence l'énumération des le port SPI accessibles par la librairie.
Méthodes des objets `YSpiPort`
spiport→clearCache() Invalide le cache.
spiport→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du port SPI au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
spiport→get_advertisedValue() Retourne la valeur courante du port SPI (pas plus de 6 caractères).
spiport→get_currentJob() Retourne le nom du fichier de tâches actif en ce moment.
spiport→get_errCount() Retourne le nombre d'erreurs de communication détectées depuis la dernière mise à zéro.
spiport→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port SPI.
spiport→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port SPI.
spiport→get_friendlyName() Retourne un identifiant global du port SPI au format `NOM_MODULE.NOM_FONCTION`.

spiport→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

spiport→get_functionId()

Retourne l'identifiant matériel du port SPI, sans référence au module.

spiport→get_hardwareId()

Retourne l'identifiant matériel unique du port SPI au format SERIAL.FUNCTIONID.

spiport→get_lastMsg()

Retourne le dernier message reçu (pour les protocoles de type Line, Frame).

spiport→get_logicalName()

Retourne le nom logique du port SPI.

spiport→get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

spiport→get_module_async(**callback**, **context**)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

spiport→get_protocol()

Retourne le type de protocole utilisé sur la communication série, sous forme d'une chaîne de caractères.

spiport→get_rxCount()

Retourne le nombre d'octets reçus depuis la dernière mise à zéro.

spiport→get_rxMsgCount()

Retourne le nombre de messages reçus depuis la dernière mise à zéro.

spiport→get_shitftSampling()

Retourne vrai si la ligne SDI est déphasée par rapport à la ligne SDO.

spiport→get_spiMode()

Retourne les paramètres de communication du port, sous forme d'une chaîne de caractères du type "125000,0,msb".

spiport→get_ssPolarity()

Retourne la polarité de la ligne Slave Select (SS).

spiport→get_startupJob()

Retourne le nom du fichier de tâches à exécuter au démarrage du module.

spiport→get_txCount()

Retourne le nombre d'octets transmis depuis la dernière mise à zéro.

spiport→get_txMsgCount()

Retourne le nombre de messages envoyés depuis la dernière mise à zéro.

spiport→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

spiport→get_voltageLevel()

Retourne le niveau de tension utilisé par le module sur le port série.

spiport→isOnline()

Vérifie si le module hébergeant le port SPI est joignable, sans déclencher d'erreur.

spiport→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le port SPI est joignable, sans déclencher d'erreur.

spiport→load(**msValidity**)

Met en cache les valeurs courantes du port SPI, avec une durée de validité spécifiée.

spiport→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

spiport→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du port SPI, avec une durée de validité spécifiée.

spiport→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

spiport→nextSpiPort()

Continue l'énumération des le port SPI commencée à l'aide de `yFirstSpiPort()`.

spiport→queryLine(query, maxWait)

Envoie un message sous forme de ligne de texte sur le port série, et lit la réponse reçue.

spiport→readArray(nChars)

Lit le contenu du tampon de réception sous forme de liste d'octets, à partir de la position courante dans le flux de donnée.

spiport→readBin(nChars)

Lit le contenu du tampon de réception sous forme d'objet binaire, à partir de la position courante dans le flux de donnée.

spiport→readByte()

Lit le prochain byte dans le tampon de réception, à partir de la position courante dans le flux de donnée.

spiport→readHex(nBytes)

Lit le contenu du tampon de réception sous forme hexadécimale, à partir de la position courante dans le flux de donnée.

spiport→readLine()

Lit la prochaine ligne (ou le prochain message) du tampon de réception, à partir de la position courante dans le flux de donnée.

spiport→readMessages(pattern, maxWait)

Cherche les messages entrants dans le tampon de réception correspondant à un format donné, à partir de la position courante.

spiport→readStr(nChars)

Lit le contenu du tampon de réception sous forme de string, à partir de la position courante dans le flux de donnée.

spiport→read_avail()

Retourne le nombre de bytes prêts à être lus dans le tampon de réception, depuis la position courante dans le flux de donnée utilisé par l'objet d'API.

spiport→read_seek(absPos)

Change le pointeur de position courante dans le flux de donnée à la valeur spécifiée.

spiport→read_tell()

Retourne la valeur actuelle du pointeur de position courante dans le flux de donnée utilisé par l'objet d'API.

spiport→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

spiport→reset()

Remet à zéro tous les compteurs et efface les tampons.

spiport→selectJob(jobfile)

Charge et execute le fichier de tâche spécifié.

spiport→set_SS(val)

Change manuellement l'état de la ligne SS.

spiport→set_currentJob(newval)

Modifie le nom du job à exécuter au démarrage du module.

spiport→set_logicalName(newval)

Modifie le nom logique du port SPI.

spiport→set_protocol(newval)

Modifie le type de protocol utilisé sur la communication série.

spiport→set_shitftSampling(newval)

Modifie le déphasage de l'échanillonage de SDI par rapport à SDO.

spiport→set_spiMode(newval)

Modifie les paramètres de communication du port, sous forme d'une chaîne de caractères du type "125000,0,msb".

spiport→set_ssPolarity(newval)

Modifie la polarité de la ligne Slave Select (SS).

spiport→set_startupJob(newval)

Modifie le nom du job à exécuter au démarrage du module.

spiport→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`.

spiport→set_voltageLevel(newval)

Modifie le niveau de tension utilisé par le module sur le port série.

spiport→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

spiport→uploadJob(jobfile, jsonDef)

Sauvegarde une définition de tâche (au format JSON) dans un fichier.

spiport→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.

spiport→writeArray(byteList)

Envoie une séquence d'octets (fournie sous forme d'une liste) sur le port série.

spiport→writeBin(buff)

Envoie un objet binaire tel quel sur le port série.

spiport→writeByte(code)

Envoie un unique byte sur le port série.

spiport→writeHex(hexString)

Envoie une séquence d'octets (fournie sous forme de chaîne hexadécimale) sur le port série.

spiport→writeLine(text)

Envoie une chaîne de caractères sur le port série, suivie d'un saut de ligne (CR LF).

spiport→writeStr(text)

Envoie une chaîne de caractères telle quelle sur le port série.

YSpiPort.FindSpiPort() YSpiPort yFindSpiPort()YSpiPort.FindSpiPort() YSpiPort.FindSpiPort()

Permet de retrouver une port SPI d'après un identifiant donné.

```
function FindSpiPort( 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 SPI soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YSpiPort.isOnline()` pour tester si le port SPI 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le port SPI sans ambiguïté

Retourne :

un objet de classe `YSpiPort` qui permet ensuite de contrôler le port SPI.

YSpiPort.FindSpiPortInContext() YSpiPort
yFindSpiPortInContext()
YSpiPort.FindSpiPortInContext()
YSpiPort.FindSpiPortInContext()

Permet de retrouver une port SPI d'après un identifiant donné dans un Context YAPI.

function **FindSpiPortInContext**(**yctx**, **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 SPI soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YSpiPort.isOnline()` pour tester si le port SPI 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le port SPI sans ambiguïté

Retourne :

un objet de classe `YSpiPort` qui permet ensuite de contrôler le port SPI.

YSpiPort.FirstSpiPort() YSpiPort yFirstSpiPort()YSpiPort.FirstSpiPort() YSpiPort.FirstSpiPort()

Commence l'énumération des le port SPI accessibles par la librairie.

function **FirstSpiPort**( )

Utiliser la fonction `YSpiPort.nextSpiPort()` pour itérer sur les autres le port SPI.

Retourne :

un pointeur sur un objet `YSpiPort`, correspondant au premier port SPI accessible en ligne, ou `null` si il n'y a pas du port SPI disponibles.

YSpiPort.FirstSpiPortInContext() YSpiPort
yFirstSpiPortInContext()
YSpiPort.FirstSpiPortInContext()
YSpiPort.FirstSpiPortInContext()

Commence l'énumération des le port SPI accessibles par la librairie.

function **FirstSpiPortInContext**(**yctx**)

Utiliser la fonction `YSpiPort.nextSpiPort()` pour itérer sur les autres le port SPI.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YSpiPort`, correspondant au premier port SPI accessible en ligne, ou `null` si il n'y a pas du port SPI disponibles.

spiport→clearCache()spiport.clearCache() YSpiPort

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du port SPI. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

spiport→describe()spiport.describe() YSpiPort

Retourne un court texte décrivant de manière non-ambigüe l'instance du port SPI au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le port SPI (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

spiport→get_advertisedValue() YSpiPort
spiport→advertisedValue()
spiport.get_advertisedValue()
spiport.get_advertisedValue()

Retourne la valeur courante du port SPI (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante du port SPI (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

spiport→get_currentJob() YSpiPort
spiport→currentJob()spiport.get_currentJob()
spiport.get_currentJob()

Retourne le nom du fichier de tâches actif en ce moment.

function **get_currentJob**( )

Retourne :
une chaîne de caractères représentant le nom du fichier de tâches actif en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTJOB_INVALID`.

spiport→get_errCount() YSpiPort
spiport→errCount()spiport.get_errCount()
spiport.get_errCount()

Retourne le nombre d'erreurs de communication détectées depuis la dernière mise à zéro.

function **get_errCount**( )

Retourne :
un entier représentant le nombre d'erreurs de communication détectées depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne `Y_ERRCOUNT_INVALID`.

spiport→get_errorMessage() YSpiPort
spiport→errorMessage()spiport.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port SPI.

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 SPI.

spiport→get_errorType() YSpiPort
spiport→errorType()spiport.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port SPI.

```
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 SPI.

spiport→get_friendlyName() YSpiPort
spiport→friendlyName()spiport.get_friendlyName()

Retourne un identifiant global du port SPI au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du port SPI si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du port SPI (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le port SPI en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

spiport→get_functionDescriptor() YSpiPort
spiport→functionDescriptor()
spiport.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

spiport→get_functionId() YSpiPort
spiport→functionId()spiport.get_functionId()

Retourne l'identifiant matériel du port SPI, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le port SPI (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

spiport→get_hardwareId() YSpiPort
spiport→hardwareId()spiport.get_hardwareId()

Retourne l'identifiant matériel unique du port SPI 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 SPI (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le port SPI (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

spiport→get_lastMsg() YSpiPort
spiport→lastMsg()spiport.get_lastMsg()
spiport.get_lastMsg()

Retourne le dernier message reçu (pour les protocoles de type Line, Frame).

function **get_lastMsg**( )

Retourne :
une chaîne de caractères représentant le dernier message reçu (pour les protocoles de type Line, Frame)

En cas d'erreur, déclenche une exception ou retourne `Y_LASTMSG_INVALID`.

spiport→get_logicalName() YSpiPort
spiport→logicalName()spiport.get_logicalName()
spiport.get_logicalName()

Retourne le nom logique du port SPI.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du port SPI.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

spiport→get_module() YSpiPort
spiport→module()spiport.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

spiport→get_protocol() YSpiPort
spiport→protocol()spiport.get_protocol()
spiport.get_protocol()

Retourne le type de protocole utilisé sur la communication série, sous forme d'une chaîne de caractères.

function **get_protocol**( )

Les valeurs possibles sont "Line" pour des messages ASCII séparés par des retours de ligne, "Frame:[timeout]ms" pour des messages binaires séparés par une temporisation, "Char" pour un flux ASCII continu ou "Byte" pour un flux binaire continue.

Retourne :
une chaîne de caractères représentant le type de protocole utilisé sur la communication série, sous forme d'une chaîne de caractères

En cas d'erreur, déclenche une exception ou retourne `Y_PROTOCOL_INVALID`.

spiport→get_rxCount() YSpiPort
spiport→rxCount()spiport.get_rxCount()
spiport.get_rxCount()

Retourne le nombre d'octets reçus depuis la dernière mise à zéro.

function **get_rxCount**( )

Retourne :
un entier représentant le nombre d'octets reçus depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne `Y_RXCOUNT_INVALID`.

spiport→get_rxMsgCount() YSpiPort
spiport→rxMsgCount()spiport.get_rxMsgCount()
spiport.get_rxMsgCount()

Retourne le nombre de messages reçus depuis la dernière mise à zéro.

function **get_rxMsgCount**( )

Retourne :
un entier représentant le nombre de messages reçus depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne `Y_RXMSGCOUNT_INVALID`.

spiport→get_shitftSampling() YSpiPort
spiport→shitftSampling()spiport.get_shitftSampling()
spiport.get_shitftSampling()

Retourne vrai si la ligne SDI est déphasée par rapport à la ligne SDO.

function **get_shitftSampling**( )

Retourne :

soit `Y_SHITFTSAMPLING_OFF`, soit `Y_SHITFTSAMPLING_ON`, selon vrai si la ligne SDI est déphasée par rapport à la ligne SDO

En cas d'erreur, déclenche une exception ou retourne `Y_SHITFTSAMPLING_INVALID`.

spiport→get_spiMode() YSpiPort
spiport→spiMode()spiport.get_spiMode()
spiport.get_spiMode()

Retourne les paramètres de communication du port, sous forme d'une chaîne de caractères du type "125000,0,msb".

function **get_spiMode**( )

La chaîne contient le taux de transfert désiré, le mode SPI (entre 0 et 3) et l'ordre des bits.

Retourne :
une chaîne de caractères représentant les paramètres de communication du port, sous forme d'une chaîne de caractères du type "125000,0,msb"

En cas d'erreur, déclenche une exception ou retourne `Y_SPIMODE_INVALID`.

spiport→get_ssPolarity() YSpiPort
spiport→ssPolarity()spiport.get_ssPolarity()
spiport.get_ssPolarity()

Retourne la polarité de la ligne Slave Select (SS).

function **get_ssPolarity**( )

Retourne :

soit Y_SSPOLARITY_ACTIVE_LOW, soit Y_SSPOLARITY_ACTIVE_HIGH, selon la polarité de la ligne Slave Select (SS)

En cas d'erreur, déclenche une exception ou retourne Y_SSPOLARITY_INVALID.

spiport→get_startupJob() YSpiPort
spiport→startupJob()spiport.get_startupJob()
spiport.get_startupJob()

Retourne le nom du fichier de tâches à exécuter au démarrage du module.

function **get_startupJob**( )

Retourne :
une chaîne de caractères représentant le nom du fichier de tâches à exécuter au démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_STARTUPJOB_INVALID`.

spiport→get_txCount() YSpiPort
spiport→txCount()spiport.get_txCount()
spiport.get_txCount()

Retourne le nombre d'octets transmis depuis la dernière mise à zéro.

function **get_txCount**( )

Retourne :
un entier représentant le nombre d'octets transmis depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne `Y_TXCOUNT_INVALID`.

spiport→get_txMsgCount() YSpiPort
spiport→txMsgCount()spiport.get_txMsgCount()
spiport.get_txMsgCount()

Retourne le nombre de messages envoyés depuis la dernière mise à zéro.

function **get_txMsgCount**( )

Retourne :
un entier représentant le nombre de messages envoyés depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne `Y_TXMSGCOUNT_INVALID`.

spiport→get_userData() YSpiPort
spiport→userData()spiport.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.

spiport→get_voltageLevel() YSpiPort
spiport→voltageLevel()spiport.get_voltageLevel()
spiport.get_voltageLevel()

Retourne le niveau de tension utilisé par le module sur le port série.

function **get_voltageLevel**( )

Retourne :

une valeur parmi Y_VOLTAGELEVEL_OFF, Y_VOLTAGELEVEL_TTL3V, Y_VOLTAGELEVEL_TTL3VR, Y_VOLTAGELEVEL_TTL5V, Y_VOLTAGELEVEL_TTL5VR, Y_VOLTAGELEVEL_RS232 et Y_VOLTAGELEVEL_RS485 représentant le niveau de tension utilisé par le module sur le port série

En cas d'erreur, déclenche une exception ou retourne Y_VOLTAGELEVEL_INVALID.

spiport→isOnline()spiport.isOnline() YSpiPort

Vérifie si le module hébergeant le port SPI est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du port SPI sont valides au moment de l'appel, le module est considéré joignable. 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 SPI est joignable, `false` sinon

spiport→load()spiport.load() YSpiPort

Met en cache les valeurs courantes du port SPI, 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 occasionellement 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.

spiport→loadAttribute()spiport.loadAttribute() YSpiPort spiport.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

spiport→muteValueCallbacks() YSpiPort
spiport.muteValueCallbacks()
spiport.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→nextSpiPort()spiport.nextSpiPort() YSpiPort spiport.nextSpiPort()

Continue l'énumération des le port SPI commencée à l'aide de `yFirstSpiPort()`.

function **nextSpiPort**( )

Retourne :

un pointeur sur un objet `YSpiPort` accessible en ligne, ou `null` lorsque l'énumération est terminée.

spiport→queryLine()spiport.queryLine() YSpiPort spiport.queryLine()

Envoie un message sous forme de ligne de texte sur le port série, et lit la réponse reçue.

```
function queryLine( query, maxWait)
```

Cette fonction est prévue pour être utilisée lorsque le module est configuré en protocole 'Line'.

Paramètres :

query le message à envoyer (sans le retour de chariot)

maxWait le temps maximum d'attente pour obtenir une réponse (en millisecondes).

Retourne :

la première ligne de texte reçue après l'envoi du message. Les lignes suivantes peuvent être obtenues avec des appels à readLine ou readMessages.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→readArray()spiport.readArray() YSpiPort spiport.readArray()

Lit le contenu du tampon de réception sous forme de liste d'octets, à partir de la position courante dans le flux de donnée.

```
function readArray( nChars)
```

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction ne retournera que les données disponibles.

Paramètres :

nChars le nombre maximum de bytes à lire

Retourne :

une liste de bytes avec le contenu du tampon de réception.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→readBin()spiport.readBin()spiport.readBin() YSpiPort

Lit le contenu du tampon de réception sous forme d'objet binaire, à partir de la position courante dans le flux de donnée.

```
function readBin( nChars)
```

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction ne retournera que les données disponibles.

Paramètres :

nChars le nombre maximum de bytes à lire

Retourne :

un objet binaire avec le contenu du tampon de réception.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→readByte()spiport.readByte() YSpiPort spiport.readByte()

Lit le prochain byte dans le tampon de réception, à partir de la position courante dans le flux de donnée.

function **readByte**( )

Si le contenu à la position n'est plus disponible dans le tampon de réception, ou si aucun octet n'est disponible pour l'instant, la fonction retourne YAPI_NO_MORE_DATA.

Retourne :
le prochain byte

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→readHex()spiport.readHex() YSpiPort spiport.readHex()

Lit le contenu du tampon de réception sous forme hexadécimale, à partir de la position courante dans le flux de donnée.

```
function readHex( nBytes)
```

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction ne retournera que les données disponibles.

Paramètres :

nBytes le nombre maximal d'octets à lire

Retourne :

une chaîne de caractère avec le contenu du tampon de réception, encodé en hexadécimal

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→readLine()spiport.readLine() YSpiPort spiport.readLine()

Lit la prochaine ligne (ou le prochain message) du tampon de réception, à partir de la position courante dans le flux de donnée.

```
function readLine( )
```

Cette fonction est destinée à être utilisée lorsque le module est configuré pour un protocole basé message, comme en mode 'Line' ou en protocole 'Frame'.

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction retournera la plus ancienne ligne disponible et déplacera le pointeur de position juste après. Si aucune nouvelle ligne entière n'est disponible, la fonction retourne un chaîne vide.

Retourne :
une chaîne de caractère avec une ligne de texte

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→readMessages()spiport.readMessages() YSpiPort spiport.readMessages()

Cherche les messages entrants dans le tampon de réception correspondant à un format donné, à partir de la position courante.

function **readMessages**(**pattern**, **maxWait**)

Cette fonction ne compare et ne retourne que les caractères imprimables. Les protocoles binaires sont gérés sous forme de représentation hexadécimale.

La recherche retourne tous les messages trouvés qui correspondent au format. Tant qu'aucun message adéquat n'est trouvé, la fonction attendra, au maximum pour le temps spécifié en argument (en millisecondes).

Paramètres :

pattern une expression régulière limitée décrivant le format de message désiré, ou une chaîne vide si aucun filtrage des messages n'est désiré. Pour les protocoles binaires, le format est appliqué à la représentation hexadécimale du message.

maxWait le temps maximum d'attente pour obtenir un message, tant qu'aucun n'est trouvé dans le tampon de réception (en millisecondes).

Retourne :

un tableau de chaînes de caractères contenant les messages trouvés. Les messages binaires sont convertis automatiquement en représentation hexadécimale.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

spiport→readStr()spiport.readStr()spiport.readStr() YSpiPort

Lit le contenu du tampon de réception sous forme de string, à partir de la position courante dans le flux de donnée.

```
function readStr( nChars)
```

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction ne retournera que les données disponibles.

Paramètres :

nChars le nombre maximum de caractères à lire

Retourne :

une chaîne de caractère avec le contenu du tampon de réception.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→read_avail()spiport.read_avail() YSpiPort spiport.read_avail()

Retourne le nombre de bytes prêts à être lus dans le tampon de réception, depuis la position courante dans le flux de donnée utilisé par l'objet d'API.

```
function read_avail( )
```

Retourne :
le nombre d'octets prêts à être lus

spiport→read_seek()spiport.read_seek() YSpiPort spiport.read_seek()

Change le pointeur de position courante dans le flux de donnée à la valeur spécifiée.

```
function read_seek( absPos)
```

Cette fonction n'a pas d'effet sur le module, elle ne fait que changer la valeur stockée dans l'objet d'API qui sera utilisée pour les prochaines operations de lecture.

Paramètres :

absPos index de position absolue pour les opérations de lecture suivantes.

Retourne :

rien du tout.

spiport→read_tell()spiport.read_tell() YSpiPort spiport.read_tell()

Retourne la valeur actuelle du pointeur de position courante dans le flux de donnée utilisé par l'objet d'API.

```
function read_tell( )
```

Retourne :

l'index de position absolue pour les prochaines opérations de lecture.

spiport→registerValueCallback() YSpiPort
spiport.registerValueCallback()
spiport.registerValueCallback()

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.

spiport→reset()spiport.reset()spiport.reset() YSpiPort

Remet à zéro tous les compteurs et efface les tampons.

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.

spiport→selectJob()spiport.selectJob() YSpiPort spiport.selectJob()

Charge et execute le fichier de tâche spécifié.

function **selectJob**(**jobfile**)

Le fichier doit avoir été préalablement créé en utilisant l'interface graphique, ou téléchargé sur le module à l'aide de la fonction `uploadJob()`.

Paramètres :

jobfile nom du fichier de tâche (fichier sur 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.

spiport→set_SS() YSpiPort
spiport→setSS()spiport.set_SS()spiport.set_SS()

Change manuellement l'état de la ligne SS.

```
function set_SS( val)
```

Cette fonction n'a pas d'effet lorsque la gestion automatique de la ligne SS est activée.

Paramètres :

val 1 pour activer la ligne SS, 0 pour la désactiver.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→set_currentJob() YSpiPort
spiport→setCurrentJob()spiport.set_currentJob()
spiport.set_currentJob()

Modifie le nom du job à exécuter au démarrage du module.

function **set_currentJob**(**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 du job à exécuter 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.

spiport→set_logicalName() YSpiPort
spiport→setLogicalName()spiport.set_logicalName()
spiport.set_logicalName()

Modifie le nom logique du port SPI.

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 SPI.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→set_protocol() YSpiPort
spiport→setProtocol()spiport.set_protocol()
spiport.set_protocol()

Modifie le type de protocol utilisé sur la communication série.

function **set_protocol**(**newval**)

Les valeurs possibles sont "Line" pour des messages ASCII séparés par des retours de ligne, "Frame:[timeout]ms" pour des messages binaires séparés par une temporisation, "Char" pour un flux ASCII continu ou "Byte" pour un flux binaire continue. Le suffixe "/[wait]ms" peut être ajouté pour réduire la cadence d'émission de sorte à ce qu'il y ait au minimum le nombre spécifié de millisecondes d'intervalle entre l'envoi de chaque byte.

Paramètres :

newval une chaîne de caractères représentant le type de protocol utilisé sur la communication série

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→set_shitftSampling() YSpiPort
spiport→setShitftSampling()
spiport.set_shitftSampling()
spiport.set_shitftSampling()

Modifie le déphasage de l'échanillonage de SDI par rapport à SDO.

function **set_shitftSampling**(**newval**)

Lorsque le déphasage est désactivé, SDI est lu au milieu de la phase du cycle de sortie de la ligne SDO. Lorsqu'il est activé, SDI est lu à la fin du cycle de sortie de SDO.

Paramètres :

newval soit `Y_SHITFTSAMPLING_OFF`, soit `Y_SHITFTSAMPLING_ON`, selon le déphasage de l'échanillonage de SDI par rapport à SDO

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→set_spiMode() YSpiPort
spiport→setSpiMode()spiport.set_spiMode()
spiport.set_spiMode()

Modifie les paramètres de communication du port, sous forme d'une chaîne de caractères du type "125000,0,msb".

function **set_spiMode**(**newval**)

La chaîne contient le taux de transfert désiré, le mode SPI (entre 0 et 3) et l'ordre des bits.

Paramètres :

newval une chaîne de caractères représentant les paramètres de communication du port, sous forme d'une chaîne de caractères du type "125000,0,msb"

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→set_ssPolarity() YSpiPort
spiport→setSsPolarity()spiport.set_ssPolarity()
spiport.set_ssPolarity()

Modifie la polarité de la ligne Slave Select (SS).

function **set_ssPolarity**(**newval**)

Paramètres :

newval soit `Y_SSPOLARITY_ACTIVE_LOW`, soit `Y_SSPOLARITY_ACTIVE_HIGH`, selon la polarité de la ligne Slave Select (SS)

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→set_startupJob() YSpiPort
spiport→setStartupJob()spiport.set_startupJob()
spiport.set_startupJob()

Modifie le nom du job à exécuter au démarrage du module.

function **set_startupJob**(**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 du job à exécuter 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.

spiport→set_userData() YSpiPort
spiport→setUserData()spiport.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

spiport→set_voltageLevel() YSpiPort
spiport→setVoltageLevel()spiport.set_voltageLevel()
spiport.set_voltageLevel()

Modifie le niveau de tension utilisé par le module sur le port série.

function **set_voltageLevel**(**newval**)

Les valeurs valides dépendent du modèle de module Yoctopuce hébergeant le port série. Consultez la documentation de votre module pour savoir quelles valeurs sont supportées. Affecter une valeur invalide n'aura aucun effet.

Paramètres :

newval une valeur parmi Y_VOLTAGELEVEL_OFF, Y_VOLTAGELEVEL_TTL3V, Y_VOLTAGELEVEL_TTL3VR, Y_VOLTAGELEVEL_TTL5V, Y_VOLTAGELEVEL_TTL5VR, Y_VOLTAGELEVEL_RS232 et Y_VOLTAGELEVEL_RS485 représentant le niveau de tension utilisé par le module sur le port série

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→unmuteValueCallbacks() YSpiPort
spiport.unmuteValueCallbacks()
spiport.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→uploadJob()spiport.uploadJob() YSpiPort spiport.uploadJob()

Sauvegarde une définition de tâche (au format JSON) dans un fichier.

function **uploadJob**(**jobfile**, **jsonDef**)

Le fichier peut ensuite être activé à l'aide de la méthode `selectJob()`.

Paramètres :

jobfile nom du fichier de tâche sur le module

jsonDef une chaîne de caractères contenant la définition du job en JSON

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→wait_async()spiport.wait_async() YSpiPort

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.

spiport→writeArray()spiport.writeArray() YSpiPort spiport.writeArray()

Envoie une séquence d'octets (fournie sous forme d'une liste) sur le port série.

function **writeArray**(**byteList**)

Paramètres :

byteList la liste d'octets à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→writeBin()spiport.writeBin() YSpiPort spiport.writeBin()

Envoie un objet binaire tel quel sur le port série.

```
function writeBin( buff)
```

Paramètres :

buff l'objet binaire à envoyer

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→writeByte()spiport.writeByte() YSpiPort
spiport.writeByte()

Envoie un unique byte sur le port série.

function **writeByte**(**code**)

Paramètres :

code le byte à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→writeHex()spiport.writeHex() YSpiPort spiport.writeHex()

Envoie une séquence d'octets (fournie sous forme de chaîne hexadécimale) sur le port série.

function **writeHex**(**hexString**)

Paramètres :

hexString la chaîne hexadécimale à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→writeLine()spiport.writeLine() YSpiPort spiport.writeLine()

Envoie une chaîne de caractères sur le port série, suivie d'un saut de ligne (CR LF).

function **writeLine**(**text**)

Paramètres :

text la chaîne de caractères à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

spiport→writeStr()spiport.writeStr()spiport.writeStr() YSpiPort

Envoie une chaîne de caractères telle quelle sur le port série.

function **writeStr**(**text**)

Paramètres :

text la chaîne de caractères à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.63. Interface de la fonction StepperMotor

La librairie de programmation Yoctopuce permet de piloter un moteur pas à pas.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_steppermotor.js'></script>
cpp	#include "yocto_steppermotor.h"
m	#import "yocto_steppermotor.h"
pas	uses yocto_steppermotor;
vb	yocto_steppermotor.vb
cs	yocto_steppermotor.cs
java	import com.yoctopuce.YoctoAPI.YStepperMotor;
uwp	import com.yoctopuce.YoctoAPI.YStepperMotor;
py	from yocto_steppermotor import *
php	require_once('yocto_steppermotor.php');
es	in HTML: <script src="../../lib/yocto_steppermotor.js"></script> in node.js: require('yoctolib-es2017/yocto_steppermotor.js');

Fonction globales	
yFindStepperMotor(**func**)	Permet de retrouver un moteur pas à pas d'après un identifiant donné.
yFindStepperMotorInContext(**yctx**, **func**)	Permet de retrouver un moteur pas à pas d'après un identifiant donné dans un Context YAPI.
yFirstStepperMotor()	Commence l'énumération des moteur pas à pas accessibles par la librairie.
yFirstStepperMotorInContext(**yctx**)	Commence l'énumération des moteur pas à pas accessibles par la librairie.

Méthodes des objets `YStepperMotor`	
steppermotor→abortAndBrake()	Stoppe le moteur en douceur dès que possible, sans attendre la fin de la commande actuelle.
steppermotor→abortAndHiZ()	Relâche le contrôle du moteur immédiatement, sans attendre la fin de la commande actuelle.
steppermotor→alertStepOut()	Avance le moteur d'un pas dans le sens inverse du mouvement en cours lors de la dernière alerte.
steppermotor→changeSpeed(**speed**)	Lance a moteur à une vitesse spécifique.
steppermotor→clearCache()	Invalide le cache.
steppermotor→describe()	Retourne un court texte décrivant de manière non-ambigüe l'instance du moteur pas à pas au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
steppermotor→emergencyStop()	Stoppe le moteur en urgence, sans autre précaution.
steppermotor→findHomePosition(**speed**)	Lance le moteur en arrière à la vitesse spécifiée, pour chercher l'origine de l'axe.
steppermotor→get_advertisedValue()	Retourne la valeur courante du moteur pas à pas (pas plus de 6 caractères).
steppermotor→get_auxSignal()	

Retourne la valeur actuelle du signal généré sur la sortie auxiliaire.

steppermotor→get_diags()

Retourne l'état déatillé du contrôleur de moteur pas-à-pas (bitmap).

steppermotor→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moteur pas à pas.

steppermotor→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moteur pas à pas.

steppermotor→get_friendlyName()

Retourne un identifiant global du moteur pas à pas au format `NOM_MODULE.NOM_FONCTION`.

steppermotor→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

steppermotor→get_functionId()

Retourne l'identifiant matériel du moteur pas à pas, sans référence au module.

steppermotor→get_hardwareId()

Retourne l'identifiant matériel unique du moteur pas à pas au format `SERIAL.FUNCTIONID`.

steppermotor→get_logicalName()

Retourne le nom logique du moteur pas à pas.

steppermotor→get_maxAccel()

Retourne l'accélération maximale du moteur, mesurée en pas par seconde^2.

steppermotor→get_maxSpeed()

Retourne la vitesse maximale du moteur, mesurée en pas par seconde.

steppermotor→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

steppermotor→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

steppermotor→get_motorState()

Retourne l'état de fonctionnement du moteur.

steppermotor→get_overcurrent()

Retourne le seuil de déclenchement de la sécurité de dépassement de courant, mesuré en mA.

steppermotor→get_pullinSpeed()

Retourne la vitesse du moteur atteignable directement depuis l'arrêt, mesurée en pas par seconde.

steppermotor→get_speed()

Retourne la vitesse actuelle du moteur, mesurée en pas par seconde.

steppermotor→get_stepPos()

Retourne la position logique actuelle du moteur, mesurée en pas.

steppermotor→get_stepping()

Retourne le type de stepping utilisé pour piloter le moteur.

steppermotor→get_tCurrRun()

Retourne la limite de courant pour la régulation de torque en mouvement, mesurée en mA.

steppermotor→get_tCurrStop()

Retourne la limite de courant pour la régulation de torque à l'arrêt, mesurée en mA.

steppermotor→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

steppermotor→isOnline()

Vérifie si le module hébergeant le moteur pas à pas est joignable, sans déclencher d'erreur.

steppermotor→isOnline_async(callback, context)

Vérifie si le module hébergeant le moteur pas à pas est joignable, sans déclencher d'erreur.

steppermotor→load(msValidity)

Met en cache les valeurs courantes du moteur pas à pas, avec une durée de validité spécifiée.

steppermotor→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

steppermotor→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du moteur pas à pas, avec une durée de validité spécifiée.

steppermotor→moveRel(relPos)

Contrôle le moteur pour atteindre une position relative donnée.

steppermotor→moveTo(absPos)

Contrôle le moteur pour atteindre une position absolue donnée.

steppermotor→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

steppermotor→nextStepperMotor()

Continue l'énumération des moteur pas à pas commencée à l'aide de `yFirstStepperMotor()`.

steppermotor→pause(waitMs)

Garde le moteur dans le même état pour la durée spécifiée, avant d'exécuter la commande suivante.

steppermotor→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

steppermotor→reset()

Réinitialise le controlleur et quittance toutes les alertes.

steppermotor→set_auxSignal(newval)

Modifie la valeur du signal généré sur la sortie auxiliaire.

steppermotor→set_logicalName(newval)

Modifie le nom logique du moteur pas à pas.

steppermotor→set_maxAccel(newval)

Modifie l'accélération maximale du moteur, mesurée en pas par seconde^2.

steppermotor→set_maxSpeed(newval)

Modifie la vitesse maximale du moteur, mesurée en pas par seconde.

steppermotor→set_overcurrent(newval)

Modifie le seuil de déclenchement de la sécurité de dépassement de courant, mesuré en mA.

steppermotor→set_pullinSpeed(newval)

Modifie la vitesse du moteur atteignable directement depuis l'arrêt, mesurée en pas par seconde.

steppermotor→set_stepPos(newval)

Modifie la position logique actuelle du moteur, mesurée en pas.

steppermotor→set_stepping(newval)

Modifie le type de stepping utilisé pour piloter le moteur.

steppermotor→set_tCurrRun(newval)

Modifie la limite de courant pour la régulation de torque en mouvement, mesurée en mA.

steppermotor→set_tCurrStop(newval)

Modifie la limite de courant pour la régulation de torque à l'arrêt, mesurée en mA.

steppermotor→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`.

steppermotor→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

steppermotor→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.

YStepperMotor.FindStepperMotor() YStepperMotor yFindStepperMotor() YStepperMotor.FindStepperMotor() YStepperMotor.FindStepperMotor()

Permet de retrouver un moteur pas à pas d'après un identifiant donné.

```
function FindStepperMotor( 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 moteur pas à pas soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YStepperMotor.isOnline()` pour tester si le moteur pas à pas 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le moteur pas à pas sans ambiguïté

Retourne :

un objet de classe `YStepperMotor` qui permet ensuite de contrôler le moteur pas à pas.

YStepperMotor.FindStepperMotorInContext() YStepperMotor yFindStepperMotorInContext() YStepperMotor.FindStepperMotorInContext() YStepperMotor.FindStepperMotorInContext()

Permet de retrouver un moteur pas à pas d'après un identifiant donné dans un Context YAPI.

function **FindStepperMotorInContext**(**yctx**, **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 moteur pas à pas soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YStepperMotor.isOnline()` pour tester si le moteur pas à pas 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le moteur pas à pas sans ambiguïté

Retourne :

un objet de classe `YStepperMotor` qui permet ensuite de contrôler le moteur pas à pas.

YStepperMotor.FirstStepperMotor() YStepperMotor
yFirstStepperMotor()
YStepperMotor.FirstStepperMotor()
YStepperMotor.FirstStepperMotor()

Commence l'énumération des moteur pas à pas accessibles par la librairie.

function **FirstStepperMotor**( )

Utiliser la fonction `YStepperMotor.nextStepperMotor()` pour itérer sur les autres moteur pas à pas.

Retourne :

un pointeur sur un objet `YStepperMotor`, correspondant au premier moteur pas à pas accessible en ligne, ou `null` si il n'y a pas de moteur pas à pas disponibles.

YStepperMotor.FirstStepperMotorInContext() YStepperMotor yFirstStepperMotorInContext() YStepperMotor.FirstStepperMotorInContext() YStepperMotor.FirstStepperMotorInContext()

Commence l'énumération des moteur pas à pas accessibles par la librairie.

```
function FirstStepperMotorInContext( yctx)
```

Utiliser la fonction `YStepperMotor.nextStepperMotor()` pour itérer sur les autres moteur pas à pas.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YStepperMotor`, correspondant au premier moteur pas à pas accessible en ligne, ou `null` si il n'y a pas de moteur pas à pas disponibles.

steppermotor→abortAndBrake() YStepperMotor
steppermotor.abortAndBrake()
steppermotor.abortAndBrake()

Stoppe le moteur en douceur dès que possible, sans attendre la fin de la commande actuelle.

function **abortAndBrake**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→abortAndHiZ() YStepperMotor
steppermotor.abortAndHiZ()
steppermotor.abortAndHiZ()

Relâche le contrôle du moteur immédiatement, sans attendre la fin de la commande actuelle.

function **abortAndHiZ**( )

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→alertStepOut() YStepperMotor
steppermotor.alertStepOut()
steppermotor.alertStepOut()

Avance le moteur d'un pas dans le sens inverse du mouvement en cours lors de la dernière alerte.

function **alertStepOut**( )

L'avance est possible même si le système est encore en alerte (interrupteur de fin de course enclanché). Attention, utilisez cette fonction avec prudence car elle peut entraîner des dégats mécaniques !

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→changeSpeed() YStepperMotor
steppermotor.changeSpeed()
steppermotor.changeSpeed()

Lance a moteur à une vitesse spécifique.

```
function changeSpeed( speed)
```

Le temps après lequel la vitesse sera atteinte dépend des paramètres d'accélération configurés pour le moteur.

Paramètres :

speed vitesse désirée, en pas par seconde. La vitesse minimale non-nulle est de 0.001 impulsion par seconde.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→clearCache() YStepperMotor
steppermotor.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du moteur pas à pas. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

steppermotor→describe()steppermotor.describe() YStepperMotor

Retourne un court texte décrivant de manière non-ambigüe l'instance du moteur pas à pas au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le moteur pas à pas (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

steppermotor→emergencyStop() YStepperMotor
steppermotor.emergencyStop()
steppermotor.emergencyStop()

Stoppe le moteur en urgence, sans autre précaution.

function **emergencyStop**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→findHomePosition() YStepperMotor
steppermotor.findHomePosition()
steppermotor.findHomePosition()

Lance le moteur en arrière à la vitesse spécifiée, pour chercher l'origine de l'axe.

function **findHomePosition**(**speed**)

Paramètres :

speed vitesse désirée, en pas par seconde.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→get_advertisedValue() YStepperMotor
steppermotor→advertisedValue()
steppermotor.get_advertisedValue()
steppermotor.get_advertisedValue()

Retourne la valeur courante du moteur pas à pas (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante du moteur pas à pas (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

steppermotor→get_auxSignal() YStepperMotor
steppermotor→auxSignal()
steppermotor.get_auxSignal()
steppermotor.get_auxSignal()

Retourne la valeur actuelle du signal généré sur la sortie auxiliaire.

function **get_auxSignal**( )

Retourne :
un entier représentant la valeur actuelle du signal généré sur la sortie auxiliaire

En cas d'erreur, déclenche une exception ou retourne `Y_AUXSIGNAL_INVALID`.

steppermotor→get_diags() YStepperMotor
steppermotor→diags()steppermotor.get_diags()
steppermotor.get_diags()

Retourne l'état déatillé du contrôleur de moteur pas-à-pas (bitmap).

function **get_diags**( )

Retourne :
un entier représentant l'état déatillé du contrôleur de moteur pas-à-pas (bitmap)

En cas d'erreur, déclenche une exception ou retourne `Y_DIAGS_INVALID`.

steppermotor→get_errorMessage() YStepperMotor
steppermotor→errorMessage()
steppermotor.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moteur pas à pas.

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 moteur pas à pas.

steppermotor→get_errorType() YStepperMotor
steppermotor→errorType()
steppermotor.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moteur pas à pas.

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 moteur pas à pas.

steppermotor→get_friendlyName() YStepperMotor
steppermotor→friendlyName()
steppermotor.get_friendlyName()

Retourne un identifiant global du moteur pas à pas au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du moteur pas à pas si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du moteur pas à pas (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le moteur pas à pas en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

steppermotor→get_functionDescriptor() YStepperMotor
steppermotor→functionDescriptor()
steppermotor.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

steppermotor→get_functionId() YStepperMotor
steppermotor→functionId()
steppermotor.get_functionId()

Retourne l'identifiant matériel du moteur pas à pas, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le moteur pas à pas (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

steppermotor→get_hardwareId() YStepperMotor
steppermotor→hardwareId()
steppermotor.get_hardwareId()

Retourne l'identifiant matériel unique du moteur pas à pas au format SERIAL.FUNCTIONID.

function **get_hardwareId**( )

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du moteur pas à pas (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le moteur pas à pas (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

steppermotor→get_logicalName() YStepperMotor
steppermotor→logicalName()
steppermotor.get_logicalName()
steppermotor.get_logicalName()

Retourne le nom logique du moteur pas à pas.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du moteur pas à pas.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

steppermotor→get_maxAccel() YStepperMotor
steppermotor→maxAccel()
steppermotor.get_maxAccel()
steppermotor.get_maxAccel()

Retourne l'accélération maximale du moteur, mesurée en pas par seconde^2.

function **get_maxAccel**( )

Retourne :
une valeur numérique représentant l'accélération maximale du moteur, mesurée en pas par seconde^2

En cas d'erreur, déclenche une exception ou retourne `Y_MAXACCEL_INVALID`.

steppermotor→get_maxSpeed() YStepperMotor
steppermotor→maxSpeed()
steppermotor.get_maxSpeed()
steppermotor.get_maxSpeed()

Retourne la vitesse maximale du moteur, mesurée en pas par seconde.

function **get_maxSpeed**( )

Retourne :
une valeur numérique représentant la vitesse maximale du moteur, mesurée en pas par seconde

En cas d'erreur, déclenche une exception ou retourne `Y_MAXSPEED_INVALID`.

steppermotor→get_module() YStepperMotor
steppermotor→module()steppermotor.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

steppermotor→get_motorState() YStepperMotor
steppermotor→motorState()
steppermotor.get_motorState()
steppermotor.get_motorState()

Retourne l'état de fonctionnement du moteur.

function **get_motorState**( )

Retourne :

une valeur parmi `Y_MOTORSTATE_ABSENT`, `Y_MOTORSTATE_ALERT`, `Y_MOTORSTATE_HI_Z`, `Y_MOTORSTATE_STOP`, `Y_MOTORSTATE_RUN` et `Y_MOTORSTATE_BATCH` représentant l'état de fonctionnement du moteur

En cas d'erreur, déclenche une exception ou retourne `Y_MOTORSTATE_INVALID`.

steppermotor→get_overcurrent() YStepperMotor
steppermotor→overcurrent()
steppermotor.get_overcurrent()
steppermotor.get_overcurrent()

Retourne le seuil de déclenchement de la sécurité de dépassement de courant, mesuré en mA.

function **get_overcurrent**( )

Retourne :
un entier représentant le seuil de déclenchement de la sécurité de dépassement de courant, mesuré en mA

En cas d'erreur, déclenche une exception ou retourne `Y_OVERCURRENT_INVALID`.

steppermotor→get_pullinSpeed() YStepperMotor
steppermotor→pullinSpeed()
steppermotor.get_pullinSpeed()
steppermotor.get_pullinSpeed()

Retourne la vitesse du moteur atteignable directement depuis l'arrêt, mesurée en pas par seconde.

function **get_pullinSpeed**( )

Retourne :

une valeur numérique représentant la vitesse du moteur atteignable directement depuis l'arrêt, mesurée en pas par seconde

En cas d'erreur, déclenche une exception ou retourne `Y_PULLINSPEED_INVALID`.

steppermotor→get_speed() YStepperMotor
steppermotor→speed()steppermotor.get_speed()
steppermotor.get_speed()

Retourne la vitesse actuelle du moteur, mesurée en pas par seconde.

function **get_speed**( )

Pour changer cette vitesse, utilisez la méthode `changeSpeed()`.

Retourne :
une valeur numérique représentant la vitesse actuelle du moteur, mesurée en pas par seconde

En cas d'erreur, déclenche une exception ou retourne `Y_SPEED_INVALID`.

steppermotor→get_stepPos() YStepperMotor steppermotor→stepPos()steppermotor.get_stepPos() steppermotor.get_stepPos()

Retourne la position logique actuelle du moteur, mesurée en pas.

function **get_stepPos**( )

La valeur peut être fractionnaire lorsque le micro-stepping est utilisé.

Retourne :
une valeur numérique représentant la position logique actuelle du moteur, mesurée en pas

En cas d'erreur, déclenche une exception ou retourne `Y_STEPPOS_INVALID`.

steppermotor→get_stepping() YStepperMotor
steppermotor→stepping()
steppermotor.get_stepping()
steppermotor.get_stepping()

Retourne le type de stepping utilisé pour piloter le moteur.

function **get_stepping**( )

Retourne :

une valeur parmi Y_STEPPING_MICROSTEP16, Y_STEPPING_MICROSTEP8, Y_STEPPING_MICROSTEP4, Y_STEPPING_HALFSTEP et Y_STEPPING_FULLSTEP représentant le type de stepping utilisé pour piloter le moteur

En cas d'erreur, déclenche une exception ou retourne Y_STEPPING_INVALID.

steppermotor→get_tCurrRun() YStepperMotor
steppermotor→tCurrRun()
steppermotor.get_tCurrRun()
steppermotor.get_tCurrRun()

Retourne la limite de courant pour la régulation de torque en mouvement, mesurée en mA.

function **get_tCurrRun**( )

Retourne :
un entier représentant la limite de courant pour la régulation de torque en mouvement, mesurée en mA

En cas d'erreur, déclenche une exception ou retourne `Y_TCURRRUN_INVALID`.

steppermotor→get_tCurrStop() YStepperMotor
steppermotor→tCurrStop()
steppermotor.get_tCurrStop()
steppermotor.get_tCurrStop()

Retourne la limite de courant pour la régulation de torque à l'arrêt, mesurée en mA.

function **get_tCurrStop**( )

Retourne :
un entier représentant la limite de courant pour la régulation de torque à l'arrêt, mesurée en mA

En cas d'erreur, déclenche une exception ou retourne `Y_TCURRSTOP_INVALID`.

steppermotor→get_userData() YStepperMotor
steppermotor→userData()
steppermotor.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.

steppermotor→isOnline()steppermotor.isOnline() YStepperMotor

Vérifie si le module hébergeant le moteur pas à pas est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du moteur pas à pas sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si le moteur pas à pas est joignable, `false` sinon

steppermotor→load()steppermotor.load() YStepperMotor

Met en cache les valeurs courantes du moteur pas à pas, 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 occasionellement 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.

steppermotor→loadAttribute() YStepperMotor
steppermotor.loadAttribute()
steppermotor.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

steppermotor→moveRel()steppermotor.moveRel() YStepperMotor steppermotor.moveRel()

Contrôle le moteur pour atteindre une position relative donnée.

function **moveRel**(**relPos**)

Le temps nécessaire pour atteindre la position dépend des paramètres d'accélération et de vitesse maximale configurés pour le moteur.

Paramètres :

relPos position relative désirée, en pas depuis la position actuelle.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→moveTo()steppermotor.moveTo() YStepperMotor steppermotor.moveTo()

Contrôle le moteur pour atteindre une position absolue donnée.

function **moveTo**(**absPos**)

Le temps nécessaire pour atteindre la position dépend des paramètres d'accélération et de vitesse maximale configurés pour le moteur.

Paramètres :

absPos position absolue désirée, en pas depuis l'origine.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→muteValueCallbacks() YStepperMotor
steppermotor.muteValueCallbacks()
steppermotor.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→nextStepperMotor() YStepperMotor
steppermotor.nextStepperMotor()
steppermotor.nextStepperMotor()

Continue l'énumération des moteur pas à pas commencée à l'aide de `yFirstStepperMotor()`.

function **nextStepperMotor**( )

Retourne :

un pointeur sur un objet `YStepperMotor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

steppermotor→pause()steppermotor.pause() YStepperMotor steppermotor.pause()

Garde le moteur dans le même état pour la durée spécifiée, avant d'exécuter la commande suivante.

function **pause**(**waitMs**)

Paramètres :

waitMs temps d'attente, en 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.

steppermotor→registerValueCallback() YStepperMotor
steppermotor.registerValueCallback()
steppermotor.registerValueCallback()

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.

steppermotor→reset()steppermotor.reset() YStepperMotor steppermotor.reset()

Réinitialise le controlleur et quittance toutes les alertes.

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.

steppermotor→set_auxSignal() YStepperMotor
steppermotor→setAuxSignal()
steppermotor.set_auxSignal()
steppermotor.set_auxSignal()

Modifie la valeur du signal généré sur la sortie auxiliaire.

function **set_auxSignal**(**newval**)

Les valeurs valides dépendent de la configuration du type de signal utilisé sur la sortie auxiliaire.

Paramètres :

newval un entier représentant la valeur du signal généré sur la sortie auxiliaire

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→set_logicalName() YStepperMotor
steppermotor→setLogicalName()
steppermotor.set_logicalName()
steppermotor.set_logicalName()

Modifie le nom logique du moteur pas à pas.

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 moteur pas à pas.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→set_maxAccel() YStepperMotor
steppermotor→setMaxAccel()
steppermotor.set_maxAccel()
steppermotor.set_maxAccel()

Modifie l'accélération maximale du moteur, mesurée en pas par seconde^2.

function **set_maxAccel**(**newval**)

Paramètres :

newval une valeur numérique représentant l'accélération maximale du moteur, mesurée en pas par seconde^2

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→set_maxSpeed() YStepperMotor
steppermotor→setMaxSpeed()
steppermotor.set_maxSpeed()
steppermotor.set_maxSpeed()

Modifie la vitesse maximale du moteur, mesurée en pas par seconde.

function **set_maxSpeed**(**newval**)

Paramètres :

newval une valeur numérique représentant la vitesse maximale du moteur, mesurée en pas par seconde

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→set_overcurrent() YStepperMotor
steppermotor→setOvercurrent()
steppermotor.set_overcurrent()
steppermotor.set_overcurrent()

Modifie le seuil de déclenchement de la sécurité de dépassement de courant, mesuré en mA.

function **set_overcurrent**(**newval**)

Paramètres :

newval un entier représentant le seuil de déclenchement de la sécurité de dépassement de courant, mesuré en mA

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→set_pullinSpeed() YStepperMotor
steppermotor→setPullinSpeed()
steppermotor.set_pullinSpeed()
steppermotor.set_pullinSpeed()

Modifie la vitesse du moteur atteignable directement depuis l'arrêt, mesurée en pas par seconde.

function **set_pullinSpeed**(**newval**)

Paramètres :

newval une valeur numérique représentant la vitesse du moteur atteignable directement depuis l'arrêt, mesurée en pas par seconde

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→set_stepPos() YStepperMotor
steppermotor→setStepPos()
steppermotor.set_stepPos()
steppermotor.set_stepPos()

Modifie la position logique actuelle du moteur, mesurée en pas.

function **set_stepPos**(**newval**)

Cette commande ne déclanche pas de mouvement du moteur, elle sert uniquement à configurer l'origine du compteur de position. La partie fractionnaire, dépendant de la position physique du rotor, n'est pas modifiée. Pour déclancher un mouvement, utilisez la méthode `moveTo()` ou la méthode `moveRel()`.

Paramètres :

newval une valeur numérique représentant la position logique actuelle du moteur, mesurée en pas

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→set_stepping() YStepperMotor
steppermotor→setStepping()
steppermotor.set_stepping()
steppermotor.set_stepping()

Modifie le type de stepping utilisé pour piloter le moteur.

function **set_stepping**(**newval**)

Paramètres :

newval une valeur parmi `Y_STEPPING_MICROSTEP16`, `Y_STEPPING_MICROSTEP8`, `Y_STEPPING_MICROSTEP4`, `Y_STEPPING_HALFSTEP` et `Y_STEPPING_FULLSTEP` représentant le type de stepping utilisé pour piloter le moteur

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→set_tCurrRun() YStepperMotor
steppermotor→setTCurrRun()
steppermotor.set_tCurrRun()
steppermotor.set_tCurrRun()

Modifie la limite de courant pour la régulation de torque en mouvement, mesurée en mA.

function **set_tCurrRun**(**newval**)

Paramètres :

newval un entier représentant la limite de courant pour la régulation de torque en mouvement, mesurée en mA

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→set_tCurrStop() YStepperMotor
steppermotor→setTCurrStop()
steppermotor.set_tCurrStop()
steppermotor.set_tCurrStop()

Modifie la limite de courant pour la régulation de torque à l'arrêt, mesurée en mA.

function **set_tCurrStop**(**newval**)

Paramètres :

newval un entier représentant la limite de courant pour la régulation de torque à l'arrêt, mesurée en mA

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→set_userData() YStepperMotor
steppermotor→setUserData()
steppermotor.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

steppermotor→unmuteValueCallbacks() YStepperMotor
steppermotor.unmuteValueCallbacks()
steppermotor.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

steppermotor→wait_async() YStepperMotor
steppermotor.wait_async()

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.

3.64. Interface de la fonction Temperature

La classe YTemperature permet de lire et de configurer les capteurs de température Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, elle permet de configurer les paramètres spécifiques de certains types de capteur (type de connection, table d'étalonnage).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_temperature.js'></script>
cpp	#include "yocto_temperature.h"
m	#import "yocto_temperature.h"
pas	uses yocto_temperature;
vb	yocto_temperature.vb
cs	yocto_temperature.cs
java	import com.yoctopuce.YoctoAPI.YTemperature;
uwp	import com.yoctopuce.YoctoAPI.YTemperature;
py	from yocto_temperature import *
php	require_once('yocto_temperature.php');
es	in HTML: <script src="../../lib/yocto_temperature.js"></script> in node.js: require('yoctolib-es2017/yocto_temperature.js');

Fonction globales
yFindTemperature(**func**) Permet de retrouver un capteur de température d'après un identifiant donné.
yFindTemperatureInContext(**yctx**, **func**) Permet de retrouver un capteur de température d'après un identifiant donné dans un Context YAPI.
yFirstTemperature() Commence l'énumération des capteurs de température accessibles par la librairie.
yFirstTemperatureInContext(**yctx**) 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→clearCache() Invalide le cache.
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), en degrés Celsius, sous forme de nombre à virgule.
temperature→get_currentValue() Retourne la valeur actuelle de la température, en degrés Celsius, sous forme de nombre à virgule.
temperature→get_dataLogger() Retourne l'objet YDataLogger du module qui héberge le senseur.
temperature→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

temperature→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

temperature→get_friendlyName()

Retourne un identifiant global du capteur de température au format `NOM_MODULE.NOM_FONCTION`.

temperature→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

temperature→get_functionId()

Retourne l'identifiant matériel du capteur de température, sans référence au module.

temperature→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de température au format `SERIAL.FUNCTIONID`.

temperature→get_highestValue()

Retourne la valeur maximale observée pour la température depuis le démarrage du module.

temperature→get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

temperature→get_logicalName()

Retourne le nom logique du capteur de température.

temperature→get_lowestValue()

Retourne la valeur minimale observée pour la température depuis le démarrage du module.

temperature→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

temperature→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

temperature→get_recordedData(**startTime**, **endTime**)

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

temperature→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

temperature→get_resolution()

Retourne la résolution des valeurs mesurées.

temperature→get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

temperature→get_sensorType()

Retourne le type de capteur de température utilisé par le module

temperature→get_signalUnit()

Retourne l'unité du signal électrique utilisé par le capteur.

temperature→get_signalValue()

Retourne la valeur actuelle du signal électrique mesuré par le capteur.

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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

temperature→load(**msValidity**)

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

temperature→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

temperature→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

temperature→loadThermistorResponseTable(**tempValues**, **resValues**)

Récupère la table de réponse d'un thermistor précédemment enregistrée à l'aide de la fonction `set_thermistorResponseTable`.

temperature→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

temperature→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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_ntcParameters(**res25**, **beta**)

Configure les paramètres d'un thermistor NTC pour calculer correctement la température sur la base de la résistance mesurée.

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_thermistorResponseTable(**tempValues**, **resValues**)

Enregistre la table de réponse d'un thermistor, afin de pouvoir interpoler la température sur la base de la résistance mesurée.

temperature→set_unit(newval)

Change l'unité dans laquelle la température mesurée est exprimée.

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→startDataLogger()

Démarre l'enregistreur de données du module.

temperature→stopDataLogger()

Arrête l'enregistreur de données du module.

temperature→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

temperature→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YTemperature.FindTemperature() YTemperature yFindTemperature()YTemperature.FindTemperature() YTemperature.FindTemperature()

Permet de retrouver un capteur de température d'après un identifiant donné.

```
function FindTemperature( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindTemperatureInContext() YTemperature
yFindTemperatureInContext()
YTemperature.FindTemperatureInContext()
YTemperature.FindTemperatureInContext()

Permet de retrouver un capteur de température d'après un identifiant donné dans un Context YAPI.

function **FindTemperatureInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de température sans ambiguïté

Retourne :

un objet de classe `YTemperature` qui permet ensuite de contrôler le capteur de température.

YTemperature.FirstTemperature() YTemperature yFirstTemperature()YTemperature.FirstTemperature() YTemperature.FirstTemperature()

Commence l'énumération des capteurs de température accessibles par la librairie.

function **FirstTemperature**( )

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.

YTemperature.FirstTemperatureInContext() YTemperature yFirstTemperatureInContext() YTemperature.FirstTemperatureInContext() YTemperature.FirstTemperatureInContext()

Commence l'énumération des capteurs de température accessibles par la librairie.

function **FirstTemperatureInContext**(**yctx**)

Utiliser la fonction `YTemperature.nextTemperature()` pour itérer sur les autres capteurs de température.

Paramètres :

yctx un contexte YAPI.

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() YTemperature
temperature.calibrateFromPoints()
temperature.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

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→clearCache()temperature.clearCache() YTemperature

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur de température. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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()
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()
temperature.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés Celsius, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés Celsius, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

temperature→get_currentValue() YTemperature
temperature→currentValue()
temperature.get_currentValue()
temperature.get_currentValue()

Retourne la valeur actuelle de la température, en degrés Celsius, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle de la température, en degrés Celsius, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

temperature→get_dataLogger() YTemperature
temperature→dataLogger()
temperature.get_dataLogger()
temperature.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

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() YTemperature
temperature→errorType()
temperature.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

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'identifant 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 example `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de température (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

temperature→get_hardwareId() YTemperature
temperature→hardwareId()
temperature.get_hardwareId()

Retourne l'identifiant matériel unique du capteur de température au format SERIAL.FUNCTIONID.

function **get_hardwareId**( )

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de température (par example 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()
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()
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()
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() YTemperature
temperature→lowestValue()
temperature.get_lowestValue()
temperature.get_lowestValue()

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_recordedData() YTemperature
temperature→recordedData()
temperature.get_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 find de l'intercalle 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()
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()
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_sensorState() YTemperature
temperature→sensorState()
temperature.get_sensorState()
temperature.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :
un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

temperature→get_sensorType() YTemperature
temperature→sensorType()
temperature.get_sensorType()
temperature.get_sensorType()

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, Y_SENSORTYPE_PT100_2WIRES, Y_SENSORTYPE_RES_OHM, Y_SENSORTYPE_RES_NTC, Y_SENSORTYPE_RES_LINEAR et Y_SENSORTYPE_RES_INTERNAL 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_signalUnit() YTemperature
temperature→signalUnit()
temperature.get_signalUnit()
temperature.get_signalUnit()

Retourne l'unité du signal électrique utilisé par le capteur.

function **get_signalUnit**( )

Retourne :
une chaîne de caractères représentant l'unité du signal électrique utilisé par le capteur

En cas d'erreur, déclenche une exception ou retourne `Y_SIGNALUNIT_INVALID`.

temperature→get_signalValue() YTemperature
temperature→signalValue()
temperature.get_signalValue()
temperature.get_signalValue()

Retourne la valeur actuelle du signal électrique mesuré par le capteur.

function **get_signalValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle du signal électrique mesuré par le capteur

En cas d'erreur, déclenche une exception ou retourne `Y_SIGNALVALUE_INVALID`.

temperature→get_unit() YTemperature
temperature→unit()temperature.get_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→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 occasionellement 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→loadAttribute() YTemperature
temperature.loadAttribute()
temperature.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

temperature→loadCalibrationPoints() YTemperature
temperature.loadCalibrationPoints()
temperature.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→loadThermistorResponseTable() YTemperature
temperature.loadThermistorResponseTable()
temperature.loadThermistorResponseTable()

Récupère la table de réponse d'un thermistor précédemment enregistrée à l'aide de la fonction `set_thermistorResponseTable`.

function **loadThermistorResponseTable**(**tempValues**, **resValues**)

Cette fonction ne peut être utilisée qu'avec les capteurs de température basés sur un thermistor.

Paramètres :

tempValues tableau de nombres flottants, qui sera rempli par la fonction avec les différentes températures (en degrés Celcius) pour lesquelles la résistance du thermistor est spécifiée.

resValues tableau de nombres flottants, qui sera rempli par la fonction avec les résistances (en Ohms) pour chacun des points de température, index par index.

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→muteValueCallbacks() YTemperature
temperature.muteValueCallbacks()
temperature.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→nextTemperature() YTemperature
temperature.nextTemperature()
temperature.nextTemperature()

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() YTemperature
temperature.registerTimedReportCallback()
temperature.registerTimedReportCallback()

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

function **registerTimedReportCallback**(**callback**)

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

temperature→registerValueCallback() YTemperature
temperature.registerValueCallback()
temperature.registerValueCallback()

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()
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() YTemperature
temperature→setLogFrequency()
temperature.set_logFrequency()
temperature.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.

temperature→set_logicalName() YTemperature
temperature→setLogicalName()
temperature.set_logicalName()
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()
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_ntcParameters() YTemperature
temperature→setNtcParameters()
temperature.set_ntcParameters()
temperature.set_ntcParameters()

Configure les paramètres d'un thermistor NTC pour calculer correctement la température sur la base de la résistance mesurée.

function **set_ntcParameters**(**res25**, **beta**)

Pour plus de précision, vous pouvez saisir une table complète à l'aide de la fonction set_thermistorResponseTable. Cette fonction ne peut être utilisée qu'avec les capteurs de température basés sur un thermistor.

Paramètres :

res25 résistance à 25 degrés Celsius

beta coefficient Beta

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()
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()
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()
temperature.set_sensorType()

Change le type de senseur utilisé par le module.

function **set_sensorType**(**newval**)

Cette function 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 ou un thermistor. 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`, `Y_SENSORTYPE_PT100_2WIRES`, `Y_SENSORTYPE_RES_OHM`, `Y_SENSORTYPE_RES_NTC`, `Y_SENSORTYPE_RES_LINEAR` et `Y_SENSORTYPE_RES_INTERNAL`

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_thermistorResponseTable() YTemperature
temperature→setThermistorResponseTable()
temperature.set_thermistorResponseTable()
temperature.set_thermistorResponseTable()

Enregistre la table de réponse d'un thermistor, afin de pouvoir interpoler la température sur la base de la résistance mesurée.

function **set_thermistorResponseTable**(**tempValues**, **resValues**)

Cette fonction ne peut être utilisée qu'avec les capteurs de température basés sur un thermistor.

Paramètres :

tempValues tableau de nombres flottants, correspondant aux différentes températures (en degrés Celcius) pour lesquelles la résistance du thermistor est spécifiée.

resValues tableau de nombres flottants, correspondant aux résistances (en Ohms) pour chacun des points de température, index par index.

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_unit() YTemperature
temperature→setUnit()temperature.set_unit()
temperature.set_unit()

Change l'unité dans laquelle la température mesurée est exprimée.

function **set_unit**(**newval**)

Cette unité est une chaîne de charactère. Si cette chaîne de caractère se termine par un F les valeurs mesurée seront rendues en degrés Fahrenheit, si elle se termine par un K, les valeurs de température seront rendues en degrés Kelvin. Si elle se termine par un C, les valeurs de température seront rendues en degrés Celsius. Si elle ne setermine n'importe quel autre caractère, le changement est ignoré. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé. Attention: si une calibration spécifique est définie pour la fonction temperature, un changement d'unité a toutes les chances de la fausser.

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.

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→startDataLogger() YTemperature
temperature.startDataLogger()
temperature.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

temperature→stopDataLogger() YTemperature
temperature.stopDataLogger()
temperature.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

temperature→unmuteValueCallbacks() YTemperature
temperature.unmuteValueCallbacks()
temperature.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.65. Interface de la fonction Tilt

La classe YSensor est la classe parente de tous les senseurs Yoctopuce. Elle permet de lire la valeur courante et l'unité de n'importe quel capteur, de lire les valeurs min/max, de configurer la fréquence d'enregistrement autonome des données et de récupérer les mesures enregistrées. Elle permet aussi d'enregistrer un callback appelé lorsque la valeur mesurée change ou à intervalle prédéfini. L'utilisation de cette classe plutôt qu'une de ces sous-classes permet de créer des application génériques, compatibles même avec les capteurs Yoctopuce futurs. Note: la classe YAnButton est le seul type d'entrée analogique qui n'hérite pas de YSensor.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_tilt.js'></script>
cpp	#include "yocto_tilt.h"
m	#import "yocto_tilt.h"
pas	uses yocto_tilt;
vb	yocto_tilt.vb
cs	yocto_tilt.cs
java	import com.yoctopuce.YoctoAPI.YTilt;
uwp	import com.yoctopuce.YoctoAPI.YTilt;
py	from yocto_tilt import *
php	require_once('yocto_tilt.php');
es	in HTML: <script src="../../lib/yocto_tilt.js"></script> in node.js: require('yoctolib-es2017/yocto_tilt.js');

Fonction globales

yFindTilt(**func**)

Permet de retrouver un inclinomètre d'après un identifiant donné.

yFindTiltInContext(**yctx**, **func**)

Permet de retrouver un inclinomètre d'après un identifiant donné dans un Context YAPI.

yFirstTilt()

Commence l'énumération des inclinomètres accessibles par la librairie.

yFirstTiltInContext(**yctx**)

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→clearCache()

Invalide le cache.

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_bandwidth()

Retourne la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

tilt→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés, sous forme de nombre à virgule.

tilt→get_currentValue()

Retourne la valeur actuelle de l'inclinaison, en degrés, sous forme de nombre à virgule.

tilt→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

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_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

tilt→load(**msValidity**)

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

tilt→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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_bandwidth(**newval**)

Modifie la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

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.

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→startDataLogger()

Démarre l'enregistreur de données du module.

tilt→stopDataLogger()

Arrête l'enregistreur de données du module.

tilt→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

tilt→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YTilt.FindTilt() YTilt yFindTilt()YTilt.FindTilt()YTilt.FindTilt()

Permet de retrouver un inclinomètre d'après un identifiant donné.

```
function FindTilt( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindTiltInContext() YTilt yFindTiltInContext()YTilt.FindTiltInContext() YTilt.FindTiltInContext()

Permet de retrouver un inclinomètre d'après un identifiant donné dans un Context YAPI.

function **FindTiltInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'inclinomètre sans ambiguïté

Retourne :

un objet de classe `YTilt` qui permet ensuite de contrôler l'inclinomètre.

YTilt.FirstTilt() YTilt yFirstTilt()YTilt.FirstTilt()YTilt.FirstTilt()

Commence l'énumération des inclinomètres accessibles par la librairie.

function **FirstTilt**( )

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.

YTilt.FirstTiltInContext() YTilt yFirstTiltInContext()YTilt.FirstTiltInContext() YTilt.FirstTiltInContext()

Commence l'énumération des inclinomètres accessibles par la librairie.

function **FirstTiltInContext**(**yctx**)

Utiliser la fonction `YTilt.nextTilt()` pour itérer sur les autres inclinomètres.

Paramètres :

yctx un contexte YAPI.

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 tilt.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→clearCache()tilt.clearCache() YTilt

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'inclinomètre. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant l'inclinomètre (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

tilt→get_advertisedValue() YTilt
tilt→advertisedValue()tilt.get_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_bandwidth() YTilt
tilt→bandwidth()tilt.get_bandwidth()
tilt.get_bandwidth()

Retourne la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

function **get_bandwidth**( )

Retourne :
un entier représentant la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement)

En cas d'erreur, déclenche une exception ou retourne `Y_BANDWIDTH_INVALID`.

tilt→get_currentRawValue() YTilt
tilt→currentRawValue()tilt.get_currentRawValue()
tilt.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

tilt→get_currentValue() YTilt
tilt→currentValue()tilt.get_currentValue()
tilt.get_currentValue()

Retourne la valeur actuelle de l'inclinaison, en degrés, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle de l'inclinaison, en degrés, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

tilt→get_dataLogger() YTilt
tilt→dataLogger()tilt.get_dataLogger()
tilt.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

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() YTilt
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'identifant 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 example `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 example 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()
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()
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()
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()
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_recordedData() YTilt
tilt→recordedData()tilt.get_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**)

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 find de l'intercalle 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()
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()
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_sensorState() YTilt
tilt→sensorState()tilt.get_sensorState()
tilt.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :
un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

tilt→get_unit() YTilt
tilt→unit()tilt.get_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→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 occasionellement 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→loadAttribute()tilt.loadAttribute() YTilt tilt.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

tilt→loadCalibrationPoints() YTilt
tilt.loadCalibrationPoints()tilt.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→muteValueCallbacks()tilt.muteValueCallbacks() YTilt tilt.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→nextTilt()tilt.nextTilt()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() YTilt
tilt.registerTimedReportCallback()
tilt.registerTimedReportCallback()

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

function **registerTimedReportCallback**(**callback**)

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

tilt→registerValueCallback() YTilt
tilt.registerValueCallback()tilt.registerValueCallback()

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_bandwidth() YTilt
tilt→setBandwidth()tilt.set_bandwidth()
tilt.set_bandwidth()

Modifie la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement).

function **set_bandwidth**(**newval**)

Lorsque la fréquence est plus basse, un moyennage est effectué.

Paramètres :

newval un entier représentant la fréquence de rafraîchissement de la mesure, en Hz (Yocto-3D-V2 seulement)

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_highestValue() YTilt
tilt→setHighestValue()tilt.set_highestValue()
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()
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()
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()
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()
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()
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'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

tilt→startDataLogger()tilt.startDataLogger() YTilt tilt.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

tilt→stopDataLogger()tilt.stopDataLogger() YTilt
tilt.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

tilt→unmuteValueCallbacks() YTilt
tilt.unmuteValueCallbacks()
tilt.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.66. Interface de la fonction Voc

La classe YVoc permet de lire et de configurer les capteurs de composants organiques volatiles Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_voc.js'></script>
cpp	#include "yocto_voc.h"
m	#import "yocto_voc.h"
pas	uses yocto_voc;
vb	yocto_voc.vb
cs	yocto_voc.cs
java	import com.yoctopuce.YoctoAPI.YVoc;
uwp	import com.yoctopuce.YoctoAPI.YVoc;
py	from yocto_voc import *
php	require_once('yocto_voc.php');
es	in HTML: <script src="../../lib/yocto_voc.js"></script> in node.js: require('yoctolib-es2017/yocto_voc.js');

Fonction globales

yFindVoc(**func**)

Permet de retrouver un capteur de Composés Organiques Volatils d'après un identifiant donné.

yFindVocInContext(**yctx**, **func**)

Permet de retrouver un capteur de Composés Organiques Volatils d'après un identifiant donné dans un Context YAPI.

yFirstVoc()

Commence l'énumération des capteurs de Composés Organiques Volatils accessibles par la librairie.

yFirstVocInContext(**yctx**)

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→clearCache()

Invalide le cache.

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), en ppm (vol), sous forme de nombre à virgule.

voc→get_currentValue()

Retourne la valeur actuelle du taux de VOC estimé, en ppm (vol), sous forme de nombre à virgule.

voc→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

voc→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

voc→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

voc→get_friendlyName()

Retourne un identifiant global du capteur de Composés Organiques Volatils au format `NOM_MODULE.NOM_FONCTION`.

voc→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

voc→get_functionId()

Retourne l'identifiant matériel du capteur de Composés Organiques Volatils, sans référence au module.

voc→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de Composés Organiques Volatils au format `SERIAL.FUNCTIONID`.

voc→get_highestValue()

Retourne la valeur maximale observée pour le taux de VOC estimé depuis le démarrage du module.

voc→get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

voc→get_logicalName()

Retourne le nom logique du capteur de Composés Organiques Volatils.

voc→get_lowestValue()

Retourne la valeur minimale observée pour le taux de VOC estimé depuis le démarrage du module.

voc→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

voc→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

voc→get_recordedData(**startTime**, **endTime**)

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

voc→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

voc→get_resolution()

Retourne la résolution des valeurs mesurées.

voc→get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

voc→load(**msValidity**)

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

voc→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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.

voc→set_logFrequency(**newval**)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

voc→set_logicalName(**newval**)

Modifie le nom logique du capteur de Composés Organiques Volatils.

voc→set_lowestValue(**newval**)

Modifie la mémoire de valeur minimale observée.

voc→set_reportFrequency(**newval**)

Modifie la fréquence de notification périodique des valeurs mesurées.

voc→set_resolution(**newval**)

Modifie la résolution des valeurs physique mesurées.

voc→set_userData(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

voc→startDataLogger()

Démarre l'enregistreur de données du module.

voc→stopDataLogger()

Arrête l'enregistreur de données du module.

voc→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

voc→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YVoc.FindVoc() YVoc yFindVoc()YVoc.FindVoc()YVoc.FindVoc()

Permet de retrouver un capteur de Composés Organiques Volatils d'après un identifiant donné.

```
function FindVoc( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindVocInContext() YVoc yFindVocInContext()YVoc.FindVocInContext() YVoc.FindVocInContext()

Permet de retrouver un capteur de Composés Organiques Volatils d'après un identifiant donné dans un Context YAPI.

function **FindVocInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de Composés Organiques Volatils sans ambiguïté

Retourne :

un objet de classe YVoc qui permet ensuite de contrôler le capteur de Composés Organiques Volatils.

YVoc.FirstVoc() YVoc yFirstVoc()YVoc.FirstVoc()YVoc.FirstVoc()

Commence l'énumération des capteurs de Composés Organiques Volatils accessibles par la librairie.

```
function FirstVoc( )
```

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.

YVoc.FirstVocInContext() YVoc yFirstVocInContext()YVoc.FirstVocInContext() YVoc.FirstVocInContext()

Commence l'énumération des capteurs de Composés Organiques Volatils accessibles par la librairie.

function **FirstVocInContext**(**yctx**)

Utiliser la fonction YVoc.nextVoc() pour itérer sur les autres capteurs de Composés Organiques Volatils.

Paramètres :

yctx un contexte YAPI.

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 voc.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→clearCache()voc.clearCache() YVoc

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur de Composés Organiques Volatils. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

Retourne :

une chaîne de caractères décrivant le capteur de Composés Organiques Volatils (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

voc→get_advertisedValue() YVoc
voc→advertisedValue()voc.get_advertisedValue()
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()
voc.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en ppm (vol), sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en ppm (vol), sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

voc→get_currentValue() YVoc
voc→currentValue()voc.get_currentValue()
voc.get_currentValue()

Retourne la valeur actuelle du taux de VOC estimé, en ppm (vol), sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle du taux de VOC estimé, en ppm (vol), sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

voc→get_dataLogger() YVoc
voc→dataLogger()voc.get_dataLogger()
voc.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

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'identifant 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 example `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de Composés Organiques Volatils (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

voc→get_hardwareId() 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 example 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()
voc.get_highestValue()

Retourne la valeur maximale observée pour le taux de VOC estimé depuis le démarrage du module.

function **get_highestValue**( )

Retourne :
une valeur numérique représentant la valeur maximale observée pour le taux de VOC estimé depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

voc→get_logFrequency() YVoc
voc→logFrequency()voc.get_logFrequency()
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()
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()
voc.get_lowestValue()

Retourne la valeur minimale observée pour le taux de VOC estimé depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :

une valeur numérique représentant la valeur minimale observée pour le taux de VOC estimé depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

voc→get_module() YVoc
voc→module()voc.get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

function **get_module**( )

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :
une instance de YModule

voc→get_recordedData() YVoc
voc→recordedData()voc.get_recordedData()
voc.get_recordedData()

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

function **get_recordedData**(**startTime**, **endTime**)

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la find de l'intercalle 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()
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()
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_sensorState() YVoc
voc→sensorState()voc.get_sensorState()
voc.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :
un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

voc→get_unit() YVoc
voc→unit()voc.get_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→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 occasionellement 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→loadAttribute()voc.loadAttribute() YVoc voc.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

voc→loadCalibrationPoints() YVoc
voc.loadCalibrationPoints()
voc.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→muteValueCallbacks() YVoc voc.muteValueCallbacks()voc.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→nextVoc()voc.nextVoc()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() YVoc
voc.registerTimedReportCallback()
voc.registerTimedReportCallback()

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() YVoc
voc.registerValueCallback()
voc.registerValueCallback()

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()
voc.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.

voc→set_logFrequency() YVoc
voc→setLogFrequency()voc.set_logFrequency()
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()
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()
voc.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.

voc→set_reportFrequency() YVoc
voc→setReportFrequency()
voc.set_reportFrequency()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()
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→startDataLogger()voc.startDataLogger() YVoc voc.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

voc→stopDataLogger()voc.stopDataLogger() YVoc voc.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

voc→unmuteValueCallbacks() YVoc
voc.unmuteValueCallbacks()
voc.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.67. Interface de la fonction Voltage

La classe YVoltage permet de lire et de configurer les capteurs de tension Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js    <script type='text/javascript' src='yocto_voltage.js'></script>
cpp   #include "yocto_voltage.h"
m     #import "yocto_voltage.h"
pas   uses yocto_voltage;
vb    yocto_voltage.vb
cs    yocto_voltage.cs
java  import com.yoctopuce.YoctoAPI.YVoltage;
uwp   import com.yoctopuce.YoctoAPI.YVoltage;
py    from yocto_voltage import *
php   require_once('yocto_voltage.php');
es    in HTML: <script src="../../lib/yocto_voltage.js"></script>
      in node.js: require('yoctolib-es2017/yocto_voltage.js');
```

Fonction globales

yFindVoltage(**func**)

Permet de retrouver un capteur de tension d'après un identifiant donné.

yFindVoltageInContext(**yctx**, **func**)

Permet de retrouver un capteur de tension d'après un identifiant donné dans un Context YAPI.

yFirstVoltage()

Commence l'énumération des capteurs de tension accessibles par la librairie.

yFirstVoltageInContext(**yctx**)

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→clearCache()

Invalide le cache.

voltage→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

voltage→get_advertisedValue()

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

voltage→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule.

voltage→get_currentValue()

Retourne la valeur actuelle de la tension, en Volt, sous forme de nombre à virgule.

voltage→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

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 depuis le démarrage du module.

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 depuis le démarrage du module.

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_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

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→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

voltage→load(**msValidity**)

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

voltage→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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.

voltage→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

voltage→set_logicalName(newval)

Modifie le nom logique du capteur de tension.

voltage→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

voltage→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

voltage→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

voltage→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

voltage→startDataLogger()

Démarre l'enregistreur de données du module.

voltage→stopDataLogger()

Arrête l'enregistreur de données du module.

voltage→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

voltage→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YVoltage.FindVoltage() YVoltage yFindVoltage()YVoltage.FindVoltage() YVoltage.FindVoltage()

Permet de retrouver un capteur de tension d'après un identifiant donné.

```
function FindVoltage( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindVoltageInContext() YVoltage
yFindVoltageInContext()
YVoltage.FindVoltageInContext()
YVoltage.FindVoltageInContext()

Permet de retrouver un capteur de tension d'après un identifiant donné dans un Context YAPI.

function **FindVoltageInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de tension sans ambiguïté

Retourne :

un objet de classe `YVoltage` qui permet ensuite de contrôler le capteur de tension.

YVoltage.FirstVoltage() YVoltage yFirstVoltage()YVoltage.FirstVoltage() YVoltage.FirstVoltage()

Commence l'énumération des capteurs de tension accessibles par la librairie.

function **FirstVoltage**( )

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.

YVoltage.FirstVoltageInContext() YVoltage
yFirstVoltageInContext()
YVoltage.FirstVoltageInContext()
YVoltage.FirstVoltageInContext()

Commence l'énumération des capteurs de tension accessibles par la librairie.

function **FirstVoltageInContext**(**yctx**)

Utiliser la fonction YVoltage.nextVoltage() pour itérer sur les autres capteurs de tension.

Paramètres :

yctx un contexte YAPI.

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() YVoltage
voltage.calibrateFromPoints()
voltage.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→clearCache()voltage.clearCache() YVoltage

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur de tension. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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()
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()
voltage.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

voltage→get_currentValue() YVoltage
voltage→currentValue()voltage.get_currentValue()
voltage.get_currentValue()

Retourne la valeur actuelle de la tension, en Volt, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :
une valeur numérique représentant la valeur actuelle de la tension, en Volt, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

voltage→get_dataLogger() YVoltage
voltage→dataLogger()voltage.get_dataLogger()
voltage.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

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'identifant 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 example `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 example 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()
voltage.get_highestValue()

Retourne la valeur maximale observée pour la tension depuis le démarrage du module.

function **get_highestValue**( )

Retourne :
une valeur numérique représentant la valeur maximale observée pour la tension depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

voltage→get_logFrequency() YVoltage
voltage→logFrequency()voltage.get_logFrequency()
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()
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()
voltage.get_lowestValue()

Retourne la valeur minimale observée pour la tension depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :
une valeur numérique représentant la valeur minimale observée pour la tension depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

voltage→get_module() YVoltage
voltage→module()voltage.get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

function **get_module**( )

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_recordedData() YVoltage
voltage→recordedData()voltage.get_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**)

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 find de l'intercalle 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()
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()
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_sensorState() YVoltage
voltage→sensorState()voltage.get_sensorState()
voltage.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :
un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

voltage→get_unit() YVoltage
voltage→unit()voltage.get_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'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.

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→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 occasionellement 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→loadAttribute()voltage.loadAttribute() YVoltage voltage.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

voltage→loadCalibrationPoints() YVoltage
voltage.loadCalibrationPoints()
voltage.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→muteValueCallbacks() YVoltage
voltage.muteValueCallbacks()
voltage.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→nextVoltage()voltage.nextVoltage() 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() YVoltage
voltage.registerTimedReportCallback()
voltage.registerTimedReportCallback()

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

function **registerTimedReportCallback**(**callback**)

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

voltage→registerValueCallback() YVoltage
voltage.registerValueCallback()
voltage.registerValueCallback()

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()voltage.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.

voltage→set_logFrequency() YVoltage
voltage→setLogFrequency()
voltage.set_logFrequency()
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()
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()
voltage.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.

voltage→set_reportFrequency() YVoltage
voltage→setReportFrequency()
voltage.set_reportFrequency()
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()
voltage.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.

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→startDataLogger()voltage.startDataLogger() YVoltage voltage.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

voltage→stopDataLogger()voltage.stopDataLogger() YVoltage voltage.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

voltage→unmuteValueCallbacks() YVoltage
voltage.unmuteValueCallbacks()
voltage.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.68. Interface de la fonction VoltageOutput

La librairie de programmation Yoctopuce permet de changer la valeur de la sortie de tension.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_voltageoutput.js'></script>
cpp	#include "yocto_voltageoutput.h"
m	#import "yocto_voltageoutput.h"
pas	uses yocto_voltageoutput;
vb	yocto_voltageoutput.vb
cs	yocto_voltageoutput.cs
java	import com.yoctopuce.YoctoAPI.YVoltageOutput;
uwp	import com.yoctopuce.YoctoAPI.YVoltageOutput;
py	from yocto_voltageoutput import *
php	require_once('yocto_voltageoutput.php');
es	in HTML: <script src="../../lib/yocto_voltageoutput.js"></script> in node.js: require('yoctolib-es2017/yocto_voltageoutput.js');

Fonction globales

yFindVoltageOutput(**func**)

Permet de retrouver une sortie de tension d'après un identifiant donné.

yFindVoltageOutputInContext(**yctx**, **func**)

Permet de retrouver une sortie de tension d'après un identifiant donné dans un Context YAPI.

yFirstVoltageOutput()

Commence l'énumération des sortie de tension accessibles par la librairie.

yFirstVoltageOutputInContext(**yctx**)

Commence l'énumération des sortie de tension accessibles par la librairie.

Méthodes des objets YVoltageOutput

voltageoutput→clearCache()

Invalide le cache.

voltageoutput→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de la sortie sortie de tension au format TYPE(NAME)=SERIAL.FUNCTIONID.

voltageoutput→get_advertisedValue()

Retourne la valeur courante de la sortie sortie de tension (pas plus de 6 caractères).

voltageoutput→get_currentVoltage()

Retourne la tension de sortie, en V.

voltageoutput→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la sortie sortie de tension.

voltageoutput→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la sortie sortie de tension.

voltageoutput→get_friendlyName()

Retourne un identifiant global de la sortie sortie de tension au format NOM_MODULE.NOM_FONCTION.

voltageoutput→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

voltageoutput→get_functionId()

Retourne l'identifiant matériel de la sortie sortie de tension, sans référence au module.

voltageoutput→get_hardwareId()

Retourne l'identifiant matériel unique de la sortie sortie de tension au format `SERIAL.FUNCTIONID`.

voltageoutput→get_logicalName()

Retourne le nom logique de la sortie sortie de tension.

voltageoutput→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

voltageoutput→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

voltageoutput→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

voltageoutput→get_voltageAtStartUp()

Retourne la tension sélectionnée au démarrage du module, en V.

voltageoutput→isOnline()

Vérifie si le module hébergeant la sortie sortie de tension est joignable, sans déclencher d'erreur.

voltageoutput→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant la sortie sortie de tension est joignable, sans déclencher d'erreur.

voltageoutput→load(**msValidity**)

Met en cache les valeurs courantes de la sortie sortie de tension, avec une durée de validité spécifiée.

voltageoutput→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

voltageoutput→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes de la sortie sortie de tension, avec une durée de validité spécifiée.

voltageoutput→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

voltageoutput→nextVoltageOutput()

Continue l'énumération des sortie de tension commencée à l'aide de `yFirstVoltageOutput()`.

voltageoutput→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

voltageoutput→set_currentVoltage(**newval**)

Modifie la tension de sortie, en V.

voltageoutput→set_logicalName(**newval**)

Modifie le nom logique de la sortie sortie de tension.

voltageoutput→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`.

voltageoutput→set_voltageAtStartUp(**newval**)

Modifie la valeur de tension au démarrage du module.

voltageoutput→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

voltageoutput→voltageMove(**V_target**, **ms_duration**)

Déclenche une transition progressive de la tension de sortie.

voltageoutput→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.

YVoltageOutput.FindVoltageOutput() YVoltageOutput yFindVoltageOutput() YVoltageOutput.FindVoltageOutput() YVoltageOutput.FindVoltageOutput()

Permet de retrouver une sortie de tension d'après un identifiant donné.

function **FindVoltageOutput**(**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 sortie sortie de tension soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVoltageOutput.isOnline()` pour tester si la sortie sortie 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence la sortie sortie de tension sans ambiguïté

Retourne :

un objet de classe `YVoltageOutput` qui permet ensuite de contrôler la sortie sortie de tension.

YVoltageOutput.FindVoltageOutputInContext() YVoltageOutput yFindVoltageOutputInContext() YVoltageOutput.FindVoltageOutputInContext() YVoltageOutput.FindVoltageOutputInContext()

Permet de retrouver une sortie de tension d'après un identifiant donné dans un Context YAPI.

function **FindVoltageOutputInContext**(**yctx**, **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 sortie sortie de tension soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVoltageOutput.isOnline()` pour tester si la sortie sortie 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence la sortie sortie de tension sans ambiguïté

Retourne :

un objet de classe `YVoltageOutput` qui permet ensuite de contrôler la sortie sortie de tension.

YVoltageOutput.FirstVoltageOutput() YVoltageOutput yFirstVoltageOutput() YVoltageOutput.FirstVoltageOutput() YVoltageOutput.FirstVoltageOutput()

Commence l'énumération des sortie de tension accessibles par la librairie.

function **FirstVoltageOutput**( )

Utiliser la fonction `YVoltageOutput.nextVoltageOutput()` pour itérer sur les autres sortie de tension.

Retourne :

un pointeur sur un objet `YVoltageOutput`, correspondant à la première sortie de tension accessible en ligne, ou `null` si il n'y a pas de sortie de tension disponibles.

YVoltageOutput.FirstVoltageOutputInContext() YVoltageOutput yFirstVoltageOutputInContext() YVoltageOutput.FirstVoltageOutputInContext() YVoltageOutput.FirstVoltageOutputInContext()

Commence l'énumération des sortie de tension accessibles par la librairie.

function **FirstVoltageOutputInContext**(**yctx**)

Utiliser la fonction `YVoltageOutput.nextVoltageOutput()` pour itérer sur les autres sortie de tension.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YVoltageOutput`, correspondant à la première sortie de tension accessible en ligne, ou `null` si il n'y a pas de sortie de tension disponibles.

voltageoutput→clearCache() YVoltageOutput
voltageoutput.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de la sortie sortie de tension. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

voltageoutput→describe()voltageoutput.describe() YVoltageOutput

Retourne un court texte décrivant de manière non-ambigüe l'instance de la sortie sortie de tension au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant la sortie sortie de tension (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

voltageoutput→get_advertisedValue() YVoltageOutput
voltageoutput→advertisedValue()
voltageoutput.get_advertisedValue()
voltageoutput.get_advertisedValue()

Retourne la valeur courante de la sortie sortie de tension (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :

une chaîne de caractères représentant la valeur courante de la sortie sortie de tension (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

voltageoutput→get_currentVoltage() YVoltageOutput
voltageoutput→currentVoltage()
voltageoutput.get_currentVoltage()
voltageoutput.get_currentVoltage()

Retourne la tension de sortie, en V.

function **get_currentVoltage**( )

Retourne :
une valeur numérique représentant la tension de sortie, en V

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVOLTAGE_INVALID`.

voltageoutput→get_errorMessage() YVoltageOutput
voltageoutput→errorMessage()
voltageoutput.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la sortie sortie 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 sortie sortie de tension.

voltageoutput→get_errorType() YVoltageOutput
voltageoutput→errorType()
voltageoutput.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la sortie sortie 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 sortie sortie de tension.

voltageoutput→get_friendlyName() YVoltageOutput
voltageoutput→friendlyName()
voltageoutput.get_friendlyName()

Retourne un identifiant global de la sortie sortie de tension au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et de la sortie sortie de tension si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel de la sortie sortie de tension (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant la sortie sortie de tension en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

voltageoutput→get_functionDescriptor() YVoltageOutput
voltageoutput→functionDescriptor()
voltageoutput.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

voltageoutput→get_functionId() YVoltageOutput
voltageoutput→functionId()
voltageoutput.get_functionId()

Retourne l'identifiant matériel de la sortie sortie de tension, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant la sortie sortie de tension (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

voltageoutput→get_hardwareId() YVoltageOutput
voltageoutput→hardwareId()
voltageoutput.get_hardwareId()

Retourne l'identifiant matériel unique de la sortie sortie 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 sortie sortie de tension (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant la sortie sortie de tension (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

voltageoutput→get_logicalName() YVoltageOutput
voltageoutput→logicalName()
voltageoutput.get_logicalName()
voltageoutput.get_logicalName()

Retourne le nom logique de la sortie sortie de tension.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique de la sortie sortie de tension.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

voltageoutput→get_module() YVoltageOutput
voltageoutput→module()voltageoutput.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

voltageoutput→get_userData() YVoltageOutput
voltageoutput→userData()
voltageoutput.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.

voltageoutput→get_voltageAtStartUp() YVoltageOutput
voltageoutput→voltageAtStartUp()
voltageoutput.get_voltageAtStartUp()
voltageoutput.get_voltageAtStartUp()

Retourne la tension sélectionnée au démarrage du module, en V.

function **get_voltageAtStartUp**( )

Retourne :
une valeur numérique représentant la tension sélectionnée au démarrage du module, en V

En cas d'erreur, déclenche une exception ou retourne `Y_VOLTAGEATSTARTUP_INVALID`.

voltageoutput→isOnline()voltageoutput.isOnline() YVoltageOutput

Vérifie si le module hébergeant la sortie sortie de tension est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache de la sortie sortie 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 sortie sortie de tension est joignable, `false` sinon

voltageoutput→load()voltageoutput.load() YVoltageOutput

Met en cache les valeurs courantes de la sortie sortie 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 occasionellement 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.

voltageoutput→loadAttribute() YVoltageOutput
voltageoutput.loadAttribute()
voltageoutput.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

voltageoutput→muteValueCallbacks() YVoltageOutput
voltageoutput.muteValueCallbacks()
voltageoutput.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltageoutput→nextVoltageOutput() YVoltageOutput
voltageoutput.nextVoltageOutput()
voltageoutput.nextVoltageOutput()

Continue l'énumération des sortie de tension commencée à l'aide de `yFirstVoltageOutput()`.

function **nextVoltageOutput**( )

Retourne :

un pointeur sur un objet `YVoltageOutput` accessible en ligne, ou `null` lorsque l'énumération est terminée.

voltageoutput→registerValueCallback() YVoltageOutput
voltageoutput.registerValueCallback()
voltageoutput.registerValueCallback()

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.

voltageoutput→set_currentVoltage() YVoltageOutput
voltageoutput→setCurrentVoltage()
voltageoutput.set_currentVoltage()
voltageoutput.set_currentVoltage()

Modifie la tension de sortie, en V.

function **set_currentVoltage**(**newval**)

Les valeurs admises sont de 0 à 10V.

Paramètres :

newval une valeur numérique représentant la tension de sortie, en V

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltageoutput→set_logicalName() YVoltageOutput
voltageoutput→setLogicalName()
voltageoutput.set_logicalName()
voltageoutput.set_logicalName()

Modifie le nom logique de la sortie sortie 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 sortie sortie 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.

voltageoutput→set_userData() YVoltageOutput
voltageoutput→setUserData()
voltageoutput.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

voltageoutput→set_voltageAtStartUp() YVoltageOutput
voltageoutput→setVoltageAtStartUp()
voltageoutput.set_voltageAtStartUp()
voltageoutput.set_voltageAtStartUp()

Modifie la valeur de tension au démarrage du module.

function **set_voltageAtStartUp**(**newval**)

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval une valeur numérique représentant la valeur de tension 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.

voltageoutput→unmuteValueCallbacks() YVoltageOutput
voltageoutput.unmuteValueCallbacks()
voltageoutput.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltageoutput→voltageMove() YVoltageOutput
voltageoutput.voltageMove()
voltageoutput.voltageMove()

Déclenche une transition progressive de la tension de sortie.

function **voltageMove**(**V_target**, **ms_duration**)

N'importe quel changement explicite de tension annulera tout processus de transition en cours.

Paramètres :

V_target	nouvelle valeur de tension à la fin de la transition (nombre flottant, représentant la tension en V)
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.

voltageoutput→wait_async() YVoltageOutput
voltageoutput.wait_async()

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.

3.69. Interface de la fonction Source de tension

La librairie de programmation Yoctopuce permet de commande la tension de srotir du module. Vous pouvez affecter une valeur fixe,ou faire des transition 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**)

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.

3.70. 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>
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;
uwp     import com.yoctopuce.YoctoAPI.YWakeUpMonitor;
py      from yocto_wakeupmonitor import *
php     require_once('yocto_wakeupmonitor.php');
es      in HTML: <script src="../../lib/yocto_wakeupmonitor.js"></script>
        in node.js: require('yoctolib-es2017/yocto_wakeupmonitor.js');
```

Fonction globales
yFindWakeUpMonitor(**func**) Permet de retrouver un moniteur d'après un identifiant donné.
yFindWakeUpMonitorInContext(**yctx**, **func**) Permet de retrouver un moniteur d'après un identifiant donné dans un Context YAPI.
yFirstWakeUpMonitor() Commence l'énumération des Moniteurs accessibles par la librairie.
yFirstWakeUpMonitorInContext(**yctx**) Commence l'énumération des Moniteurs accessibles par la librairie.
Méthodes des objets `YWakeUpMonitor`
wakeupmonitor→clearCache() Invalide le cache.
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.

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→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

wakeupmonitor→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

wakeupmonitor→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

wakeupmonitor→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.

wakeupmonitor→wakeUp()

Force un réveil.

YWakeUpMonitor.FindWakeUpMonitor() YWakeUpMonitor yFindWakeUpMonitor() YWakeUpMonitor.FindWakeUpMonitor() YWakeUpMonitor.FindWakeUpMonitor()

Permet de retrouver un moniteur d'après un identifiant donné.

```
function FindWakeUpMonitor( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindWakeUpMonitorInContext() YWakeUpMonitor yFindWakeUpMonitorInContext()
YWakeUpMonitor.FindWakeUpMonitorInContext()
YWakeUpMonitor.FindWakeUpMonitorInContext()

Permet de retrouver un moniteur d'après un identifiant donné dans un Context YAPI.

function **FindWakeUpMonitorInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le moniteur sans ambiguïté

Retourne :

un objet de classe `YWakeUpMonitor` qui permet ensuite de contrôler le moniteur.

YWakeUpMonitor.FirstWakeUpMonitor() YWakeUpMonitor yFirstWakeUpMonitor() YWakeUpMonitor.FirstWakeUpMonitor() YWakeUpMonitor.FirstWakeUpMonitor()

Commence l'énumération des Moniteurs accessibles par la librairie.

function **FirstWakeUpMonitor**( )

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.

YWakeUpMonitor.FirstWakeUpMonitorInContext() YWakeUpMonitor yFirstWakeUpMonitorInContext()
YWakeUpMonitor.FirstWakeUpMonitorInContext()
YWakeUpMonitor.FirstWakeUpMonitorInContext()

Commence l'énumération des Moniteurs accessibles par la librairie.

```
function FirstWakeUpMonitorInContext( yctx)
```

Utiliser la fonction `YWakeUpMonitor.nextWakeUpMonitor()` pour itérer sur les autres Moniteurs.

Paramètres :

yctx un contexte YAPI.

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→clearCache() YWakeUpMonitor
wakeupmonitor.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du moniteur. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

wakeupmonitor→describe() YWakeUpMonitor
wakeupmonitor.describe()

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 utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le moniteur (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

wakeupmonitor→get_advertisedValue() YWakeUpMonitor
wakeupmonitor→advertisedValue()
wakeupmonitor.get_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'identifant 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 example `relay1`.

Retourne :

une chaîne de caractères identifiant le moniteur (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

wakeupmonitor→get_hardwareId() 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 example 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()
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_nextWakeUp() YWakeUpMonitor
wakeupmonitor→nextWakeUp()
wakeupmonitor.get_nextWakeUp()
wakeupmonitor.get_nextWakeUp()

Retourne la prochaine date/heure de réveil agendée (format UNIX).

function **get_nextWakeUp**( )

Retourne :
un entier représentant la prochaine date/heure de réveil agendée (format UNIX)

En cas d'erreur, déclenche une exception ou retourne `Y_NEXTWAKEUP_INVALID`.

wakeupmonitor→get_powerDuration() YWakeUpMonitor
wakeupmonitor→powerDuration()
wakeupmonitor.get_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()
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_SLEEPCOUNTDOWN_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()
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_SCHEDULE1 et Y_WAKEUPREASON_SCHEDULE2

En cas d'erreur, déclenche une exception ou retourne Y_WAKEUPREASON_INVALID.

wakeupmonitor→get_wakeUpState() YWakeUpMonitor
wakeupmonitor→wakeUpState()
wakeupmonitor.get_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→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 occasionellement 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→loadAttribute() YWakeUpMonitor
wakeupmonitor.loadAttribute()
wakeupmonitor.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

wakeupmonitor→muteValueCallbacks() YWakeUpMonitor
wakeupmonitor.muteValueCallbacks()
wakeupmonitor.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→nextWakeUpMonitor() YWakeUpMonitor
wakeupmonitor.nextWakeUpMonitor()
wakeupmonitor.nextWakeUpMonitor()

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() YWakeUpMonitor
wakeupmonitor.registerValueCallback()
wakeupmonitor.registerValueCallback()

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() YWakeUpMonitor
wakeupmonitor.resetSleepCountDown()
wakeupmonitor.resetSleepCountDown()

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() YWakeUpMonitor
wakeupmonitor→setLogicalName()
wakeupmonitor.set_logicalName()
wakeupmonitor.set_logicalName()

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()
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()
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()
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'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

wakeupmonitor→sleep()wakeupmonitor.sleep() YWakeUpMonitor wakeupmonitor.sleep()

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() YWakeUpMonitor wakeupmonitor.sleepFor()wakeupmonitor.sleepFor()

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 nombre de secondes avant le prochain réveil

secBeforeSleep nombre de secondes avant la mise en sommeil

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→sleepUntil() YWakeUpMonitor
wakeupmonitor.sleepUntil()
wakeupmonitor.sleepUntil()

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→unmuteValueCallbacks() YWakeUpMonitor
wakeupmonitor.unmuteValueCallbacks()
wakeupmonitor.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→wait_async() YWakeUpMonitor
wakeupmonitor.wait_async()

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 wakeupmonitor.wakeUp()

Force un réveil.

function **wakeUp**( )

3.71. 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>
cpp     #include "yocto_wakeupschedule.h"
m       #import "yocto_wakeupschedule.h"
pas     uses yocto_wakeupschedule;
vb      yocto_wakeupschedule.vb
cs      yocto_wakeupschedule.cs
java    import com.yoctopuce.YoctoAPI.YWakeUpSchedule;
uwp     import com.yoctopuce.YoctoAPI.YWakeUpSchedule;
py      from yocto_wakeupschedule import *
php     require_once('yocto_wakeupschedule.php');
es      in HTML: <script src="../../lib/yocto_wakeupschedule.js"></script>
        in node.js: require('yoctolib-es2017/yocto_wakeupschedule.js');
```

Fonction globales
yFindWakeUpSchedule(**func**) Permet de retrouver un réveil agendé d'après un identifiant donné.
yFindWakeUpScheduleInContext(**yctx**, **func**) Permet de retrouver un réveil agendé d'après un identifiant donné dans un Context YAPI.
yFirstWakeUpSchedule() Commence l'énumération des réveils agendés accessibles par la librairie.
yFirstWakeUpScheduleInContext(**yctx**) Commence l'énumération des réveils agendés accessibles par la librairie.
Méthodes des objets YWakeUpSchedule
wakeupschedule→clearCache() Invalide le cache.
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'interval 00-29 de chaque heures où le réveil est actif.

wakeupschedule→get_minutesB()

Retourne les minutes de l'interval 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 occurence 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→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

wakeupschedule→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

wakeupschedule→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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**)

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)

Modifie les minutes de l'interval 00-29 où un réveil doit avoir lieu.

wakeupschedule→set_minutesB(newval)

Modifie les minutes de l'interval 30-59 où un réveil doit avoir lieu.

wakeupschedule→set_monthDays(newval)

Modifie les jours du mois où un réveil doit avoir lieu.

wakeupschedule→set_months(newval)

Modifie les mois où un réveil doit avoir lieu.

wakeupschedule→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

wakeupschedule→set_weekDays(newval)

Modifie les jours de la semaine où un réveil doit avoir lieu.

wakeupschedule→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

wakeupschedule→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YWakeUpSchedule.FindWakeUpSchedule() YWakeUpSchedule yFindWakeUpSchedule() YWakeUpSchedule.FindWakeUpSchedule() YWakeUpSchedule.FindWakeUpSchedule()

Permet de retrouver un réveil agendé d'après un identifiant donné.

function **FindWakeUpSchedule**(**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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindWakeUpScheduleInContext() YWakeUpSchedule yFindWakeUpScheduleInContext()

Permet de retrouver un réveil agendé d'après un identifiant donné dans un Context YAPI.

function **FindWakeUpScheduleInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le réveil agendé sans ambiguïté

Retourne :

un objet de classe `YWakeUpSchedule` qui permet ensuite de contrôler le réveil agendé.

YWakeUpSchedule.FirstWakeUpSchedule() YWakeUpSchedule yFirstWakeUpSchedule() YWakeUpSchedule.FirstWakeUpSchedule() YWakeUpSchedule.FirstWakeUpSchedule()

Commence l'énumération des réveils agendés accessibles par la librairie.

function **FirstWakeUpSchedule**( )

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.

YWakeUpSchedule.FirstWakeUpScheduleInContext() YWakeUpSchedule yFirstWakeUpScheduleInContext() YWakeUpSchedule.FirstWakeUpScheduleInContext() YWakeUpSchedule.FirstWakeUpScheduleInContext()

Commence l'énumération des réveils agendés accessibles par la librairie.

function **FirstWakeUpScheduleInContext**(**yctx**)

Utiliser la fonction `YWakeUpSchedule.nextWakeUpSchedule()` pour itérer sur les autres réveils agendés.

Paramètres :

yctx un contexte YAPI.

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→clearCache() YWakeUpSchedule
wakeupschedule.clearCache()

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du réveil agendé. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

wakeupschedule→describe() YWakeUpSchedule
wakeupschedule.describe()

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 utilsé 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 debuggeur.

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()
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'identifant 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 example `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 example 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()
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()
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()
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()
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()
wakeupschedule.get_minutesB()

Retourne les minutes de l'interval 30-59 de chaque heure où le réveil est actif.

function **get_minutesB**( )

Retourne :
un entier représentant les minutes de l'interval 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_monthDays() YWakeUpSchedule
wakeupschedule→monthDays()
wakeupschedule.get_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()
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()
wakeupschedule.get_nextOccurence()

Retourne la date/heure de la prochaine occurence de réveil.

function **get_nextOccurence**( )

Retourne :
un entier représentant la date/heure de la prochaine occurence de réveil

En cas d'erreur, déclenche une exception ou retourne `Y_NEXTOCCURENCE_INVALID`.

wakeupschedule→get_userData() 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()
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() YWakeUpSchedule
wakeupschedule.isOnline()

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→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 occasionellement 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→loadAttribute() YWakeUpSchedule
wakeupschedule.loadAttribute()
wakeupschedule.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

function **loadAttribute**(**attrName**)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

wakeupschedule→muteValueCallbacks() YWakeUpSchedule
wakeupschedule.muteValueCallbacks()
wakeupschedule.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→nextWakeUpSchedule() YWakeUpSchedule
wakeupschedule.nextWakeUpSchedule()
wakeupschedule.nextWakeUpSchedule()

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() YWakeUpSchedule
wakeupschedule.registerValueCallback()
wakeupschedule.registerValueCallback()

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()
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

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()
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()
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()
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

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()
wakeupschedule.set_minutesB()

Modifie les minutes de l'interval 30-59 où un réveil doit avoir lieu.

function **set_minutesB**(**newval**)

Paramètres :

newval un entier représentant les minutes de l'interval 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()
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

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()
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

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()
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

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→unmuteValueCallbacks() YWakeUpSchedule
wakeupschedule.unmuteValueCallbacks()
wakeupschedule.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→wait_async() YWakeUpSchedule
wakeupschedule.wait_async()

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.

3.72. 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 intervale régulier pour l'empêcher de provoquer la réinitialisation. Le watchog 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	<script type='text/javascript' src='yocto_watchdog.js'></script>
cpp	#include "yocto_watchdog.h"
m	#import "yocto_watchdog.h"
pas	uses yocto_watchdog;
vb	yocto_watchdog.vb
cs	yocto_watchdog.cs
java	import com.yoctopuce.YoctoAPI.YWatchdog;
uwp	import com.yoctopuce.YoctoAPI.YWatchdog;
py	from yocto_watchdog import *
php	require_once('yocto_watchdog.php');
es	in HTML: <script src="../../lib/yocto_watchdog.js"></script> in node.js: require('yoctolib-es2017/yocto_watchdog.js');

Fonction globales

yFindWatchdog(**func**)

Permet de retrouver un watchdog d'après un identifiant donné.

yFindWatchdogInContext(**yctx**, **func**)

Permet de retrouver un watchdog d'après un identifiant donné dans un Context YAPI.

yFirstWatchdog()

Commence l'énumération des watchdog accessibles par la librairie.

yFirstWatchdogInContext(**yctx**)

Commence l'énumération des watchdog accessibles par la librairie.

Méthodes des objets YWatchdog

watchdog→clearCache()

Invalide le cache.

watchdog→delayedPulse(**ms_delay**, **ms_duration**)

Pré-programme une impulsion

watchdog→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du watchdog au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

watchdog→get_advertisedValue()

Retourne la valeur courante du watchdog (pas plus de 6 caractères).

watchdog→get_autoStart()

Retourne l'état du watchdog à la mise sous tension du module.

watchdog→get_countdown()

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse().

watchdog→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

watchdog→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

watchdog→get_friendlyName()

Retourne un identifiant global du watchdog au format `NOM_MODULE.NOM_FONCTION`.

watchdog→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

watchdog→get_functionId()

Retourne l'identifiant matériel du watchdog, sans référence au module.

watchdog→get_hardwareId()

Retourne l'identifiant matériel unique du watchdog au format `SERIAL.FUNCTIONID`.

watchdog→get_logicalName()

Retourne le nom logique du watchdog.

watchdog→get_maxTimeOnStateA()

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

watchdog→get_maxTimeOnStateB()

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

watchdog→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

watchdog→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

watchdog→get_output()

Retourne l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

watchdog→get_pulseTimer()

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

watchdog→get_running()

Retourne l'état du watchdog.

watchdog→get_state()

Retourne l'état du watchdog (A pour la position de repos, B pour l'état actif).

watchdog→get_stateAtPowerOn()

Retourne l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

watchdog→get_triggerDelay()

Retourne le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchog, en millisecondes.

watchdog→get_triggerDuration()

Retourne la durée d'un reset généré par le watchog, 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→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

watchdog→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

watchdog→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

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 watchog, en millisecondes.

watchdog→set_triggerDuration(newval)

Modifie la durée des resets générés par le watchog, 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→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

watchdog→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YWatchdog.FindWatchdog() YWatchdog yFindWatchdog()YWatchdog.FindWatchdog() YWatchdog.FindWatchdog()

Permet de retrouver un watchdog d'après un identifiant donné.

```
function FindWatchdog( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindWatchdogInContext() YWatchdog
yFindWatchdogInContext()
YWatchdog.FindWatchdogInContext()
YWatchdog.FindWatchdogInContext()

Permet de retrouver un watchdog d'après un identifiant donné dans un Context YAPI.

function **FindWatchdogInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

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() YWatchdog yFirstWatchdog()YWatchdog.FirstWatchdog() YWatchdog.FirstWatchdog()

Commence l'énumération des watchdog accessibles par la librairie.

function **FirstWatchdog**( )

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.

YWatchdog.FirstWatchdogInContext() YWatchdog
yFirstWatchdogInContext()
YWatchdog.FirstWatchdogInContext()
YWatchdog.FirstWatchdogInContext()

Commence l'énumération des watchdog accessibles par la librairie.

function **FirstWatchdogInContext**(**yctx**)

Utiliser la fonction `YWatchdog.nextWatchdog()` pour itérer sur les autres watchdog.

Paramètres :

yctx un contexte YAPI.

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→clearCache()watchdog.clearCache() YWatchdog

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du watchdog. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

watchdog→delayedPulse()watchdog.delayedPulse() YWatchdog

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.

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 utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le watchdog (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

watchdog→get_advertisedValue() YWatchdog
watchdog→advertisedValue()
watchdog.get_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()
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()
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'identifant 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 example `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 example 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()
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()
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()
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_output() YWatchdog
watchdog→output()watchdog.get_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()
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()
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()
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()
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()
watchdog.get_triggerDelay()

Retourne le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchog, en millisecondes.

function **get_triggerDelay**( )

Retourne :

un entier représentant le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchog, en millisecondes

En cas d'erreur, déclenche une exception ou retourne `Y_TRIGGERDELAY_INVALID`.

watchdog→get_triggerDuration() YWatchdog
watchdog→triggerDuration()
watchdog.get_triggerDuration()
watchdog.get_triggerDuration()

Retourne la durée d'un reset généré par le watchog, en millisecondes.

function **get_triggerDuration**( )

Retourne :
un entier représentant la durée d'un reset généré par le watchog, 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→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 occasionellement 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→loadAttribute()watchdog.loadAttribute() YWatchdog watchdog.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

watchdog→muteValueCallbacks() YWatchdog
watchdog.muteValueCallbacks()
watchdog.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→nextWatchdog() YWatchdog
watchdog.nextWatchdog()watchdog.nextWatchdog()

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()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() YWatchdog
watchdog.registerValueCallback()
watchdog.registerValueCallback()

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() YWatchdog
watchdog.resetWatchdog()
watchdog.resetWatchdog()

Réinitialise le WatchDog.

function **resetWatchdog**( )

Quand le watchdog est en fonctionnement cette fonction doit être appelée à interval régulier, pour empécher 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()
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()
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()
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()
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()
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()
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()
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()
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()
watchdog.set_triggerDelay()

Modifie le délai d'attente avant qu'un reset ne soit généré par le watchog, en millisecondes.

function **set_triggerDelay**(**newval**)

Paramètres :

newval un entier représentant le délai d'attente avant qu'un reset ne soit généré par le watchog, en millisecondes

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()
watchdog.set_triggerDuration()

Modifie la durée des resets générés par le watchog, en millisecondes.

function **set_triggerDuration**(**newval**)

Paramètres :

newval un entier représentant la durée des resets générés par le watchog, en millisecondes

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→unmuteValueCallbacks() YWatchdog
watchdog.unmuteValueCallbacks()
watchdog.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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.

3.73. Interface de la fonction WeighScale

La classe YWeighScale permet d'obtenir une mesure de poids à partir d'une cellule de poids ratiométrique. Elle permet de contrôler le mode d'excitation de la cellule, pour éviter les dérives liées aux changements de température de l'électronique, et d'appliquer automatiquement une correction supplémentaire en fonction de la température pour compenser la dérive de la cellule de poids elle-même.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_weighscale.js'></script>
cpp     #include "yocto_weighscale.h"
m       #import "yocto_weighscale.h"
pas     uses yocto_weighscale;
vb      yocto_weighscale.vb
cs      yocto_weighscale.cs
java    import com.yoctopuce.YoctoAPI.YWeighScale;
uwp     import com.yoctopuce.YoctoAPI.YWeighScale;
py      from yocto_weighscale import *
php     require_once('yocto_weighscale.php');
es      in HTML: <script src="../../lib/yocto_weighscale.js"></script>
        in node.js: require('yoctolib-es2017/yocto_weighscale.js');
```

Fonction globales
yFindWeighScale(**func**) Permet de retrouver un capteur de poids d'après un identifiant donné.
yFindWeighScaleInContext(**yctx**, **func**) Permet de retrouver un capteur de poids d'après un identifiant donné dans un Context YAPI.
yFirstWeighScale() Commence l'énumération des capteur de poids accessibles par la librairie.
yFirstWeighScaleInContext(**yctx**) Commence l'énumération des capteur de poids accessibles par la librairie.
Méthodes des objets YWeighScale
weighscale→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.
weighscale→clearCache() Invalide le cache.
weighscale→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de poids au format TYPE(NAME)=SERIAL.FUNCTIONID.
weighscale→get_adaptRatio() Retourne le taux d'adaptation de la température de compensation, en pour cents.
weighscale→get_advertisedValue() Retourne la valeur courante du capteur de poids (pas plus de 6 caractères).
weighscale→get_compTemperature() Retourne la température utilisée actuellement pour la compensation thermique.
weighscale→get_compensation() Retourne la valeur de la compensation thermique courante.
weighscale→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule.

weighscale→get_currentValue()

Retourne la valeur actuelle de la measure, en l'unité spécifiée, sous forme de nombre à virgule.

weighscale→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

weighscale→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de poids.

weighscale→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de poids.

weighscale→get_excitation()

Retourne la méthode d'excitation de la cellule de poids.

weighscale→get_friendlyName()

Retourne un identifiant global du capteur de poids au format `NOM_MODULE.NOM_FONCTION`.

weighscale→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

weighscale→get_functionId()

Retourne l'identifiant matériel du capteur de poids, sans référence au module.

weighscale→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de poids au format `SERIAL.FUNCTIONID`.

weighscale→get_highestValue()

Retourne la valeur maximale observée pour la measure depuis le démarrage du module.

weighscale→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.

weighscale→get_logicalName()

Retourne le nom logique du capteur de poids.

weighscale→get_lowestValue()

Retourne la valeur minimale observée pour la measure depuis le démarrage du module.

weighscale→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

weighscale→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

weighscale→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.

weighscale→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

weighscale→get_resolution()

Retourne la résolution des valeurs mesurées.

weighscale→get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

weighscale→get_unit()

Retourne l'unité dans laquelle la measure est exprimée.

weighscale→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

weighscale→get_zeroTracking()

Retourne la valeur seuil pour le suivi continu du zéro.

weighscale→isOnline()

Vérifie si le module hébergeant le capteur de poids est joignable, sans déclencher d'erreur.

weighscale→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le capteur de poids est joignable, sans déclencher d'erreur.

weighscale→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

weighscale→load(**msValidity**)

Met en cache les valeurs courantes du capteur de poids, avec une durée de validité spécifiée.

weighscale→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

weighscale→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

weighscale→loadOffsetCompensationTable(**tempValues**, **compValues**)

Récupère la table de compensation en température de l'offset de poids précédemment enregistrée à l'aide de la fonction `set_offsetCompensationTable`.

weighscale→loadSpanCompensationTable(**tempValues**, **compValues**)

Récupère la table de compensation en température de l'amplitude de poids précédemment enregistrée à l'aide de la fonction `set_spanCompensationTable`.

weighscale→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du capteur de poids, avec une durée de validité spécifiée.

weighscale→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

weighscale→nextWeighScale()

Continue l'énumération des capteur de poids commencée à l'aide de `yFirstWeighScale()`.

weighscale→registerTimedReportCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

weighscale→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

weighscale→set_adaptRatio(**newval**)

Modifie le taux d'adaptation de la température de compensation, en pour cents.

weighscale→set_excitation(**newval**)

Modifie la méthode d'excitation de la cellule de poids.

weighscale→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

weighscale→set_logFrequency(**newval**)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

weighscale→set_logicalName(**newval**)

Modifie le nom logique du capteur de poids.

weighscale→set_lowestValue(**newval**)

Modifie la mémoire de valeur minimale observée.

weighscale→set_offsetCompensationTable(**tempValues**, **compValues**)

Enregistre la table de compensation en température de l'offset de poids, afin de pouvoir corriger automatiquement le poids mesuré sur la base de la température de compensation.

weighscale→set_reportFrequency(**newval**)

Modifie la fréquence de notification périodique des valeurs mesurées.

weighscale→set_resolution(**newval**)

Modifie la résolution des valeurs physique mesurées.

weighscale→set_spanCompensationTable(**tempValues**, **compValues**)

Enregistre la table de compensation en température de l'amplitude du poids, afin de pouvoir corriger automatiquement le poids mesuré sur la base de la température de compensation.

weighscale→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`.

weighscale→set_zeroTracking(**newval**)

Modifie le taux d'adaptation de la température de compensation, en pour cents.

weighscale→setupSpan(**currWeight**, **maxWeight**)

Configure le facteur de poids et l'amplitude maximale de la cellule de poids (stockés dans le genericSensor correspondant) de sorte à ce que le signal actuel corresponde au poids de référence spécifié.

weighscale→startDataLogger()

Démarre l'enregistreur de données du module.

weighscale→stopDataLogger()

Arrête l'enregistreur de données du module.

weighscale→tare()

Adapte le biais de mesure de la cellule de poids (stocké dans le genericSensor correspondant) de sorte à ce que le signal actuel corresponde à un poids nul.

weighscale→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

weighscale→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.

YWeighScale.FindWeighScale() YWeighScale yFindWeighScale()YWeighScale.FindWeighScale() YWeighScale.FindWeighScale()

Permet de retrouver un capteur de poids d'après un identifiant donné.

```
function FindWeighScale( 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 poids soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWeighScale.isOnline()` pour tester si le capteur de poids 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le capteur de poids sans ambiguïté

Retourne :

un objet de classe `YWeighScale` qui permet ensuite de contrôler le capteur de poids.

YWeighScale.FindWeighScaleInContext() YWeighScale
yFindWeighScaleInContext()
YWeighScale.FindWeighScaleInContext()
YWeighScale.FindWeighScaleInContext()

Permet de retrouver un capteur de poids d'après un identifiant donné dans un Context YAPI.

function **FindWeighScaleInContext**(**yctx**, **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 poids soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWeighScale.isOnline()` pour tester si le capteur de poids 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de poids sans ambiguïté

Retourne :

un objet de classe `YWeighScale` qui permet ensuite de contrôler le capteur de poids.

YWeighScale.FirstWeighScale() YWeighScale yFirstWeighScale()YWeighScale.FirstWeighScale() YWeighScale.FirstWeighScale()

Commence l'énumération des capteur de poids accessibles par la librairie.

function **FirstWeighScale**( )

Utiliser la fonction `YWeighScale.nextWeighScale()` pour itérer sur les autres capteur de poids.

Retourne :

un pointeur sur un objet `YWeighScale`, correspondant au premier capteur de poids accessible en ligne, ou `null` si il n'y a pas de capteur de poids disponibles.

YWeighScale.FirstWeighScaleInContext() YWeighScale
yFirstWeighScaleInContext()
YWeighScale.FirstWeighScaleInContext()
YWeighScale.FirstWeighScaleInContext()

Commence l'énumération des capteur de poids accessibles par la librairie.

function **FirstWeighScaleInContext**(**yctx**)

Utiliser la fonction `YWeighScale.nextWeighScale()` pour itérer sur les autres capteur de poids.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YWeighScale`, correspondant au premier capteur de poids accessible en ligne, ou `null` si il n'y a pas de capteur de poids disponibles.

weighscale→calibrateFromPoints() YWeighScale
weighscale.calibrateFromPoints()
weighscale.calibrateFromPoints()

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

function **calibrateFromPoints**(**rawValues**, **refValues**)

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

weighscale→clearCache()weighscale.clearCache() YWeighScale

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes du capteur de poids. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

weighscale→describe()weighscale.describe() YWeighScale

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de poids au format TYPE(NAME)=SERIAL.FUNCTIONID.

function **describe**( )

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé 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 debuggeur.

Retourne :
une chaîne de caractères décrivant le capteur de poids (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

weighscale→get_adaptRatio() YWeighScale
weighscale→adaptRatio()
weighscale.get_adaptRatio()
weighscale.get_adaptRatio()

Retourne le taux d'adaptation de la température de compensation, en pour cents.

function **get_adaptRatio**( )

La température de compensation est adaptée toutes les 3 secondes, en appliquant ce taux d'adaptation à la différence entre la température ambiante mesurée et la température de compensation actuelle. Le taux d'adaptation maximal est de 65 pourcents.

Retourne :
une valeur numérique représentant le taux d'adaptation de la température de compensation, en pour cents

En cas d'erreur, déclenche une exception ou retourne `Y_ADAPTRATIO_INVALID`.

weighscale→get_advertisedValue() YWeighScale
weighscale→advertisedValue()
weighscale.get_advertisedValue()
weighscale.get_advertisedValue()

Retourne la valeur courante du capteur de poids (pas plus de 6 caractères).

function **get_advertisedValue**( )

Retourne :
une chaîne de caractères représentant la valeur courante du capteur de poids (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

weighscale→get_compTemperature() YWeighScale
weighscale→compTemperature()
weighscale.get_compTemperature()
weighscale.get_compTemperature()

Retourne la température utilisée actuellement pour la compensation thermique.

function **get_compTemperature**( )

Retourne :
une valeur numérique représentant la température utilisée actuellement pour la compensation thermique

En cas d'erreur, déclenche une exception ou retourne `Y_COMPTEMPERATURE_INVALID`.

weighscale→get_compensation() YWeighScale
weighscale→compensation()
weighscale.get_compensation()
weighscale.get_compensation()

Retourne la valeur de la compensation thermique courante.

function **get_compensation**( )

Retourne :
une valeur numérique représentant la valeur de la compensation thermique courante

En cas d'erreur, déclenche une exception ou retourne `Y_COMPENSATION_INVALID`.

weighscale→get_currentRawValue() YWeighScale
weighscale→currentRawValue()
weighscale.get_currentRawValue()
weighscale.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule.

function **get_currentRawValue**( )

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

weighscale→get_currentValue() YWeighScale
weighscale→currentValue()
weighscale.get_currentValue()
weighscale.get_currentValue()

Retourne la valeur actuelle de la measure, en l'unité spécifiée, sous forme de nombre à virgule.

function **get_currentValue**( )

Retourne :

une valeur numérique représentant la valeur actuelle de la measure, en l'unité spécifiée, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

weighscale→get_dataLogger() YWeighScale
weighscale→dataLogger()
weighscale.get_dataLogger()
weighscale.get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

function **get_dataLogger**( )

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet de classe YDataLogger ou null en cas d'erreur.

weighscale→get_errorMessage() YWeighScale
weighscale→errorMessage()
weighscale.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de poids.

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 poids.

weighscale→get_errorType() YWeighScale
weighscale→errorType()weighscale.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de poids.

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 poids.

weighscale→get_excitation() YWeighScale
weighscale→excitation()weighscale.get_excitation()
weighscale.get_excitation()

Retourne la méthode d'excitation de la cellule de poids.

function **get_excitation**( )

Retourne :

une valeur parmi `Y_EXCITATION_OFF`, `Y_EXCITATION_DC` et `Y_EXCITATION_AC` représentant la méthode d'excitation de la cellule de poids

En cas d'erreur, déclenche une exception ou retourne `Y_EXCITATION_INVALID`.

weighscale→get_friendlyName() YWeighScale
weighscale→friendlyName()
weighscale.get_friendlyName()

Retourne un identifiant global du capteur de poids au format NOM_MODULE.NOM_FONCTION.

function **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du capteur de poids si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du capteur de poids (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant le capteur de poids en utilisant les noms logiques (ex: MyCustomName.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FRIENDLYNAME_INVALID.

weighscale→get_functionDescriptor() YWeighScale
weighscale→functionDescriptor()
weighscale.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

weighscale→get_functionId() YWeighScale
weighscale→functionId()weighscale.get_functionId()

Retourne l'identifiant matériel du capteur de poids, sans référence au module.

function **get_functionId**( )

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de poids (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

weighscale→get_hardwareId() YWeighScale
weighscale→hardwareId()
weighscale.get_hardwareId()

Retourne l'identifiant matériel unique du capteur de poids 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 poids (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant le capteur de poids (ex: RELAYLO1-123456.relay1)

En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

weighscale→get_highestValue() YWeighScale
weighscale→highestValue()
weighscale.get_highestValue()
weighscale.get_highestValue()

Retourne la valeur maximale observée pour la measure depuis le démarrage du module.

function **get_highestValue**( )

Retourne :

une valeur numérique représentant la valeur maximale observée pour la measure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

weighscale→get_logFrequency() YWeighScale
weighscale→logFrequency()
weighscale.get_logFrequency()
weighscale.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`.

weighscale→get_logicalName() YWeighScale
weighscale→logicalName()
weighscale.get_logicalName()
weighscale.get_logicalName()

Retourne le nom logique du capteur de poids.

function **get_logicalName**( )

Retourne :
une chaîne de caractères représentant le nom logique du capteur de poids.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

weighscale→get_lowestValue() YWeighScale
weighscale→lowestValue()
weighscale.get_lowestValue()
weighscale.get_lowestValue()

Retourne la valeur minimale observée pour la measure depuis le démarrage du module.

function **get_lowestValue**( )

Retourne :

une valeur numérique représentant la valeur minimale observée pour la measure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

weighscale→get_module() YWeighScale
weighscale→module()weighscale.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

weighscale→get_recordedData() YWeighScale
weighscale→recordedData()
weighscale.get_recordedData()
weighscale.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 find de l'intercalle 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.

weighscale→get_reportFrequency() YWeighScale
weighscale→reportFrequency()
weighscale.get_reportFrequency()
weighscale.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`.

weighscale→get_resolution() YWeighScale
weighscale→resolution()weighscale.get_resolution()
weighscale.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`.

weighscale→get_sensorState() YWeighScale
weighscale→sensorState()
weighscale.get_sensorState()
weighscale.get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

function **get_sensorState**( )

Retourne :
un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `Y_SENSORSTATE_INVALID`.

weighscale→get_unit() YWeighScale
weighscale→unit()weighscale.get_unit()
weighscale.get_unit()

Retourne l'unité dans laquelle la measure est exprimée.

function **get_unit**( )

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la measure est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

weighscale→get_userData() YWeighScale
weighscale→userData()weighscale.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.

weighscale→get_zeroTracking() YWeighScale
weighscale→zeroTracking()
weighscale.get_zeroTracking()
weighscale.get_zeroTracking()

Retourne la valeur seuil pour le suivi continu du zéro.

function **get_zeroTracking**( )

Lorsque ce seuil est supérieure à zéro, les mesures inférieures à cette valeur sont automatiquement ignorées et le zéro est continuellement compensé en conséquence.

Retourne :
une valeur numérique représentant la valeur seuil pour le suivi continu du zéro

En cas d'erreur, déclenche une exception ou retourne `Y_ZEROTRACKING_INVALID`.

weighscale→isOnline()weighscale.isOnline() YWeighScale

Vérifie si le module hébergeant le capteur de poids est joignable, sans déclencher d'erreur.

function **isOnline**( )

Si les valeurs des attributs en cache du capteur de poids sont valides au moment de l'appel, le module est considéré joignable. 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 poids est joignable, `false` sinon

weighscale→load()weighscale.load() YWeighScale

Met en cache les valeurs courantes du capteur de poids, 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 occasionellement 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.

weighscale→loadAttribute() YWeighScale
weighscale.loadAttribute()weighscale.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

weighscale→loadCalibrationPoints() YWeighScale
weighscale.loadCalibrationPoints()
weighscale.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

function **loadCalibrationPoints**(**rawValues**, **refValues**)

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

weighscale→loadOffsetCompensationTable() YWeighScale
weighscale.loadOffsetCompensationTable()
weighscale.loadOffsetCompensationTable()

Récupère la table de compensation en température de l'offset de poids précédemment enregistrée à l'aide de la fonction `set_offsetCompensationTable`.

function **loadOffsetCompensationTable**(**tempValues**, **compValues**)

La correction de poids est faite par interpolation linéaire entre les points spécifiés.

Paramètres :

tempValues tableau de nombres flottants, qui sera rempli par la fonction avec les différentes températures pour lesquelles une correction d'offset est spécifiée.

compValues tableau de nombres flottants, qui sera rempli par la fonction avec la correction d'offset appliquée pour chacun des points de température, index par index.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

weighscale→loadSpanCompensationTable() YWeighScale
weighscale.loadSpanCompensationTable()
weighscale.loadSpanCompensationTable()

Récupère la table de compensation en température de l'amplitude de poids précédemment enregistrée à l'aide de la fonction `set_spanCompensationTable`.

function **loadSpanCompensationTable**(**tempValues**, **compValues**)

La correction de poids est faite par interpolation linéaire entre les points spécifiés.

Paramètres :

tempValues tableau de nombres flottants, qui sera rempli par la fonction avec les différentes températures pour lesquelles une correction d'amplitude est spécifiée.

compValues tableau de nombres flottants, qui sera rempli par la fonction avec la correction d'amplitude appliquée pour chacun des points de température, index par index.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

weighscale→muteValueCallbacks() YWeighScale
weighscale.muteValueCallbacks()
weighscale.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

weighscale→nextWeighScale() YWeighScale
weighscale.nextWeighScale()
weighscale.nextWeighScale()

Continue l'énumération des capteur de poids commencée à l'aide de `yFirstWeighScale()`.

function **nextWeighScale**( )

Retourne :

un pointeur sur un objet `YWeighScale` accessible en ligne, ou `null` lorsque l'énumération est terminée.

weighscale→registerTimedReportCallback() YWeighScale
weighscale.registerTimedReportCallback()
weighscale.registerTimedReportCallback()

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.

weighscale→registerValueCallback() YWeighScale
weighscale.registerValueCallback()
weighscale.registerValueCallback()

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.

weighscale→set_adaptRatio() YWeighScale
weighscale→setAdaptRatio()
weighscale.set_adaptRatio()
weighscale.set_adaptRatio()

Modifie le taux d'adaptation de la température de compensation, en pour cents.

function **set_adaptRatio**(**newval**)

Paramètres :

newval une valeur numérique représentant le taux d'adaptation de la température de compensation, 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.

weighscale→set_excitation() YWeighScale
weighscale→setExcitation()
weighscale.set_excitation()
weighscale.set_excitation()

Modifie la méthode d'excitation de la cellule de poids.

function **set_excitation**(**newval**)

Paramètres :

newval une valeur parmi `Y_EXCITATION_OFF`, `Y_EXCITATION_DC` et `Y_EXCITATION_AC` représentant la méthode d'excitation de la cellule de poids

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

weighscale→set_highestValue() YWeighScale
weighscale→setHighestValue()
weighscale.set_highestValue()
weighscale.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.

weighscale→set_logFrequency() YWeighScale
weighscale→setLogFrequency()
weighscale.set_logFrequency()
weighscale.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.

weighscale→set_logicalName() YWeighScale
weighscale→setLogicalName()
weighscale.set_logicalName()
weighscale.set_logicalName()

Modifie le nom logique du capteur de poids.

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 poids.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

weighscale→set_lowestValue() YWeighScale
weighscale→setLowestValue()
weighscale.set_lowestValue()
weighscale.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.

weighscale→set_offsetCompensationTable() YWeighScale
weighscale→setOffsetCompensationTable()
weighscale.set_offsetCompensationTable()
weighscale.set_offsetCompensationTable()

Enregistre la table de compensation en température de l'offset de poids, afin de pouvoir corriger automatiquement le poids mesuré sur la base de la température de compensation.

function **set_offsetCompensationTable**(**tempValues**, **compValues**)

La correction de poids sera faite par interpolation linéaire entre les points spécifiés.

Paramètres :

tempValues tableau de nombres flottants, correspondant aux différentes températures pour lesquelles une correction d'offset est spécifiée.

compValues tableau de nombres flottants, correspondant à la correction d'offset à appliquer pour chacun des points de température, index par index.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

weighscale→set_reportFrequency() YWeighScale
weighscale→setReportFrequency()
weighscale.set_reportFrequency()
weighscale.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.

weighscale→set_resolution() YWeighScale
weighscale→setResolution()
weighscale.set_resolution()
weighscale.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.

weighscale→set_spanCompensationTable() YWeighScale
weighscale→setSpanCompensationTable()
weighscale.set_spanCompensationTable()
weighscale.set_spanCompensationTable()

Enregistre la table de compensation en température de l'amplitude du poids, afin de pouvoir corriger automatiquement le poids mesuré sur la base de la température de compensation.

function **set_spanCompensationTable**(**tempValues**, **compValues**)

La correction de poids sera faite par interpolation linéaire entre les points spécifiés.

Paramètres :

tempValues tableau de nombres flottants, correspondant aux différentes températures pour lesquelles une correction d'amplitude est spécifiée.

compValues tableau de nombres flottants, correspondant à la correction d'amplitude (en pour cents) à appliquer pour chacun des points de température, index par index.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

weighscale→set_userData() YWeighScale
weighscale→setUserData()
weighscale.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

weighscale→set_zeroTracking() YWeighScale
weighscale→setZeroTracking()
weighscale.set_zeroTracking()
weighscale.set_zeroTracking()

Modifie le taux d'adaptation de la température de compensation, en pour cents.

function **set_zeroTracking**(**newval**)

Paramètres :

newval une valeur numérique représentant le taux d'adaptation de la température de compensation, 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.

weighscale→setupSpan()weighscale.setupSpan() YWeighScale weighscale.setupSpan()

Configure le facteur de poids et l'amplitude maximale de la cellule de poids (stockés dans le genericSensor correspondant) de sorte à ce que le signal actuel corresponde au poids de référence spécifié.

function **setupSpan**(**currWeight**, **maxWeight**)

Paramètres :

currWeight poids de référence actuellement sur la cellule de charge.

maxWeight poids maximum qui sera utilisé sur la cellule de charge.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

weighscale→startDataLogger() YWeighScale
weighscale.startDataLogger()
weighscale.startDataLogger()

Démarre l'enregistreur de données du module.

function **startDataLogger**( )

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

weighscale→stopDataLogger() YWeighScale
weighscale.stopDataLogger()
weighscale.stopDataLogger()

Arrête l'enregistreur de données du module.

function **stopDataLogger**( )

Retourne :
YAPI_SUCCESS si l'opération se déroule sans erreur.

weighscale→tare()weighscale.tare()weighscale.tare() YWeighScale

Adapte le biais de mesure de la cellule de poids (stocké dans le genericSensor correspondant) de sorte à ce que le signal actuel corresponde à un poids nul.

function **tare**( )

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

weighscale→unmuteValueCallbacks() YWeighScale
weighscale.unmuteValueCallbacks()
weighscale.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

weighscale→wait_async()weighscale.wait_async() YWeighScale

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.

3.74. 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>
cpp	#include "yocto_wireless.h"
m	#import "yocto_wireless.h"
pas	uses yocto_wireless;
vb	yocto_wireless.vb
cs	yocto_wireless.cs
java	import com.yoctopuce.YoctoAPI.YWireless;
uwp	import com.yoctopuce.YoctoAPI.YWireless;
py	from yocto_wireless import *
php	require_once('yocto_wireless.php');
es	in HTML: <script src="../../lib/yocto_wireless.js"></script> in node.js: require('yoctolib-es2017/yocto_wireless.js');

Fonction globales

yFindWireless(**func**)

Permet de retrouver une interface réseau sans fil d'après un identifiant donné.

yFindWirelessInContext(**yctx**, **func**)

Permet de retrouver une interface réseau sans fil d'après un identifiant donné dans un Context YAPI.

yFirstWireless()

Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.

yFirstWirelessInContext(**yctx**)

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→clearCache()

Invalide le cache.

wireless→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau sans fil au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

wireless→get_advertisedValue()

Retourne la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).

wireless→get_channel()

Retourne le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.

wireless→get_detectedWlans()

Retourne une liste d'objets objet YFileRecord qui décrivent les réseaux sans fils détectés.

wireless→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

wireless→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

wireless→get_friendlyName()

Retourne un identifiant global de l'interface réseau sans fil au format `NOM_MODULE.NOM_FONCTION`.

wireless→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

wireless→get_functionId()

Retourne l'identifiant matériel de l'interface réseau sans fil, sans référence au module.

wireless→get_hardwareId()

Retourne l'identifiant matériel unique de l'interface réseau sans fil au format `SERIAL.FUNCTIONID`.

wireless→get_linkQuality()

Retourne la qualité de la connection, exprimée en pourcents.

wireless→get_logicalName()

Retourne le nom logique de l'interface réseau sans fil.

wireless→get_message()

Retourne le dernier message de diagnostique de l'interface au réseau sans fil.

wireless→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

wireless→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

wireless→get_security()

Retourne l'algorithme de sécurité utilisé par le réseau sans fil sélectionné.

wireless→get_ssid()

Retourne le nom (SSID) du réseau sans fil sélectionné.

wireless→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

wireless→get_wlanState()

Retourne l'état actuel de l'interface réseau sans fil.

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→loadAttribute(**attrName**)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

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→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

wireless→nextWireless()

Continue l'énumération des interfaces réseau sans fil commencée à l'aide de `yFirstWireless()`.

wireless→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

wireless→set_logicalName(newval)

Modifie le nom logique de l'interface réseau sans fil.

wireless→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

wireless→softAPNetwork(ssid, securityKey)

Modifie la configuration de l'interface réseau sans fil pour créer un pseudo point d'accès sans fil ("Soft AP").

wireless→startWlanScan()

Déclenche un balayage des fréquences utilisable et construit la liste de réseau sans fils disponible.

wireless→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

wireless→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YWireless.FindWireless() YWireless yFindWireless()YWireless.FindWireless() YWireless.FindWireless()

Permet de retrouver une interface réseau sans fil d'après un identifiant donné.

```
function FindWireless( 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 ou 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. Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

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.FindWirelessInContext() YWireless
yFindWirelessInContext()
YWireless.FindWirelessInContext()
YWireless.FindWirelessInContext()

Permet de retrouver une interface réseau sans fil d'après un identifiant donné dans un Context YAPI.

function **FindWirelessInContext**(**yctx**, **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 ou 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 :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'interface réseau sans fil sans ambiguïté

Retourne :

un objet de classe `YWireless` qui permet ensuite de contrôler l'interface réseau sans fil.

YWireless.FirstWireless() YWireless yFirstWireless()YWireless.FirstWireless() YWireless.FirstWireless()

Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.

function **FirstWireless**( )

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.

YWireless.FirstWirelessInContext() YWireless
yFirstWirelessInContext()
YWireless.FirstWirelessInContext()
YWireless.FirstWirelessInContext()

Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.

function **FirstWirelessInContext**(**yctx**)

Utiliser la fonction `YWireless.nextWireless()` pour itérer sur les autres interfaces réseau sans fil.

Paramètres :

yctx un contexte YAPI.

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 wireless.adhocNetwork()

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**)

Sur le YoctoHub-Wireless-g, il est recommandé d'utiliser de préférence la fonction softAPNetwork() qui crée un pseudo point d'accès, plus efficace et mieux supporté qu'un réseau ad-hoc.

Si une clef d'accès est configurée pour un réseau ad-hoc, le réseau sera protégé par une sécurité WEP40 (5 caractères ou 10 chiffres hexadécimaux) ou WEP128 (13 caractères ou 26 chiffres hexadécimaux). Pour réduire les risques d'intrusion, il est recommandé d'utiliser une clé WEP128 basée sur 26 chiffres hexadécimaux provenant d'une bonne source aléatoire.

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

ssid	nom du réseau sans fil à créer
securityKey	clé d'accès de réseau, sous forme de chaîne de caractères

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wireless→clearCache()wireless.clearCache() YWireless

Invalide le cache.

function **clearCache**( )

Invalide le cache des valeurs courantes de l'interface réseau sans fil. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

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 utilsé 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 debuggeur.

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()
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()
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()
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 `startWlanScan()`. L'appelant est responsable de la désallocation de la liste retournée dans les langages ne disposant pas de "garbage collection".

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'identifant 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() YWireless
wireless→functionDescriptor()
wireless.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

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 example `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 example 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()
wireless.get_linkQuality()

Retourne la qualité de la connection, exprimée en pourcents.

function **get_linkQuality**( )

Retourne :
un entier représentant la qualité de la connection, exprimée en pourcents

En cas d'erreur, déclenche une exception ou retourne `Y_LINKQUALITY_INVALID`.

wireless→get_logicalName() YWireless
wireless→logicalName()wireless.get_logicalName()
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()
wireless.get_message()

Retourne le dernier message de diagnostique de l'interface au réseau sans fil.

function **get_message**( )

Retourne :
une chaîne de caractères représentant le dernier message de diagnostique de l'interface au réseau sans fil

En cas d'erreur, déclenche une exception ou retourne `Y_MESSAGE_INVALID`.

wireless→get_module() YWireless
wireless→module()wireless.get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

function **get_module**( )

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :
une instance de YModule

wireless→get_security() YWireless
wireless→security()wireless.get_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()
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→get_wlanState() YWireless
wireless→wlanState()wireless.get_wlanState()
wireless.get_wlanState()

Retourne l'état actuel de l'interface réseau sans fil.

function **get_wlanState**( )

L'état Y_WLANSTATE_DOWN indique que l'interface n'est connectée à aucun réseau. L'état Y_WLANSTATE_SCANNING signifie que la carte réseau effectue un balayage des fréquences utilisables. Dans cet état le module n'est pas joignable et la configuration réseau n'est pas encore appliquée. L'état Y_WLANSTATE_CONNECTED signifie que la configuration réseau a pu être appliqué et que le module est joignable. Si le l'interface est configurée pour fonctionner en mode ad-hoc ou Soft AP, cela signifie que le réseau est fonctionnel et que un périphérique peut se connecter au réseau. L'état Y_WLANSTATE_REJECTED signifie que l'interface réseau n'a pas pu joindre le réseau souhaité. la source de l'erreur peut être obtenue à l'aide de la méthode get_message().

Retourne :

une valeur parmi Y_WLANSTATE_DOWN, Y_WLANSTATE_SCANNING, Y_WLANSTATE_CONNECTED et Y_WLANSTATE_REJECTED représentant l'état actuel de l'interface réseau sans fil

En cas d'erreur, déclenche une exception ou retourne Y_WLANSTATE_INVALID.

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→joinNetwork()wireless.joinNetwork() YWireless wireless.joinNetwork()

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 occasionellement 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→loadAttribute()wireless.loadAttribute() YWireless wireless.loadAttribute()

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
function loadAttribute( attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

wireless→muteValueCallbacks() YWireless
wireless.muteValueCallbacks()
wireless.muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **muteValueCallbacks**( )

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wireless→nextWireless()wireless.nextWireless() YWireless wireless.nextWireless()

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() YWireless
wireless.registerValueCallback()
wireless.registerValueCallback()

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()
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→softAPNetwork()wireless.softAPNetwork() YWireless wireless.softAPNetwork()

Modifie la configuration de l'interface réseau sans fil pour créer un pseudo point d'accès sans fil ("Soft AP").

function **softAPNetwork**(**ssid**, **securityKey**)

Cette fonction ne fonctionne que sur le YoctoHub-Wireless-g.

Si une clef d'accès est configurée pour un réseau SoftAP, le réseau sera protégé par une sécurité WEP40 (5 caractères ou 10 chiffres hexadécimaux) ou WEP128 (13 caractères ou 26 chiffres hexadécimaux). Pour réduire les risques d'intrusion, il est recommandé d'utiliser une clé WEP128 basée sur 26 chiffres hexadécimaux provenant d'une bonne source aléatoire.

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

ssid	nom du réseau sans fil à créer
securityKey	clé d'accès de réseau, sous forme de chaîne de caractères

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wireless→startWlanScan()wireless.startWlanScan() YWireless
wireless.startWlanScan()

Déclenche un balayage des fréquences utilisable et construit la liste de réseau sans fils disponible.

function **startWlanScan**( )

Pendant le balayage des réseau sans fils, l'interface se déconnecte du réseau sans fils actuel. A la fin du balayage l'interface essaye de se reconnecter au réseau sans fil. Pendant le balayage le `wlanState` passe par l'état `Y_WLANSTATE_DOWN` puis par l'état `Y_WLANSTATE_SCANNING`. La liste des réseaux sans fils disponible peut être récupéré avec la méthode `get_detectedWlans()` à partir du moment ou `get_wlanState()` retourne `Y_WLANSTATE_REJECTED` ou `Y_WLANSTATE_CONNECTED`.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wireless→unmuteValueCallbacks() YWireless
wireless.unmuteValueCallbacks()
wireless.unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

function **unmuteValueCallbacks**( )

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

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→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

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z