Référence de l'API en ligne de commande

Table des matières

1. Introduction

Ce manuel est votre référence pour l'utilisation de la librairie en ligne de commande de Yoctopuce pour interfacer vos senseurs et contrôleurs USB.

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

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

2. Utilisation du Yocto-Demo en ligne de commande

Lorsque vous désirez effectuer une opération ponctuelle sur votre Yocto-Demo, comme la lecture d'une valeur, le changement d'un nom logique, etc.. vous pouvez bien sur utiliser le Virtual Hub, mais il existe une méthode encore plus simple, rapide et efficace: l'API en ligne de commande.

L'API en ligne de commande se présente sous la forme d'un ensemble d'exécutables, un par type de fonctionnalité offerte par l'ensemble des produits Yoctopuce. Ces exécutables sont fournis pré-compilés pour toutes les plateformes/OS officiellement supportés par Yoctopuce. Bien entendu, les sources de ces exécutables sont aussi fournies[1].

2.1. Installation

Téléchargez l'API en ligne de commande[2]. Il n'y a pas de programme d'installation à lancer, copiez simplement les exécutables correspondant à votre plateforme/OS dans le répertoire de votre choix. Ajoutez éventuellement ce répertoire à votre variable environnement PATH pour avoir accès aux exécutables depuis n'importe où. C'est tout, il ne vous reste plus qu'à brancher votre Yocto-Demo, ouvrir un shell et commencer à travailler en tapant par exemple:

```
C:\>YLed any set_power ON

C:\>YLed any set_blinking RELAX
```

Sous Linux, pour utiliser l'API en ligne de commande, vous devez soit être root, soit définir une règle *udev* pour votre système. Vous trouverez plus de détails au chapitre *Problèmes courants*.

2.2. Utilisation: description générale

Tous les exécutables de la l'API en ligne de commande fonctionnent sur le même principe: ils doivent être appelés de la manière suivante:

```
C:\>Executable [options] [cible] commande [paramètres]
```

Les `[options]` gèrent le fonctionnement global des commandes , elles permettent par exemple de piloter des modules à distance à travers le réseau, ou encore elles peuvent forcer les modules à sauver leur configuration après l'exécution de la commande.

[1] Si vous souhaitez recompiler l'API en ligne de commande, vous aurez aussi besoin de l'API C++
[2] http://www.yoctopuce.com/FR/libraries.php

La `[cible]` est le nom du module ou de la fonction auquel la commande va s'appliquer. Certaines commandes très génériques n'ont pas besoin de cible. Vous pouvez aussi utiliser les alias "*any*" ou "*all*", ou encore une liste de noms, séparés par des virgules, sans espace.

La `commande` est la commande que l'on souhaite exécuter. La quasi-totalité des fonctions disponibles dans les API de programmation classiques sont disponibles sous forme de commandes. Vous n'êtes pas obligé des respecter les minuscules/majuscules et les caractères soulignés dans le nom de la commande.

Les `[paramètres]` sont, assez logiquement, les paramètres dont la commande a besoin.

A tout moment les exécutables de l'API en ligne de commande sont capables de fournir une aide assez détaillée: Utilisez par exemple

```
C:\>executable /help
```

pour connaître la liste de commandes disponibles pour un exécutable particulier de l'API en ligne de commande, ou encore:

```
C:\>executable commande /help
```

Pour obtenir une description détaillée des paramètres d'une commande.

2.3. Contrôle de la fonction Led

Pour contrôler la fonction Led de votre Yocto-Demo, vous avez besoin de l'exécutable YLed.

Vous pouvez par exemple lancer:

```
C:\>YLed any set_power ON

C:\>YLed any set_blinking RELAX
```

Cet exemple utilise la cible "*any*" pour signifier que l'on désire travailler sur la première fonction Led trouvée parmi toutes celles disponibles sur les modules Yoctopuce accessibles au moment de l'exécution. Cela vous évite d'avoir à connaître le nom exact de votre fonction et celui de votre module.

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

```
C:\>YLed YCTOPOC1-123456.led describe

C:\>YLed YCTOPOC1-123456.MaFonction describe

C:\>YLed MonModule.led describe

C:\>YLed MonModule.MaFonction describe

C:\>YLed MaFonction describe
```

Pour travailler sur toutes les fonctions Led à la fois, utilisez la cible "*all*".

```
C:\>YLed all describe
```

Pour plus de détails sur les possibilités de l'exécutable`YLed`, utilisez:

```
C:\>YLed /help
```

2.4. Contrôle de la partie module

Chaque module peut être contrôlé d'une manière similaire à l'aide de l'exécutable `YModule`. Par exemple, pour obtenir la liste de tous les modules connectés, utilisez:

```
C:\>YModule inventory
```

Vous pouvez aussi utiliser la commande suivante pour obtenir une liste encore plus détaillée des modules connectés:

```
C:\>YModule all describe
```

Chaque propriété `xxx` du module peut être obtenue grâce à une commande du type `get_xxxx()`, et les propriétés qui ne sont pas en lecture seule peuvent être modifiées à l'aide de la commande `set_xxx()`. Par exemple:

```
C:\>YModule YCTOPOC1-12346 set_logicalName MonPremierModule

C:\>YModule YCTOPOC1-12346 get_logicalName
```

Modifications des réglages du module

Lorsque que vous souhaitez modifier les réglages d'un module, il suffit d'utiliser la commande `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 commande `saveToFlash`. Inversement il est possible de forcer le module à oublier ses réglages courants en utilisant la méthode `revertFromFlash`. Par exemple:

```
C:\>YModule YCTOPOC1-12346 set_logicalName MonPremierModule
C:\>YModule YCTOPOC1-12346 saveToFlash
```

Notez que vous pouvez faire la même chose en seule fois à l'aide de l'option `-s`

```
C:\>YModule -s  YCTOPOC1-12346 set_logicalName MonPremierModule
```

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

2.5. Limitations

L'API en ligne de commande est sujette à la même limitation que les autres API: il ne peut y avoir q'une seule application à la fois qui accède aux modules de manière native. Par défaut l'API en ligne de commande fonctionne en natif.

Cette limitation peut aisément être contournée en utilisant un Virtual Hub: il suffit de faire tourner le VirtualHub[3] sur la machine concernée et d'utiliser les executables de l'API en ligne de commande avec l'option `-r` par exemple, si vous utilisez:

```
C:\>YModule  inventory
```

[3] http://www.yoctopuce.com/FR/virtualhub.php

Vous obtenez un inventaire des modules connectés par USB, en utilisant un accès natif. Si il y a déjà une autre commande en cours qui accède aux modules en natif, cela ne fonctionnera pas. Mais si vous lancez un virtual hub et que vous lancez votre commande sous la forme:

```
C:\>YModule -r 127.0.0.1 inventory
```

cela marchera parce que la commande ne sera plus exécutée nativement, mais à travers le Virtual Hub. Notez que le Virtual Hub compte comme une application native.

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

Fonction globales
yCheckLogicalName(**name**) Vérifie si un nom donné est valide comme nom logique pour un module ou une fonction.
yDisableExceptions() Désactive l'utilisation d'exceptions pour la gestion des erreurs.
yEnableExceptions() Réactive l'utilisation d'exceptions pour la gestion des erreurs.
yEnableUSBHost(**osContext**) Cette fonction est utilisée uniquement sous Android.
yFreeAPI() Libère la mémoire dynamique utilisée par la librairie Yoctopuce.
yGetAPIVersion() Retourne la version de la librairie Yoctopuce utilisée.
yGetTickCount() Retourne la valeur du compteur monotone de temps (en millisecondes).
yHandleEvents(**errmsg**) Maintient la communication de la librairie avec les modules Yoctopuce.
yInitAPI(**mode**, **errmsg**) Initialise la librairie de programmation de Yoctopuce explicitement.
yPreregisterHub(**url**, **errmsg**) Alternative plus 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`.

ySetTimeout(**callback**, **ms_timeout**, **arguments**)

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

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

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

yTriggerHubDiscovery(**errmsg**)

Relance une détection des hubs réseau.

yUnregisterHub(**url**)

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

yUpdateDeviceList(**errmsg**)

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

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

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

3.2. Interface de la fonction Accelerometer

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
yFindAccelerometer(**func**) Permet de retrouver un accéléromètre d'après un identifiant donné.
yFirstAccelerometer() Commence l'énumération des accéléromètres accessibles par la librairie.
Méthodes des objets `YAccelerometer`
accelerometer→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
accelerometer→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de l'accéléromètre au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
accelerometer→get_advertisedValue() Retourne la valeur courante de l'accéléromètre (pas plus de 6 caractères).
accelerometer→get_currentRawValue() Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en g, sous forme de nombre à virgule.
accelerometer→get_currentValue() Retourne la valeur actuelle de l'accélération, en g, sous forme de nombre à virgule.
accelerometer→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'accéléromètre.
accelerometer→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'accéléromètre.
accelerometer→get_friendlyName() Retourne un identifiant global de l'accéléromètre au format `NOM_MODULE.NOM_FONCTION`.
accelerometer→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
accelerometer→get_functionId() Retourne l'identifiant matériel de l'accéléromètre, sans référence au module.
accelerometer→get_hardwareId()

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

accelerometer→get_highestValue()

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

accelerometer→get_logFrequency()

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

accelerometer→get_logicalName()

Retourne le nom logique de l'accéléromètre.

accelerometer→get_lowestValue()

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

accelerometer→get_module()

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

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

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

accelerometer→get_recordedData(**startTime**, **endTime**)

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

accelerometer→get_reportFrequency()

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

accelerometer→get_resolution()

Retourne la résolution des valeurs mesurées.

accelerometer→get_unit()

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

accelerometer→get_userData()

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

accelerometer→get_xValue()

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

accelerometer→get_yValue()

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

accelerometer→get_zValue()

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

accelerometer→isOnline()

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

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

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

accelerometer→load(**msValidity**)

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

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

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

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

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

accelerometer→nextAccelerometer()

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

accelerometer→registerTimedReportCallback(**callback**)

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

accelerometer→registerValueCallback(callback)

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

accelerometer→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

accelerometer→set_logFrequency(newval)

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

accelerometer→set_logicalName(newval)

Modifie le nom logique de l'accéléromètre.

accelerometer→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

accelerometer→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

accelerometer→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

accelerometer→set_userData(data)

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

accelerometer→wait_async(callback, context)

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

accelerometer→calibrateFromPoints() YAccelerometer
YAccelerometer calibrateFromPoints

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

YAccelerometer **target calibrateFromPoints rawValues refValues**

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

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

Paramètres :

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

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

Retourne :

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

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

accelerometer→get_advertisedValue() YAccelerometer
accelerometer→advertisedValue()YAccelerometer
get_advertisedValue

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

YAccelerometer **target** **get_advertisedValue**

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

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

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

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

YAccelerometer **target** **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()YAccelerometer
get_currentValue

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

YAccelerometer **target** **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_highestValue() YAccelerometer
accelerometer→highestValue()YAccelerometer
get_highestValue

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

YAccelerometer **target** **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()YAccelerometer
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.

YAccelerometer **target** **get_logFrequency**

Retourne :

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

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

accelerometer→get_logicalName() YAccelerometer
accelerometer→logicalName()YAccelerometer
get_logicalName

Retourne le nom logique de l'accéléromètre.

YAccelerometer **target** **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()YAccelerometer
get_lowestValue

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

YAccelerometer **target** **get_lowestValue**

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

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

accelerometer→get_recordedData() YAccelerometer
accelerometer→recordedData()YAccelerometer
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.

```
YAccelerometer target get_recordedData startTime endTime
```

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

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

Paramètres :

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

endTime la 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()YAccelerometer
get_reportFrequency

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

YAccelerometer **target** **get_reportFrequency**

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

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

accelerometer→get_resolution() YAccelerometer
accelerometer→resolution()YAccelerometer
get_resolution

Retourne la résolution des valeurs mesurées.

YAccelerometer **target get_resolution**

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

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

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

accelerometer→get_unit() YAccelerometer
accelerometer→unit()YAccelerometer get_unit

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

YAccelerometer **target** **get_unit**

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

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

accelerometer→get_xValue() YAccelerometer
accelerometer→xValue()YAccelerometer get_xValue

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

```
YAccelerometer target get_xValue
```

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

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

accelerometer→get_yValue() YAccelerometer
accelerometer→yValue()YAccelerometer get_yValue

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

```
YAccelerometer target get_yValue
```

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

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

accelerometer→get_zValue() YAccelerometer
accelerometer→zValue()YAccelerometer get_zValue

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

YAccelerometer **target get_zValue**

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

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

accelerometer→loadCalibrationPoints() YAccelerometer
YAccelerometer loadCalibrationPoints

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

```
YAccelerometer target loadCalibrationPoints rawValues refValues
```

Paramètres :

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

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

Retourne :

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

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

accelerometer→set_highestValue() YAccelerometer accelerometer→setHighestValue()YAccelerometer set_highestValue

Modifie la mémoire de valeur maximale observée.

```
YAccelerometer target set_highestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

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

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

accelerometer→set_logFrequency() YAccelerometer accelerometer→setLogFrequency()YAccelerometer set_logFrequency

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

YAccelerometer **target** **set_logFrequency** **newval**

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

Paramètres :

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

Retourne :

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

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

accelerometer→set_logicalName() YAccelerometer
accelerometer→setLogicalName()YAccelerometer
set_logicalName

Modifie le nom logique de l'accéléromètre.

```
YAccelerometer target set_logicalName newval
```

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

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'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()YAccelerometer
set_lowestValue

Modifie la mémoire de valeur minimale observée.

YAccelerometer **target set_lowestValue newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

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

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

accelerometer→set_reportFrequency() YAccelerometer
accelerometer→setReportFrequency()
YAccelerometer set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

```
YAccelerometer target set_reportFrequency newval
```

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

Paramètres :

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

Retourne :

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

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

accelerometer→set_resolution() YAccelerometer
accelerometer→setResolution()YAccelerometer
set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YAccelerometer target set_resolution newval
```

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

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

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

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

3.3. Interface de la fonction Altitude

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
yFindAltitude(**func**) Permet de retrouver un altimetre d'après un identifiant donné.
yFirstAltitude() Commence l'énumération des altimètres accessibles par la librairie.
Méthodes des objets `YAltitude`
altitude→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
altitude→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de l'altimètre au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
altitude→get_advertisedValue() Retourne la valeur courante de l'altimètre (pas plus de 6 caractères).
altitude→get_currentRawValue() Retourne la valeur brute 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_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_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→load(**msValidity**)

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

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→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→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.

altitude→calibrateFromPoints()YAltitude YAltitude calibrateFromPoints

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

YAltitude **target calibrateFromPoints rawValues refValues**

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

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

Paramètres :

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

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

Retourne :

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

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

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

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

YAltitude **target** **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()YAltitude
get_currentRawValue

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

YAltitude **target 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()YAltitude get_currentValue

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

YAltitude **target** **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_highestValue() YAltitude
altitude→highestValue()YAltitude get_highestValue

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

YAltitude **target 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()YAltitude 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.

YAltitude **target** **get_logFrequency**

Retourne :

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

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

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

Retourne le nom logique de l'altimètre.

YAltitude **target 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()YAltitude get_lowestValue

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

YAltitude **target** **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_qnh() YAltitude
altitude→qnh()YAltitude get_qnh

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

YAltitude **target 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()YAltitude 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.

```
YAltitude target get_recordedData startTime endTime
```

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

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

Paramètres :

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

endTime la 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()YAltitude
get_reportFrequency

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

YAltitude **target get_reportFrequency**

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

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

altitude→get_resolution() YAltitude altitude→resolution()YAltitude get_resolution

Retourne la résolution des valeurs mesurées.

YAltitude **target** **get_resolution**

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

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

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

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

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

YAltitude **target 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→loadCalibrationPoints()YAltitude YAltitude loadCalibrationPoints

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

YAltitude **target loadCalibrationPoints rawValues refValues**

Paramètres :

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

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

Retourne :

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

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

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

Modifie l'altitude actuelle supposée.

```
YAltitude target 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()YAltitude
set_highestValue

Modifie la mémoire de valeur maximale observée.

YAltitude **target** **set_highestValue** **newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

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

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

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

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

YAltitude **target set_logFrequency newval**

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

Paramètres :

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

Retourne :

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

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

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

Modifie le nom logique de l'altimètre.

```
YAltitude target set_logicalName newval
```

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

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'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()YAltitude set_lowestValue

Modifie la mémoire de valeur minimale observée.

```
YAltitude target set_lowestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

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

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

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

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

YAltitude **target** **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()YAltitude
set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

YAltitude **target set_reportFrequency newval**

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

Paramètres :

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

Retourne :

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

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

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

Modifie la résolution des valeurs physique mesurées.

```
YAltitude target set_resolution newval
```

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

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

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

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

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

Fonction globales
yFindAnButton(**func**) Permet de retrouver une entrée analogique d'après un identifiant donné.
yFirstAnButton() Commence l'énumération des entrées analogiques accessibles par la librairie.
Méthodes des objets `YAnButton`
anbutton→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de l'entrée analogique au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
anbutton→get_advertisedValue() Retourne la valeur courante de l'entrée analogique (pas plus de 6 caractères).
anbutton→get_analogCalibration() Permet de savoir si une procédure de calibration est actuellement en cours.
anbutton→get_calibratedValue() Retourne la valeur calibrée de l'entrée (entre 0 et 1000 inclus).
anbutton→get_calibrationMax() Retourne la valeur maximale observée durant la calibration (entre 0 et 4095 inclus).
anbutton→get_calibrationMin() Retourne la valeur minimale observée durant la calibration (entre 0 et 4095 inclus).
anbutton→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'entrée analogique.
anbutton→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'entrée analogique.
anbutton→get_friendlyName() Retourne un identifiant global de l'entrée analogique au format `NOM_MODULE.NOM_FONCTION`.
anbutton→get_functionDescriptor()

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

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

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

anbutton→nextAnButton()

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

anbutton→registerValueCallback(**callback**)

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

anbutton→resetCounter()

réinitialise le compteur d'impulsions et son timer

anbutton→set_analogCalibration(**newval**)

Enclenche ou déclenche le procédure de calibration.

anbutton→set_calibrationMax(**newval**)

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→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.

anbutton→get_advertisedValue() YAnButton
anbutton→advertisedValue()YAnButton
get_advertisedValue

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

YAnButton **target 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()YAnButton
get_analogCalibration

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

YAnButton **target** **get_analogCalibration**

Retourne :

soit `Y_ANALOGCALIBRATION_OFF`, soit `Y_ANALOGCALIBRATION_ON`

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

anbutton→get_calibratedValue() YAnButton
anbutton→calibratedValue()YAnButton
get_calibratedValue

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

YAnButton **target 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()YAnButton
get_calibrationMax

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

YAnButton **target get_calibrationMax**

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

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

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

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

YAnButton **target 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_isPressed() YAnButton
anbutton→isPressed()YAnButton get_isPressed

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

YAnButton **target get_isPressed**

Retourne :

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

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

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

YAnButton **target get_lastTimePressed**

Retourne :

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

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

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

YAnButton **target get_lastTimeReleased**

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

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

anbutton→get_logicalName() YAnButton
anbutton→logicalName()YAnButton get_logicalName

Retourne le nom logique de l'entrée analogique.

YAnButton **target get_logicalName**

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

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

anbutton→get_rawValue() YAnButton
anbutton→rawValue()YAnButton get_rawValue

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

YAnButton **target 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()YAnButton get_sensitivity

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

YAnButton **target 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→set_analogCalibration() YAnButton
anbutton→setAnalogCalibration()YAnButton
set_analogCalibration

Enclenche ou déclenche le procédure de calibration.

YAnButton **target set_analogCalibration newval**

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

Paramètres :

newval soit `Y_ANALOGCALIBRATION_OFF`, soit `Y_ANALOGCALIBRATION_ON`

Retourne :

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

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

anbutton→set_calibrationMax() YAnButton anbutton→setCalibrationMax()YAnButton set_calibrationMax

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

YAnButton **target 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()YAnButton set_calibrationMin

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

YAnButton **target** **set_calibrationMin** **newval**

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

Paramètres :

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

Retourne :

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

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

anbutton→set_logicalName() YAnButton
anbutton→setLogicalName()YAnButton
set_logicalName

Modifie le nom logique de l'entrée analogique.

```
YAnButton target set_logicalName newval
```

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

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'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()YAnButton set_sensitivity

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

YAnButton **target** **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.

3.5. Interface de la fonction CarbonDioxide

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
yFindCarbonDioxide(**func**) Permet de retrouver un capteur de CO2 d'après un identifiant donné.
yFirstCarbonDioxide() Commence l'énumération des capteurs de CO2 accessibles par la librairie.
Méthodes des objets `YCarbonDioxide`
carbondioxide→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
carbondioxide→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de CO2 au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
carbondioxide→get_advertisedValue() Retourne la valeur courante du capteur de CO2 (pas plus de 6 caractères).
carbondioxide→get_currentRawValue() Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en ppm (val), sous forme de nombre à virgule.
carbondioxide→get_currentValue() Retourne la valeur actuelle du taux de CO2, en ppm (val), sous forme de nombre à virgule.
carbondioxide→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de CO2.
carbondioxide→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de CO2.
carbondioxide→get_friendlyName() Retourne un identifiant global du capteur de CO2 au format `NOM_MODULE.NOM_FONCTION`.
carbondioxide→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
carbondioxide→get_functionId() Retourne l'identifiant matériel du capteur de CO2, sans référence au module.
carbondioxide→get_hardwareId()

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

carbondioxide→get_highestValue()

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

carbondioxide→get_logFrequency()

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

carbondioxide→get_logicalName()

Retourne le nom logique du capteur de CO2.

carbondioxide→get_lowestValue()

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

carbondioxide→get_module()

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

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

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

carbondioxide→get_recordedData(**startTime**, **endTime**)

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

carbondioxide→get_reportFrequency()

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

carbondioxide→get_resolution()

Retourne la résolution des valeurs mesurées.

carbondioxide→get_unit()

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

carbondioxide→get_userData()

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

carbondioxide→isOnline()

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

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

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

carbondioxide→load(**msValidity**)

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

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

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

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

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

carbondioxide→nextCarbonDioxide()

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

carbondioxide→registerTimedReportCallback(**callback**)

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

carbondioxide→registerValueCallback(**callback**)

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

carbondioxide→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

carbondioxide→set_logFrequency(**newval**)

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

carbondioxide→set_logicalName(newval)

Modifie le nom logique du capteur de CO2.

carbondioxide→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

carbondioxide→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

carbondioxide→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

carbondioxide→set_userData(data)

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

carbondioxide→wait_async(callback, context)

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

carbondioxide→calibrateFromPoints() YCarbonDioxide
YCarbonDioxide calibrateFromPoints

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

YCarbonDioxide **target calibrateFromPoints rawValues refValues**

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

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

Paramètres :

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

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

Retourne :

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

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

carbondioxide→get_advertisedValue() YCarbonDioxide
carbondioxide→advertisedValue()YCarbonDioxide
get_advertisedValue

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

YCarbonDioxide **target** **get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante du capteur de CO2 (pas plus de 6 caractères).

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

carbondioxide→get_currentRawValue() YCarbonDioxide
carbondioxide→currentRawValue()YCarbonDioxide
get_currentRawValue

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

YCarbonDioxide **target 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()YCarbonDioxide
get_currentValue

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

YCarbonDioxide **target** **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_highestValue() YCarbonDioxide
carbondioxide→highestValue()YCarbonDioxide
get_highestValue

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

YCarbonDioxide **target 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()YCarbonDioxide
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.

YCarbonDioxide **target** **get_logFrequency**

Retourne :

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

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

carbondioxide→get_logicalName() YCarbonDioxide
carbondioxide→logicalName()YCarbonDioxide
get_logicalName

Retourne le nom logique du capteur de CO2.

YCarbonDioxide **target get_logicalName**

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

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

carbondioxide→get_lowestValue() YCarbonDioxide
carbondioxide→lowestValue()YCarbonDioxide
get_lowestValue

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

YCarbonDioxide **target** **get_lowestValue**

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

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

carbondioxide→get_recordedData() YCarbonDioxide
carbondioxide→recordedData()YCarbonDioxide
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.

YCarbonDioxide **target get_recordedData startTime endTime**

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

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

Paramètres :

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

endTime la 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()YCarbonDioxide
get_reportFrequency

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

YCarbonDioxide **target** **get_reportFrequency**

Retourne :

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

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

carbondioxide→get_resolution() YCarbonDioxide
carbondioxide→resolution()YCarbonDioxide
get_resolution

Retourne la résolution des valeurs mesurées.

YCarbonDioxide **target get_resolution**

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

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

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

carbondioxide→get_unit() YCarbonDioxide
carbondioxide→unit()YCarbonDioxide get_unit

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

YCarbonDioxide **target** **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→loadCalibrationPoints() YCarbonDioxide
YCarbonDioxide loadCalibrationPoints

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

YCarbonDioxide **target loadCalibrationPoints rawValues refValues**

Paramètres :

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

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

Retourne :

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

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

carbondioxide→set_highestValue() YCarbonDioxide
carbondioxide→setHighestValue()YCarbonDioxide
set_highestValue

Modifie la mémoire de valeur maximale observée.

YCarbonDioxide **target** **set_highestValue** **newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

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

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

carbondioxide→set_logFrequency() YCarbonDioxide carbondioxide→setLogFrequency()YCarbonDioxide set_logFrequency

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

YCarbonDioxide **target set_logFrequency newval**

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

Paramètres :

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

Retourne :

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

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

carbondioxide→set_logicalName() YCarbonDioxide
carbondioxide→setLogicalName()YCarbonDioxide
set_logicalName

Modifie le nom logique du capteur de CO2.

```
YCarbonDioxide target set_logicalName newval
```

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

Paramètres :

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

Retourne :

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

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

carbondioxide→set_lowestValue() YCarbonDioxide
carbondioxide→setLowestValue()YCarbonDioxide
set_lowestValue

Modifie la mémoire de valeur minimale observée.

YCarbonDioxide **target set_lowestValue newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

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

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

carbondioxide→set_reportFrequency() YCarbonDioxide
carbondioxide→setReportFrequency()
YCarbonDioxide set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

```
YCarbonDioxide target set_reportFrequency newval
```

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

Paramètres :

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

Retourne :

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

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

carbondioxide→set_resolution() YCarbonDioxide
carbondioxide→setResolution()YCarbonDioxide
set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YCarbonDioxide target set_resolution newval
```

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

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

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

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

3.6. Interface de la fonction ColorLed

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
yFindColorLed(**func**) Permet de retrouver une led RGB d'après un identifiant donné.
yFirstColorLed() Commence l'énumération des leds RGB accessibles par la librairie.
Méthodes des objets `YColorLed`
colorled→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de la led RGB au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
colorled→get_advertisedValue() Retourne la valeur courante de la led RGB (pas plus de 6 caractères).
colorled→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la led RGB.
colorled→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la led RGB.
colorled→get_friendlyName() Retourne un identifiant global de la led RGB au format `NOM_MODULE.NOM_FONCTION`.
colorled→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
colorled→get_functionId() Retourne l'identifiant matériel de la led RGB, sans référence au module.
colorled→get_hardwareId() Retourne l'identifiant matériel unique de la led RGB au format `SERIAL.FUNCTIONID`.
colorled→get_hslColor() Retourne la couleur HSL courante de la led.
colorled→get_logicalName() Retourne le nom logique de la led RGB.

colorled→get_module()

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

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

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

colorled→get_rgbColor()

Retourne la couleur RGB courante de la led.

colorled→get_rgbColorAtPowerOn()

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

colorled→get_userData()

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

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

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

colorled→isOnline()

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

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

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

colorled→load(**msValidity**)

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

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

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

colorled→nextColorLed()

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

colorled→registerValueCallback(**callback**)

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

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

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

colorled→set_hslColor(**newval**)

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

colorled→set_logicalName(**newval**)

Modifie le nom logique de la led RGB.

colorled→set_rgbColor(**newval**)

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

colorled→set_rgbColorAtPowerOn(**newval**)

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

colorled→set_userData(**data**)

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

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

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

colorled→get_advertisedValue() YColorLed
colorled→advertisedValue()YColorLed
get_advertisedValue

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

```
YColorLed target 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_hslColor() YColorLed
colorled→hslColor()YColorLed get_hslColor

Retourne la couleur HSL courante de la led.

```
YColorLed target get_hslColor
```

Retourne :
un entier représentant la couleur HSL courante de la led

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

colorled→get_logicalName() YColorLed
colorled→logicalName()YColorLed get_logicalName

Retourne le nom logique de la led RGB.

YColorLed **target 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_rgbColor() YColorLed
colorled→rgbColor()YColorLed get_rgbColor

Retourne la couleur RGB courante de la led.

YColorLed **target get_rgbColor**

Retourne :
un entier représentant la couleur RGB courante de la led

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

colorled→get_rgbColorAtPowerOn() YColorLed
colorled→rgbColorAtPowerOn()YColorLed
get_rgbColorAtPowerOn

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

```
YColorLed target get_rgbColorAtPowerOn
```

Retourne :
un entier représentant la couleur configurée pour être affichage à l'allumage du module

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

colorled→hslMove()YColorLed hslMove YColorLed

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

YColorLed **target 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→rgbMove()YColorLed rgbMove YColorLed

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

YColorLed **target rgbMove rgb_target ms_duration**

Paramètres :

rgb_target couleur RGB désirée à la fin de la transition

ms_duration durée de la transition, en millisecondes

Retourne :

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

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

colorled→set_hslColor() YColorLed
colorled→setHslColor()YColorLed set_hslColor

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

```
YColorLed target 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()YColorLed
set_logicalName

Modifie le nom logique de la led RGB.

```
YColorLed target set_logicalName newval
```

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

Paramètres :

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

Retourne :

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

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

colorled→set_rgbColor() YColorLed
colorled→setRgbColor()YColorLed set_rgbColor

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

```
YColorLed target 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()YColorLed set_rgbColorAtPowerOn

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

```
YColorLed target set_rgbColorAtPowerOn newval
```

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

Paramètres :

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

Retourne :

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

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

3.7. Interface de la fonction Compass

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

yFindCompass(**func**)

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

yFirstCompass()

Commence l'énumération des compas accessibles par la librairie.

Méthodes des objets `YCompass`

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

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

compass→describe()

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

compass→get_advertisedValue()

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

compass→get_currentRawValue()

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

compass→get_currentValue()

Retourne la valeur actuelle du cap relatif, en degrés, sous forme de nombre à virgule.

compass→get_errorMessage()

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

compass→get_errorType()

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

compass→get_friendlyName()

Retourne un identifiant global du compas au format `NOM_MODULE.NOM_FONCTION`.

compass→get_functionDescriptor()

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

compass→get_functionId()

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

compass→get_hardwareId()

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

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_unit()

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

compass→get_userData()

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

compass→isOnline()

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

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

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

compass→load(**msValidity**)

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

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

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

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

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

compass→nextCompass()

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

compass→registerTimedReportCallback(**callback**)

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

compass→registerValueCallback(**callback**)

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

compass→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

compass→set_logFrequency(**newval**)

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

compass→set_logicalName(**newval**)

Modifie le nom logique du compas.

compass→set_lowestValue(**newval**)

Modifie la mémoire de valeur minimale observée.

compass→set_reportFrequency(**newval**)

Modifie la fréquence de notification périodique des valeurs mesurées.

compass→set_resolution(**newval**)

Modifie la résolution des valeurs physique mesurées.

compass→set_userData(**data**)

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

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

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

compass→calibrateFromPoints()YCompass YCompass calibrateFromPoints

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

YCompass **target calibrateFromPoints rawValues refValues**

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

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

Paramètres :

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

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

Retourne :

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

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

compass→get_advertisedValue() YCompass
compass→advertisedValue()YCompass
get_advertisedValue

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

```
YCompass target get_advertisedValue
```

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

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

compass→get_currentRawValue() YCompass
compass→currentRawValue()YCompass
get_currentRawValue

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

YCompass **target** **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()YCompass
get_currentValue

Retourne la valeur actuelle du cap relatif, en degrés, sous forme de nombre à virgule.

```
YCompass target 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_highestValue() YCompass
compass→highestValue()YCompass
get_highestValue

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

YCompass **target** **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()YCompass
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.

YCompass **target** **get_logFrequency**

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

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

compass→get_logicalName() YCompass
compass→logicalName()YCompass get_logicalName

Retourne le nom logique du compas.

YCompass **target get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du compas.

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

compass→get_lowestValue() YCompass
compass→lowestValue()YCompass get_lowestValue

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

YCompass **target** **get_lowestValue**

Retourne :
une valeur numérique représentant la valeur minimale observée pour le cap relatif depuis le démarrage du module

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

compass→get_magneticHeading() YCompass
compass→magneticHeading()YCompass
get_magneticHeading

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

YCompass **target** **get_magneticHeading**

Retourne :
une valeur numérique représentant la direction du nord magnétique, indépendemment du cap configuré

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

compass→get_recordedData() YCompass
compass→recordedData()YCompass
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.

YCompass **target get_recordedData startTime endTime**

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

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

Paramètres :

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

endTime la 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()YCompass
get_reportFrequency

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

YCompass **target** **get_reportFrequency**

Retourne :

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

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

compass→get_resolution() YCompass
compass→resolution()YCompass get_resolution

Retourne la résolution des valeurs mesurées.

```
YCompass target get_resolution
```

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

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

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

compass→get_unit() YCompass
compass→unit()YCompass get_unit

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

YCompass **target get_unit**

Retourne :
une chaîne de caractères représentant l'unité dans laquelle le cap relatif est exprimée

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

compass→loadCalibrationPoints()YCompass loadCalibrationPoints YCompass

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

YCompass **target loadCalibrationPoints rawValues refValues**

Paramètres :

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

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

Retourne :

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

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

compass→set_highestValue() YCompass
compass→setHighestValue()YCompass
set_highestValue

Modifie la mémoire de valeur maximale observée.

YCompass **target** **set_highestValue** **newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

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

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

compass→set_logFrequency() YCompass
compass→setLogFrequency()YCompass
set_logFrequency

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

```
YCompass target set_logFrequency newval
```

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

Paramètres :

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

Retourne :

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

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

compass→set_logicalName() YCompass
compass→setLogicalName()YCompass
set_logicalName

Modifie le nom logique du compas.

```
YCompass target set_logicalName newval
```

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

Paramètres :

newval une chaîne de caractères représentant le nom logique du 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()YCompass
set_lowestValue

Modifie la mémoire de valeur minimale observée.

```
YCompass target set_lowestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

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

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

compass→set_reportFrequency() YCompass
compass→setReportFrequency()YCompass
set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

```
YCompass target set_reportFrequency newval
```

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

Paramètres :

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

Retourne :

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

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

compass→set_resolution() YCompass
compass→setResolution()YCompass set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YCompass target set_resolution newval
```

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

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

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

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

3.8. Interface de la fonction Current

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
yFindCurrent(**func**) Permet de retrouver un capteur de courant d'après un identifiant donné.
yFirstCurrent() Commence l'énumération des capteurs de courant accessibles par la librairie.
Méthodes des objets `YCurrent`
current→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
current→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de courant au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
current→get_advertisedValue() Retourne la valeur courante du capteur de courant (pas plus de 6 caractères).
current→get_currentRawValue() Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en mA, sous forme de nombre à virgule.
current→get_currentValue() Retourne la valeur actuelle du courant, en mA, sous forme de nombre à virgule.
current→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.
current→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.
current→get_friendlyName() Retourne un identifiant global du capteur de courant au format `NOM_MODULE.NOM_FONCTION`.
current→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
current→get_functionId() Retourne l'identifiant matériel du capteur de courant, sans référence au module.
current→get_hardwareId()

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_unit()

Retourne l'unité dans laquelle le courant est exprimée.

current→get_userData()

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

current→isOnline()

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

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

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

current→load(**msValidity**)

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

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

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

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

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

current→nextCurrent()

Continue l'énumération des capteurs de courant commencée à l'aide de `yFirstCurrent()`.

current→registerTimedReportCallback(**callback**)

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

current→registerValueCallback(**callback**)

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

current→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

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→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.

current→calibrateFromPoints()YCurrent YCurrent calibrateFromPoints

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

YCurrent **target calibrateFromPoints rawValues refValues**

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

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

Paramètres :

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

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

Retourne :

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

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

current→get_advertisedValue() YCurrent
current→advertisedValue()YCurrent
get_advertisedValue

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

YCurrent **target** **get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante du capteur de courant (pas plus de 6 caractères).

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

current→get_currentRawValue() YCurrent
current→currentRawValue()YCurrent
get_currentRawValue

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

```
YCurrent target 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()YCurrent get_currentValue

Retourne la valeur actuelle du courant, en mA, sous forme de nombre à virgule.

YCurrent **target** **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_highestValue() YCurrent
current→highestValue()YCurrent get_highestValue

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

```
YCurrent target 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()YCurrent 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.

YCurrent **target** **get_logFrequency**

Retourne :

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

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

current→get_logicalName() YCurrent
current→logicalName()YCurrent get_logicalName

Retourne le nom logique du capteur de courant.

```
YCurrent target get_logicalName
```

Retourne :
une chaîne de caractères représentant le nom logique du capteur de courant.

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

current→get_lowestValue() YCurrent
current→lowestValue()YCurrent get_lowestValue

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

YCurrent **target** **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_recordedData() YCurrent
current→recordedData()YCurrent 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.

```
YCurrent target get_recordedData startTime endTime
```

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

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

Paramètres :

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

endTime la 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()YCurrent
get_reportFrequency

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

```
YCurrent target get_reportFrequency
```

Retourne :

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

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

current→get_resolution() YCurrent
current→resolution()YCurrent get_resolution

Retourne la résolution des valeurs mesurées.

```
YCurrent target get_resolution
```

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

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

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

current→get_unit() YCurrent
current→unit()YCurrent get_unit

Retourne l'unité dans laquelle le courant est exprimée.

```
YCurrent target 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→loadCalibrationPoints()YCurrent loadCalibrationPoints — YCurrent

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

```
YCurrent target loadCalibrationPoints rawValues refValues
```

Paramètres :

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

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

Retourne :

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

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

current→set_highestValue() YCurrent
current→setHighestValue()YCurrent
set_highestValue

Modifie la mémoire de valeur maximale observée.

```
YCurrent target set_highestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

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

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

current→set_logFrequency() YCurrent
current→setLogFrequency()YCurrent
set_logFrequency

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

```
YCurrent target set_logFrequency newval
```

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

Paramètres :

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

Retourne :

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

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

current→set_logicalName() YCurrent
current→setLogicalName()YCurrent set_logicalName

Modifie le nom logique du capteur de courant.

```
YCurrent target set_logicalName newval
```

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

Paramètres :

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

Retourne :

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

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

current→set_lowestValue() YCurrent
current→setLowestValue()YCurrent set_lowestValue

Modifie la mémoire de valeur minimale observée.

```
YCurrent target set_lowestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

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

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

current→set_reportFrequency() YCurrent
current→setReportFrequency()YCurrent
set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

```
YCurrent target set_reportFrequency newval
```

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

Paramètres :

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

Retourne :

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

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

current→set_resolution() YCurrent
current→setResolution()YCurrent set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YCurrent target set_resolution newval
```

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

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

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

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

3.9. Interface de la fonction DataLogger

Les capteurs de Yoctopuce sont équipés d'une mémoire non-volatile permettant de mémoriser les données mesurées d'une manière autonome, sans nécessiter le suivi permanent d'un ordinateur. La fonction DataLogger contrôle les paramètres globaux de cet enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_datalogger.js'></script>
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 *
```

Fonction globales
yFindDataLogger(**func**) Permet de retrouver un enregistreur de données d'après un identifiant donné.
yFirstDataLogger() Commence l'énumération des enregistreurs de données accessibles par la librairie.
Méthodes des objets `YDataLogger`
datalogger→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de l'enregistreur de données au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
datalogger→forgetAllDataStreams() Efface tout l'historique des mesures de l'enregistreur de données.
datalogger→get_advertisedValue() Retourne la valeur courante de l'enregistreur de données (pas plus de 6 caractères).
datalogger→get_autoStart() Retourne le mode d'activation automatique de l'enregistreur de données à la mise sous tension.
datalogger→get_beaconDriven() Retourne 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→load_async(**msValidity**, **callback**, **context**)

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

datalogger→nextDataLogger()

Continue l'énumération des enregistreurs de données commencée à l'aide de `yFirstDataLogger()`.

datalogger→registerValueCallback(**callback**)

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

datalogger→set_autoStart(**newval**)

Modifie le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

datalogger→set_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→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.

datalogger→forgetAllDataStreams()YDataLogger forgetAllDataStreams YDataLogger

Efface tout l'historique des mesures de l'enregistreur de données.

YDataLogger **target** **forgetAllDataStreams**

Cette méthode remet aussi à zéro le compteur de Runs.

Retourne :

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

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

datalogger→get_advertisedValue() YDataLogger
datalogger→advertisedValue()YDataLogger
get_advertisedValue

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

YDataLogger **target get_advertisedValue**

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

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

datalogger→get_autoStart() YDataLogger
datalogger→autoStart()YDataLogger get_autoStart

Retourne le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

YDataLogger **target** **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()YDataLogger
get_beaconDriven

Retourne vrais si l'enregistreur de données est synchronisé avec la balise de localisation.

YDataLogger **target 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()YDataLogger
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.

YDataLogger **target** **get_currentRunIndex**

Retourne :
un entier représentant le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active

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

datalogger→get_dataSets() YDataLogger
datalogger→dataSets()YDataLogger get_dataSets

Retourne une liste d'objets YDataSet permettant de récupérer toutes les mesures stockées par l'enregistreur de données.

```
YDataLogger target get_dataSets
```

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

Retourne :
une liste d'objets YDataSet

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

datalogger→get_logicalName() YDataLogger
datalogger→logicalName()YDataLogger
get_logicalName

Retourne le nom logique de l'enregistreur de données.

YDataLogger **target** **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_recording() YDataLogger
datalogger→recording()YDataLogger get_recording

Retourne l'état d'activation de l'enregistreur de données.

YDataLogger **target get_recording**

Retourne :

soit `Y_RECORDING_OFF`, soit `Y_RECORDING_ON`, selon l'état d'activation de l'enregistreur de données

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

datalogger→get_timeUTC() YDataLogger
datalogger→timeUTC()YDataLogger get_timeUTC

Retourne le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue.

YDataLogger **target 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→set_autoStart() YDataLogger
datalogger→setAutoStart()YDataLogger
set_autoStart

Modifie le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

YDataLogger **target 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()YDataLogger
set_beaconDriven

Modifie le mode de synchronisation de l'enregistreur de données .

YDataLogger **target 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()YDataLogger
set_logicalName

Modifie le nom logique de l'enregistreur de données.

```
YDataLogger target set_logicalName newval
```

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

Paramètres :

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

Retourne :

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

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

datalogger→set_recording() YDataLogger
datalogger→setRecording()YDataLogger
set_recording

Modifie l'état d'activation de l'enregistreur de données.

YDataLogger **target set_recording newval**

Paramètres :

newval soit `Y_RECORDING_OFF`, soit `Y_RECORDING_ON`, selon l'état d'activation de l'enregistreur de données

Retourne :

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

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

datalogger→set_timeUTC() YDataLogger
datalogger→setTimeUTC()YDataLogger set_timeUTC

Modifie la référence de temps UTC, afin de l'attacher aux données enregistrées.

YDataLogger **target set_timeUTC newval**

Paramètres :

newval un entier représentant la référence de temps UTC, afin de l'attacher aux données enregistrées

Retourne :

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

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

3.10. Séquence de données mise en forme

Un Run est un intervalle de temps pendant lequel un module est sous tension. Les objets YDataRun fournissent un accès facilité à toutes les mesures collectées durant un Run donné, y compris en permettant la lecture par mesure distantes d'un intervalle spécifié.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Méthodes des objets `YDataRun`
datarun→get_averageValue(**measureName**, **pos**) Retourne la valeur moyenne des mesures observées au moment choisi.
datarun→get_duration() Retourne la durée (en secondes) du Run.
datarun→get_maxValue(**measureName**, **pos**) Retourne la valeur maximale des mesures observées au moment choisi.
datarun→get_measureNames() Retourne les noms des valeurs mesurées par l'enregistreur de données.
datarun→get_minValue(**measureName**, **pos**) Retourne la valeur minimale des mesures observées au moment choisi.
datarun→get_startTimeUTC() Retourne l'heure absolue du début du Run, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).
datarun→get_valueCount() Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.
datarun→get_valueInterval() Retourne l'intervalle de temps représenté par chaque valeur de ce run.
datarun→set_valueInterval(**valueInterval**) Change l'intervalle de temps représenté par chaque valeur de ce run.

datarun→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.

3.11. Séquence de données enregistrées

Les objets YDataSet permettent de récupérer un ensemble de mesures enregistrées correspondant à un capteur donné, pour une période choisie. Ils permettent le chargement progressif des données. Lorsque l'objet YDataSet est instancié par la fonction `get_recordedData()`, aucune donnée n'est encore chargée du module. Ce sont les appels successifs à la méthode `loadMore()` qui procèdent au chargement effectif des données depuis l'enregistreur de données.

Un résumé des mesures disponibles est disponible via la fonction `get_preview()` dès le premier appel à `loadMore()`. Les mesures elles-même sont disponibles via la fonction `get_measures()` au fur et à mesure de leur chargement.

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Méthodes des objets `YDataSet`
dataset→get_endTimeUTC()
Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).
dataset→get_functionId()
Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.
dataset→get_hardwareId()
Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format `SERIAL.FUNCTIONID`.
dataset→get_measures()
Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.
dataset→get_preview()
Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.
dataset→get_progress()
Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.
dataset→get_startTimeUTC()
Retourne l'heure absolue du début des mesures 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.

3.12. Séquence de données enregistrées brute

Les objets YDataStream correspondent aux séquences de mesures enregistrées brutes, directement telles qu'obtenues par l'enregistreur de données présent dans les senseurs de Yoctopuce.

Dans la plupart des cas, il n'est pas nécessaire d'utiliser les objets DataStream, car les objets YDataSet (retournés par la méthode `get_recordedData()` des senseurs et la méthode `get_dataSets()` du DataLogger) fournissent une interface plus pratique.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Méthodes des objets `YDataStream`
datastream→get_averageValue()
Retourne la moyenne des valeurs observées durant cette séquence.
datastream→get_columnCount()
Retourne le nombre de colonnes de données contenus dans la séquence.
datastream→get_columnNames()
Retourne le nom (la sémantique) des colonnes de données contenus dans la séquence.
datastream→get_data(**row**, **col**)
Retourne une mesure unique de la séquence, spécifiée par l'index de l'enregistrement (ligne) et de la mesure (colonne).
datastream→get_dataRows()
Retourne toutes les données mesurées 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).

3.13. Interface de la fonction DigitalIO

La librairie de programmation Yoctopuce permet simplement de changer l'état de chaque bit du port d'entrée sortie. Il est possible de changer tous les bits du port à la fois, ou de les changer indépendamment. La librairie permet aussi de créer des courtes impulsions de durée déterminée. Le comportement électrique de 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>
nodejs  var yoctolib = require('yoctolib');
        var YDigitalIO = yoctolib.YDigitalIO;
php     require_once('yocto_digitalio.php');
cpp     #include "yocto_digitalio.h"
m       #import "yocto_digitalio.h"
pas     uses yocto_digitalio;
vb      yocto_digitalio.vb
cs      yocto_digitalio.cs
java    import com.yoctopuce.YoctoAPI.YDigitalIO;
py      from yocto_digitalio import *
```

Fonction globales
yFindDigitalIO(**func**) Permet de retrouver un port d'E/S digital d'après un identifiant donné.
yFirstDigitalIO() Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.
Méthodes des objets `YDigitalIO`
digitalio→delayedPulse(**bitno**, **ms_delay**, **ms_duration**) Préprogramme une impulsion de durée spécifiée sur un bit choisi.
digitalio→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du port d'E/S digital au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
digitalio→get_advertisedValue() Retourne la valeur courante du port d'E/S digital (pas plus de 6 caractères).
digitalio→get_bitDirection(**bitno**) Retourne la direction d'un seul bit du port d'E/S.
digitalio→get_bitOpenDrain(**bitno**) Retourne la direction d'un seul bit du port d'E/S.
digitalio→get_bitPolarity(**bitno**) Retourne la polarité d'un seul bit du port d'E/S.
digitalio→get_bitState(**bitno**) Retourne l'état d'un seul bit du port d'E/S.
digitalio→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.
digitalio→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.
digitalio→get_friendlyName() Retourne un identifiant global du port d'E/S digital au format `NOM_MODULE.NOM_FONCTION`.
digitalio→get_functionDescriptor()

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

digitalio→get_functionId()
Retourne l'identifiant matériel du port d'E/S digital, sans référence au module.

digitalio→get_hardwareId()
Retourne l'identifiant matériel unique du port d'E/S digital au format `SERIAL.FUNCTIONID`.

digitalio→get_logicalName()
Retourne le nom logique du port d'E/S digital.

digitalio→get_module()
Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

digitalio→get_module_async(**callback**, **context**)
Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

digitalio→get_outputVoltage()
Retourne la source de tension utilisée pour piloter les bits en sortie.

digitalio→get_portDirection()
Retourne la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

digitalio→get_portOpenDrain()
Retourne le type d'interface électrique de chaque bit du port (bitmap).

digitalio→get_portPolarity()
Retourne la polarité des bits du port (bitmap).

digitalio→get_portSize()
Retourne le nombre de bits implémentés dans le port d'E/S.

digitalio→get_portState()
Retourne l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

digitalio→get_userData()
Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

digitalio→isOnline()
Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

digitalio→isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

digitalio→load(**msValidity**)
Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

digitalio→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

digitalio→nextDigitalIO()
Continue l'énumération des ports d'E/S digitaux commencée à l'aide de `yFirstDigitalIO()`.

digitalio→pulse(**bitno**, **ms_duration**)
Déclenche une impulsion de durée spécifiée sur un bit choisi.

digitalio→registerValueCallback(**callback**)
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

digitalio→set_bitDirection(**bitno**, **bitdirection**)
Change la direction d'un seul bit du port d'E/S.

digitalio→set_bitOpenDrain(**bitno**, **opendrain**)
Change le type d'interface électrique d'un seul bit du port d'E/S.

digitalio→set_bitPolarity(**bitno**, **bitpolarity**)
Change la polarité d'un seul bit du port d'E/S.

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→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.

digitalio→delayedPulse()YDigitalIO delayedPulse YDigitalIO

Préprogramme une impulsion de durée spécifiée sur un bit choisi.

YDigitalIO **target delayedPulse bitno ms_delay ms_duration**

Le bit va passer à 1 puis automatiquement revenir à 0 après le temps donné.

Paramètres :

bitno	index du bit dans le port; le bit de 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→get_advertisedValue() YDigitalIO
digitalio→advertisedValue()YDigitalIO
get_advertisedValue

Retourne la valeur courante du port d'E/S digital (pas plus de 6 caractères).

YDigitalIO **target get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante du port d'E/S digital (pas plus de 6 caractères).

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

digitalio→get_bitDirection() YDigitalIO
digitalio→bitDirection()YDigitalIO get_bitDirection

Retourne la direction d'un seul bit du port d'E/S.

```
YDigitalIO target 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()YDigitalIO get_bitOpenDrain

Retourne la direction d'un seul bit du port d'E/S.

YDigitalIO **target 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()YDigitalIO get_bitPolarity

Retourne la polarité d'un seul bit du port d'E/S.

```
YDigitalIO target 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()YDigitalIO get_bitState

Retourne l'état d'un seul bit du port d'E/S.

YDigitalIO **target 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_logicalName() YDigitalIO
digitalio→logicalName()YDigitalIO get_logicalName

Retourne le nom logique du port d'E/S digital.

YDigitalIO **target** **get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du port d'E/S digital.

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

digitalio→get_outputVoltage() YDigitalIO
digitalio→outputVoltage()YDigitalIO
get_outputVoltage

Retourne la source de tension utilisée pour piloter les bits en sortie.

YDigitalIO **target get_outputVoltage**

Retourne :

une valeur parmi `Y_OUTPUTVOLTAGE_USB_5V`, `Y_OUTPUTVOLTAGE_USB_3V` et `Y_OUTPUTVOLTAGE_EXT_V` représentant la source de tension utilisée pour piloter les bits en sortie

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

digitalio→get_portDirection() YDigitalIO
digitalio→portDirection()YDigitalIO get_portDirection

Retourne la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

YDigitalIO **target 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()YDigitalIO
get_portOpenDrain

Retourne le type d'interface électrique de chaque bit du port (bitmap).

YDigitalIO **target get_portOpenDrain**

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert).

Retourne :
un entier représentant le type d'interface électrique de chaque bit du port (bitmap)

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

digitalio→get_portPolarity() YDigitalIO
digitalio→portPolarity()YDigitalIO get_portPolarity

Retourne la polarité des bits du port (bitmap).

YDigitalIO **target** **get_portPolarity**

Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée.

Retourne :
un entier représentant la polarité des bits du port (bitmap)

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

digitalio→get_portSize() YDigitalIO
digitalio→portSize()YDigitalIO get_portSize

Retourne le nombre de bits implémentés dans le port d'E/S.

YDigitalIO **target 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()YDigitalIO get_portState

Retourne l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

YDigitalIO **target** **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→pulse()YDigitalIO pulse YDigitalIO

Déclenche une impulsion de durée spécifiée sur un bit choisi.

YDigitalIO **target pulse bitno ms_duration**

Le bit va passer à 1 puis automatiquement revenir à 0 après le temps donné.

Paramètres :

bitno	index du bit dans le port; le bit de 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→set_bitDirection() YDigitalIO
digitalio→setBitDirection()YDigitalIO set_bitDirection

Change la direction d'un seul bit du port d'E/S.

YDigitalIO **target 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()YDigitalIO
set_bitOpenDrain

Change le type d'interface électrique d'un seul bit du port d'E/S.

YDigitalIO **target 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()YDigitalIO set_bitPolarity

Change la polarité d'un seul bit du port d'E/S.

YDigitalIO **target 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()YDigitalIO set_bitState

Change l'état d'un seul bit du port d'E/S.

YDigitalIO **target 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()YDigitalIO
set_logicalName

Modifie le nom logique du port d'E/S digital.

YDigitalIO **target** **set_logicalName** **newval**

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

Paramètres :

newval une chaîne de caractères représentant le nom logique du port d'E/S digital.

Retourne :

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

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

digitalio→set_outputVoltage() YDigitalIO
digitalio→setOutputVoltage()YDigitalIO
set_outputVoltage

Modifie la source de tension utilisée pour piloter les bits en sortie.

YDigitalIO **target 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()YDigitalIO
set_portDirection

Modifie la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

YDigitalIO **target** **set_portDirection** **newval**

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

Paramètres :

newval un entier représentant la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie

Retourne :

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

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

digitalio→set_portOpenDrain() YDigitalIO
digitalio→setPortOpenDrain()YDigitalIO
set_portOpenDrain

Modifie le type d'interface électrique de chaque bit du port (bitmap).

YDigitalIO **target 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()YDigitalIO 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.

YDigitalIO **target** **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()YDigitalIO set_portState

Modifie l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

YDigitalIO **target 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→toggle_bitState()YDigitalIO toggle_bitState YDigitalIO

Inverse l'état d'un seul bit du port d'E/S.

YDigitalIO **target 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.

3.14. Interface de la fonction Display

L'interface de contrôle des écrans Yoctopuce est conçue pour afficher facilement des informations et des images. Le module est capable de gérer seul la superposition de plusieurs couches graphiques, qui peuvent être dessinées individuellement, sans affichage immédiat, puis librement positionnées sur l'écran. Il est aussi capable de rejouer des séquences de commandes pré-enregistrées (animations).

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
yFindDisplay(**func**) Permet de retrouver un ecran d'après un identifiant donné.
yFirstDisplay() Commence l'énumération des écran accessibles par la librairie.
Méthodes des objets `YDisplay`
display→copyLayerContent(**srcLayerId**, **dstLayerId**) Copie le 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→load_async(**msValidity**, **callback**, **context**)

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

display→newSequence()

Enclanche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

display→nextDisplay()

Continue l'énumération des écran commencée à l'aide de `yFirstDisplay()`.

display→pauseSequence(**delay_ms**)

Attend pour la durée spécifiée (en millisecondes) avant de jouer les commandes suivantes de la séquence active.

display→playSequence(sequenceName)

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes `newSequence()` et `saveSequence()`.

display→registerValueCallback(callback)

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

display→resetAll()

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

display→saveSequence(sequenceName)

Termine l'enregistrement d'une séquence et la sauvegarde sur la mémoire interne de l'écran, sous le nom choisi.

display→set_brightness(newval)

Modifie la 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→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.

display→copyLayerContent()YDisplay YDisplay copyLayerContent

Copie le contentu d'un couche d'affichage vers une autre couche.

YDisplay **target copyLayerContent srcLayerId dstLayerId**

La couleur et la transparence de tous les pixels de la couche de destination sont changés pour correspondre à la couche source. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

Paramètres :

srcLayerId l'identifiant de la couche d'origine (un chiffre parmi 0..layerCount-1)

dstLayerId l'identifiant de la couche de destination (un chiffre parmi 0..layerCount-1)

Retourne :

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

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

display→fade()YDisplay fade YDisplay

Change la luminosité de l'écran en douceur, pour produire un effet de fade-in ou fade-out.

YDisplay **target fade brightness duration**

Paramètres :

brightness nouvelle valeur de luminosité de l'écran

duration durée en millisecondes de la transition.

Retourne :

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

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

display→get_advertisedValue() YDisplay
display→advertisedValue()YDisplay
get_advertisedValue

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

YDisplay **target** **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()YDisplay get_brightness

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

YDisplay **target get_brightness**

Retourne :
un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

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

display→get_displayHeight() YDisplay
display→displayHeight()YDisplay get_displayHeight

Retourne la hauteur de l'écran, en pixels.

YDisplay **target** **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_displayType() YDisplay
display→displayType()YDisplay get_displayType

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

YDisplay **target get_displayType**

Retourne :

une valeur parmi `Y_DISPLAYTYPE_MONO`, `Y_DISPLAYTYPE_GRAY` et `Y_DISPLAYTYPE_RGB` représentant le type de l'écran: monochrome, niveaux de gris ou couleur

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

display→get_displayWidth() YDisplay
display→displayWidth()YDisplay get_displayWidth

Retourne la largeur de l'écran, en pixels.

YDisplay **target** **get_displayWidth**

Retourne :
un entier représentant la largeur de l'écran, en pixels

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

display→get_enabled() YDisplay
display→enabled()YDisplay get_enabled

Retourne vrai si le l'ecran est alimenté, faux sinon.

YDisplay **target 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_layerCount() YDisplay
display→layerCount()YDisplay get_layerCount

Retourne le nombre des couches affichables disponibles.

YDisplay **target** **get_layerCount**

Retourne :
un entier représentant le nombre des couches affichables disponibles

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

display→get_layerHeight() YDisplay
display→layerHeight()YDisplay get_layerHeight

Retourne la hauteur des couches affichables, en pixels.

YDisplay **target get_layerHeight**

Retourne :
un entier représentant la hauteur des couches affichables, en pixels

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

display→get_layerWidth() YDisplay
display→layerWidth()YDisplay get_layerWidth

Retourne la largeur des couches affichables, en pixels.

YDisplay **target** **get_layerWidth**

Retourne :
un entier représentant la largeur des couches affichables, en pixels

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

display→get_logicalName() YDisplay
display→logicalName()YDisplay get_logicalName

Retourne le nom logique de l'ecran.

YDisplay **target 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_orientation() YDisplay
display→orientation()YDisplay get_orientation

Retourne l'orientation sélectionnée pour l'écran.

YDisplay **target** **get_orientation**

Retourne :

une valeur parmi `Y_ORIENTATION_LEFT`, `Y_ORIENTATION_UP`, `Y_ORIENTATION_RIGHT` et `Y_ORIENTATION_DOWN` représentant l'orientation sélectionnée pour l'écran

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

display→get_startupSeq() YDisplay
display→startupSeq()YDisplay get_startupSeq

Retourne le nom de la séquence à jouer à la mise sous tension de l'écran.

YDisplay **target get_startupSeq**

Retourne :
une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

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

display→newSequence()YDisplay newSequence YDisplay

Enclanche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

YDisplay **target** **newSequence**

Le nom de la séquence sera donné au moment de l'appel à `saveSequence()`, une fois la séquence terminée.

Retourne :

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

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

display→pauseSequence()YDisplay pauseSequence YDisplay

Attend pour la durée spécifiée (en millisecondes) avant de jouer les commandes suivantes de la séquence active.

YDisplay **target pauseSequence delay_ms**

Cette méthode peut être utilisée lors de l'enregistrement d'une séquence d'affichage, pour insérer une attente mesurée lors de l'exécution (mais sans effet immédiat). Cette méthode peut aussi être appelée dynamiquement pendant l'exécution d'une séquence enregistrée, pour suspendre temporairement ou reprendre l'exécution. Pour annuler une attente, appelez simplement la méthode avec une attente de zéro.

Paramètres :

delay_ms la durée de l'attente, en millisecondes

Retourne :

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

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

display→playSequence()YDisplay playSequence YDisplay

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes `newSequence()` et `saveSequence()`.

YDisplay **target playSequence sequenceName**

Paramètres :

sequenceName le nom de la nouvelle séquence créée

Retourne :

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

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

display→resetAll()YDisplay resetAll YDisplay

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

```
YDisplay target 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()YDisplay saveSequence YDisplay

Termine l'enregistrement d'une séquence et la sauvegarde sur la mémoire interne de l'écran, sous le nom choisi.

YDisplay **target saveSequence sequenceName**

La séquence peut être rejouée ultérieurement à l'aide de la méthode `playSequence()`.

Paramètres :

sequenceName le nom de la nouvelle séquence créée

Retourne :

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

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

display→set_brightness() YDisplay
display→setBrightness()YDisplay set_brightness

Modifie la luminositéde l'écran.

```
YDisplay target 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()YDisplay set_enabled

Modifie l'état d'activité de l'écran.

YDisplay **target** **set_enabled** **newval**

Paramètres :

newval soit `Y_ENABLED_FALSE`, soit `Y_ENABLED_TRUE`, selon l'état d'activité de l'écran

Retourne :

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

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

display→set_logicalName() YDisplay
display→setLogicalName()YDisplay set_logicalName

Modifie le nom logique de l'ecran.

```
YDisplay target set_logicalName newval
```

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

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'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()YDisplay set_orientation

Modifie l'orientation de l'écran.

YDisplay **target** **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()YDisplay set_startupSeq

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

YDisplay **target set_startupSeq newval**

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

Paramètres :

newval une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

Retourne :

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

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

display→stopSequence()YDisplay stopSequence YDisplay

Arrête immédiatement la séquence d'affichage actuellement jouée sur l'écran.

YDisplay **target** **stopSequence**

L'affichage est laissé tel quel.

Paramètres :

sequenceName le nom de la nouvelle séquence créée

Retourne :

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

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

display→swapLayerContent()YDisplay swapLayerContent YDisplay

Permute le contentu de deux couches d'affichage.

```
YDisplay target swapLayerContent layerIdA layerIdB
```

La couleur et la transparence de tous les pixels des deux couches sont permutées. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. En particulier, la visibilité des deux couches reste inchangée. Cela permet d'implémenter très efficacement un affichage par double-buffering, en utilisant une couche cachée et une couche visible. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

Paramètres :

layerIdA l'identifiant de la première couche (un chiffre parmi 0..layerCount-1)

layerIdB l'identifiant de la deuxième couche (un chiffre parmi 0..layerCount-1)

Retourne :

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

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

display→upload()YDisplay upload YDisplay

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

```
YDisplay target upload pathname content
```

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

Paramètres :

pathname nom complet du fichier, y compris le chemin d'accès.

content contenu du fichier à télécharger

Retourne :

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

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

3.15. Interface des objets DisplayLayer

Un DisplayLayer est une couche de contenu affichable (images, texte, etc.). Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Méthodes des objets `YDisplayLayer`
displaylayer→clear() Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.
displaylayer→clearConsole() Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.
displaylayer→consoleOut(**text**) Affiche un message dans la zone de console, et déplace le curseur de la console à la fin du texte.
displaylayer→drawBar(**x1**, **y1**, **x2**, **y2**) Dessine un rectangle plein à une position spécifiée.
displaylayer→drawBitmap(**x**, **y**, **w**, **bitmap**, **bgcol**) Dessine un bitmap à la position spécifiée de la couche.
displaylayer→drawCircle(**x**, **y**, **r**) Dessine un cercle vide à une position spécifiée.
displaylayer→drawDisc(**x**, **y**, **r**) Dessine un disque plein à une position spécifiée.
displaylayer→drawImage(**x**, **y**, **imagename**) Dessine une image GIF à la position spécifiée de la couche.
displaylayer→drawPixel(**x**, **y**) Dessine un pixel unique à une position spécifiée.
displaylayer→drawRect(**x1**, **y1**, **x2**, **y2**) Dessine un rectangle vide à une position spécifiée.
displaylayer→drawText(**x**, **y**, **anchor**, **text**) Affiche un texte à la position spécifiée de la couche.
displaylayer→get_display() Retourne l'YDisplay parent.
displaylayer→get_displayHeight() Retourne la hauteur de l'écran, en pixels.
displaylayer→get_displayWidth() Retourne la largeur de l'écran, en pixels.

displaylayer→get_layerHeight()

Retourne la hauteur des couches affichables, en pixels.

displaylayer→get_layerWidth()

Retourne la largeur des couches affichables, en pixels.

displaylayer→hide()

Cache la couche de dessin.

displaylayer→lineTo(**x**, **y**)

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

displaylayer→moveTo(**x**, **y**)

Déplace le point de dessin courant de cette couche à la position spécifiée.

displaylayer→reset()

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

displaylayer→selectColorPen(**color**)

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

displaylayer→selectEraser()

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de copie d'images bitmaps.

displaylayer→selectFont(**fontname**)

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

displaylayer→selectGrayPen(**graylevel**)

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

displaylayer→setAntialiasingMode(**mode**)

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

displaylayer→setConsoleBackground(**bgcol**)

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

displaylayer→setConsoleMargins(**x1**, **y1**, **x2**, **y2**)

Configure les marges d'affichage pour la fonction `consoleOut`.

displaylayer→setConsoleWordWrap(**wordwrap**)

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

displaylayer→setLayerPosition(**x**, **y**, **scrollTime**)

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

displaylayer→unhide()

Affiche la couche.

displaylayer→clear()YDisplayLayer clear — YDisplayLayer

Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.

YDisplay **target** [-layer **layerId**] **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 clearConsole — YDisplayLayer

Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.

YDisplay **target** [-layer **layerId**] **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()YDisplayLayer consoleOut — YDisplayLayer

Affiche un message dans la zone de console, et déplace le curseur de la console à la fin du texte.

YDisplay **target** [-layer **layerId**] **consoleOut** **text**

Le curseur revient automatiquement en début de ligne suivante lorsqu'un saut de ligne est rencontré, ou lorsque la marge droite est atteinte. Lorsque le texte à afficher s'apprête à dépasser la marge inférieure, le contenu de la zone de console est automatiquement décalé vers le haut afin de laisser la place à la nouvelle ligne de texte.

Paramètres :

text le message à afficher

Retourne :

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

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

displaylayer→drawBar()YDisplayLayer drawBar YDisplayLayer

Dessine un rectangle plein à une position spécifiée.

YDisplay **target** [-layer **layerId**] **drawBar** **x1 y1 x2 y2**

Paramètres :

x1 la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle
y1 la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle
x2 la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle
y2 la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

Retourne :

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

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

displaylayer→drawBitmap()YDisplayLayer drawBitmap — YDisplayLayer

Dessine un bitmap à la position spécifiée de la couche.

YDisplay **target** [-layer **layerId**] **drawBitmap** **x y w bitmap bgcol**

Le bitmap est passé sous forme d'un objet binaire, où chaque bit correspond à un pixel, de gauche à droite et de haut en bas. Le bit de poids fort de chaque octet correspond au pixel de gauche, et le bit de poids faible au pixel le plus à droite. Les bits à 1 sont dessinés avec la couleur active de la couche. Les bits à 0 avec la couleur de fond spécifiée, sauf si la valeur -1 a été choisie, auquel cas ils ne sont pas dessinés (ils sont considérés comme transparents). Chaque ligne commence sur un nouvel octet. La hauteur du bitmap est donnée implicitement par la taille de l'objet binaire.

Paramètres :

x	la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du bitmap
y	la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du bitmap
w	la largeur du bitmap, en pixels
bitmap	l'objet binaire contenant le bitmap
bgcol	le niveau de gris à utiliser pour les bits à zéro (0 = noir, 255 = blanc), ou -1 pour lasser les pixels inchangés

Retourne :

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

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

displaylayer→drawCircle()YDisplayLayer drawCircle YDisplayLayer

Dessine un cercle vide à une position spécifiée.

```
YDisplay target [-layer layerId] 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()YDisplayLayer drawDisc YDisplayLayer

Dessine un disque plein à une position spécifiée.

YDisplay **target** [-layer **layerId**] **drawDisc** **x y r**

Paramètres :

- **x** la distance en pixels depuis la gauche de la couche jusqu'au centre du disque
- **y** la distance en pixels depuis le haut de la couche jusqu'au centre du disque
- **r** le rayon du disque, en pixels

Retourne :

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

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

displaylayer→drawImage()YDisplayLayer drawImage YDisplayLayer

Dessine une image GIF à la position spécifiée de la couche.

YDisplay **target** [-layer **layerId**] **drawImage x y imagename**

L'image GIF doit avoir été préalablement préchargée dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une image bitmap, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier d'image manquant ou d'un format de fichier invalide.

Paramètres :

x	la distance en pixels depuis la gauche de la couche jusqu'au bord gauche de l'image
y	la distance en pixels depuis le haut de la couche jusqu'au bord supérieur de l'image
imagename	le nom du fichier GIF à afficher

Retourne :

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

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

displaylayer→drawPixel()YDisplayLayer drawPixel — YDisplayLayer

Dessine un pixel unique à une position spécifiée.

YDisplay **target** [-layer **layerId**] **drawPixel** **x y**

Paramètres :

x la distance en pixels depuis la gauche de la couche

y la distance en pixels depuis le haut de la couche

Retourne :

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

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

displaylayer→drawRect()YDisplayLayer drawRect — YDisplayLayer

Dessine un rectangle vide à une position spécifiée.

YDisplay **target** [-layer **layerId**] **drawRect** **x1 y1 x2 y2**

Paramètres :

- **x1** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle
- **y1** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle
- **x2** la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle
- **y2** la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

Retourne :

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

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

displaylayer→drawText()YDisplayLayer drawText YDisplayLayer

Affiche un texte à la position spécifiée de la couche.

YDisplay **target** [-layer **layerId**] **drawText** **x y anchor text**

Le point du texte qui sera aligné sur la position spécifiée est appelé point d'ancrage, et peut être choisi parmi plusieurs options.

Paramètres :

x	la distance en pixels depuis la gauche de la couche jusqu'au point d'ancrage du texte
y	la distance en pixels depuis le haut de la couche jusqu'au point d'ancrage du texte
anchor	le point d'ancrage du texte, choisi parmi l'énumération `Y_ALIGN`: `Y_ALIGN_TOP_LEFT`, `Y_ALIGN_CENTER_LEFT`, `Y_ALIGN_BASELINE_LEFT`, `Y_ALIGN_BOTTOM_LEFT`, `Y_ALIGN_TOP_CENTER`, `Y_ALIGN_CENTER`, `Y_ALIGN_BASELINE_CENTER`, `Y_ALIGN_BOTTOM_CENTER`, `Y_ALIGN_TOP_DECIMAL`, `Y_ALIGN_CENTER_DECIMAL`, `Y_ALIGN_BASELINE_DECIMAL`, `Y_ALIGN_BOTTOM_DECIMAL`, `Y_ALIGN_TOP_RIGHT`, `Y_ALIGN_CENTER_RIGHT`, `Y_ALIGN_BASELINE_RIGHT`, `Y_ALIGN_BOTTOM_RIGHT`.
text	le texte à afficher

Retourne :

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

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

displaylayer→get_displayHeight() YDisplayLayer
displaylayer→displayHeight()YDisplayLayer
get_displayHeight

Retourne la hauteur de l'écran, en pixels.

YDisplay **target** [-layer **layerId**] **get_displayHeight**

Retourne :
un entier représentant la hauteur de l'écran, en pixels

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

displaylayer→get_displayWidth() YDisplayLayer
displaylayer→displayWidth()YDisplayLayer
get_displayWidth

Retourne la largeur de l'écran, en pixels.

YDisplay **target** [-layer **layerId**] **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()YDisplayLayer
get_layerHeight

Retourne la hauteur des couches affichables, en pixels.

YDisplay **target** [-layer **layerId**] **get_layerHeight**

Retourne :
un entier représentant la hauteur des couches affichables, en pixels. En cas d'erreur, déclenche une exception ou retourne Y_LAYERHEIGHT_INVALID.

displaylayer→get_layerWidth() YDisplayLayer
displaylayer→layerWidth()YDisplayLayer
get_layerWidth

Retourne la largeur des couches affichables, en pixels.

YDisplay **target** [-layer **layerId**] **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()YDisplayLayer hide — YDisplayLayer

Cache la couche de dessin.

YDisplay **target** [-layer **layerId**] **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()YDisplayLayer lineTo — YDisplayLayer

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

```
YDisplay target [-layer layerId] 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()YDisplayLayer moveTo YDisplayLayer

Déplace le point de dessin courant de cette couche à la position spécifiée.

YDisplay **target** [-layer **layerId**] **moveTo** **x y**

Paramètres :

x la distance en pixels depuis la gauche de la couche de dessin

y la distance en pixels depuis le haut de la couche de dessin

Retourne :

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

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

displaylayer→reset()YDisplayLayer reset — YDisplayLayer

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

YDisplay **target** [-layer **layerId**] **reset**

Réinitialise la position du point de dessin courant au coin supérieur gauche, et la couleur de dessin à la valeur la plus lumineuse. Si vous désirez simplement effacer le contenu de la couche, utilisez plutôt la méthode `clear()`.

Retourne :

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

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

displaylayer→selectColorPen()YDisplayLayer selectColorPen YDisplayLayer

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

YDisplay **target** [-layer **layerId**] **selectColorPen** **color**

La couleur est fournie sous forme de couleur RGB. Pour les écrans monochromes ou en niveaux de gris, la couleur est automatiquement ramenée dans les valeurs permises.

Paramètres :

color la couleur RGB désirée (sous forme d'entier 24 bits)

Retourne :

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

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

displaylayer→selectEraser()YDisplayLayer selectEraser YDisplayLayer

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de copie d'images bitmaps.

YDisplay **target** [-layer **layerId**] **selectEraser**

Tous les points dessinés à la gomme redeviennent transparents (comme ils l'étaient lorsque la couche était vide), rendant ainsi visibles les couches inférieures.

Retourne :

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

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

displaylayer→selectFont()YDisplayLayer selectFont YDisplayLayer

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

YDisplay **target** [-layer **layerId**] **selectFont** **fontname**

La police est spécifiée par le nom de son fichier. Vous pouvez utiliser l'une des polices prédéfinies dans le module, ou une autre police que vous avez préalablement préchargé dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une police de caractères, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier de police manquant ou d'un format de fichier invalide.

Paramètres :

fontname le nom du fichier définissant la police de caractères

Retourne :

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

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

displaylayer→selectGrayPen()YDisplayLayer selectGrayPen YDisplayLayer

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

YDisplay **target** [-layer **layerId**] **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 setAntialiasingMode YDisplayLayer

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

YDisplay **target** [-layer **layerId**] **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
YDisplayLayer setConsoleBackground

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

YDisplay **target** [-layer **layerId**] **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 setConsoleMargins — YDisplayLayer

Configure les marges d'affichage pour la fonction `consoleOut`.

YDisplay **target** [-layer **layerId**] **setConsoleMargins** **x1 y1 x2 y2**

Paramètres :

x1 la distance en pixels depuis la gauche de la couche jusqu'à la marge gauche

y1 la distance en pixels depuis le haut de la couche jusqu'à la marge supérieure

x2 la distance en pixels depuis la gauche de la couche jusqu'à la marge droite

y2 la distance en pixels depuis le haut de la couche jusqu'à la marge inférieure

Retourne :

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

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

displaylayer→setConsoleWordWrap()YDisplayLayer setConsoleWordWrap — YDisplayLayer

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

YDisplay **target** [-layer **layerId**] **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 setLayerPosition YDisplayLayer

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

YDisplay **target** [-layer **layerId**] **setLayerPosition** **x y scrollTime**

Lorsqu'une durée de défilement est configurée, la position d'affichage de la couche est automatiquement mise à jour durant les millisecondes suivantes pour animer le déplacement.

Paramètres :

x	la distance en pixels depuis la gauche de l'écran jusqu'à l'origine de la couche.
y	la distance en pixels depuis le haut de l'écran jusqu'à l'origine de la couche.
scrollTime	durée en millisecondes du déplacement, ou 0 si le déplacement doit être immédiat.

Retourne :

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

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

displaylayer→unhide()YDisplayLayer unhide YDisplayLayer

Affiche la couche.

YDisplay **target** [-layer **layerId**] **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.

3.16. Interface de contrôle de l'alimentation

La librairie de programmation Yoctopuce permet de contrôler la source d'alimentation qui doit être utilisée pour les fonctions du module consommant beaucoup de courant. Le module est par ailleurs capable de couper automatiquement l'alimentation externe lorsqu'il détecte que la tension a trop chuté (batterie épuisée).

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
yFindDualPower(**func**) Permet de retrouver un contrôle d'alimentation d'après un identifiant donné.
yFirstDualPower() Commence l'énumération des contrôles d'alimentation accessibles par la librairie.
Méthodes des objets `YDualPower`
dualpower→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'alimentation au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
dualpower→get_advertisedValue() Retourne la valeur courante du contrôle d'alimentation (pas plus de 6 caractères).
dualpower→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.
dualpower→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.
dualpower→get_extVoltage() Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.
dualpower→get_friendlyName() Retourne un identifiant global du contrôle d'alimentation au format `NOM_MODULE.NOM_FONCTION`.
dualpower→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
dualpower→get_functionId() Retourne l'identifiant matériel du contrôle d'alimentation, sans référence au module.
dualpower→get_hardwareId() Retourne l'identifiant matériel unique du contrôle d'alimentation au format `SERIAL.FUNCTIONID`.
dualpower→get_logicalName()

Retourne le nom logique du contrôle d'alimentation.

dualpower→get_module()

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

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

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

dualpower→get_powerControl()

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

dualpower→get_powerState()

Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

dualpower→get_userData()

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

dualpower→isOnline()

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

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

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

dualpower→load(**msValidity**)

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

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

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

dualpower→nextDualPower()

Continue l'énumération des contrôles d'alimentation commencée à l'aide de `yFirstDualPower()`.

dualpower→registerValueCallback(**callback**)

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

dualpower→set_logicalName(**newval**)

Modifie le nom logique du contrôle d'alimentation.

dualpower→set_powerControl(**newval**)

Modifie le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

dualpower→set_userData(**data**)

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

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

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

dualpower→get_advertisedValue() YDualPower
dualpower→advertisedValue()YDualPower
get_advertisedValue

Retourne la valeur courante du contrôle d'alimentation (pas plus de 6 caractères).

YDualPower **target** **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_extVoltage() YDualPower
dualpower→extVoltage()YDualPower get_extVoltage

Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.

YDualPower **target get_extVoltage**

Retourne :
un entier représentant la tension mesurée sur l'alimentation de puissance externe, en millivolts

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

dualpower→get_logicalName() YDualPower
dualpower→logicalName()YDualPower
get_logicalName

Retourne le nom logique du contrôle d'alimentation.

YDualPower **target 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_powerControl() YDualPower
dualpower→powerControl()YDualPower
get_powerControl

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

YDualPower **target get_powerControl**

Retourne :

une valeur parmi `Y_POWERCONTROL_AUTO`, `Y_POWERCONTROL_FROM_USB`, `Y_POWERCONTROL_FROM_EXT` et `Y_POWERCONTROL_OFF` représentant le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant

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

dualpower→get_powerState() YDualPower
dualpower→powerState()YDualPower
get_powerState

Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

YDualPower **target 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→set_logicalName() YDualPower
dualpower→setLogicalName()YDualPower
set_logicalName

Modifie le nom logique du contrôle d'alimentation.

```
YDualPower target set_logicalName newval
```

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

Paramètres :

newval une chaîne de caractères représentant le nom logique du contrôle d'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()YDualPower
set_powerControl

Modifie le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

YDualPower **target** **set_powerControl** **newval**

Paramètres :

newval une valeur parmi `Y_POWERCONTROL_AUTO`, `Y_POWERCONTROL_FROM_USB`, `Y_POWERCONTROL_FROM_EXT` et `Y_POWERCONTROL_OFF` représentant le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant

Retourne :

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

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

3.17. Interface de la fonction Files

L'interface de stockage de fichiers permet de stocker des fichiers sur certains modules, par exemple pour personnaliser un service web (dans le cas d'un module connecté au réseau) ou pour ajouter un police de caractères (dans le cas d'un module d'affichage).

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
yFindFiles(**func**) Permet de retrouver un système de fichier d'après un identifiant donné.
yFirstFiles() Commence l'énumération des système de fichier accessibles par la librairie.
Méthodes des objets `YFiles`
files→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
files→download(**pathname**) Télécharge le fichier choisi du filesystème et retourne son contenu.
files→download_async(**pathname**, **callback**, **context**) Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.
files→format_fs() Rétabli le système de fichier dans on état original, défragmenté.
files→get_advertisedValue() Retourne la valeur courante du système de fichier (pas plus de 6 caractères).
files→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.
files→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.
files→get_filesCount() Retourne le nombre de fichiers présents dans le système de fichier.
files→get_freeSpace() Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.
files→get_friendlyName() Retourne un identifiant global du système de fichier au format `NOM_MODULE.NOM_FONCTION`.
files→get_functionDescriptor()

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

files→get_functionId()

Retourne l'identifiant matériel du système de fichier, sans référence au module.

files→get_hardwareId()

Retourne l'identifiant matériel unique du système de fichier au format SERIAL.FUNCTIONID.

files→get_list(**pattern**)

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

files→get_logicalName()

Retourne le nom logique du système de fichier.

files→get_module()

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

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

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

files→get_userData()

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

files→isOnline()

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

files→load(**msValidity**)

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

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

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

files→nextFiles()

Continue l'énumération des système de fichier commencée à l'aide de yFirstFiles().

files→registerValueCallback(**callback**)

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

files→remove(**pathname**)

Efface un fichier, spécifié par son path complet, du système de fichier.

files→set_logicalName(**newval**)

Modifie le nom logique du système de fichier.

files→set_userData(**data**)

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

files→upload(**pathname**, **content**)

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

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

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

files→download()YFiles download — YFiles

Télécharge le fichier choisi du filesystème et retourne son contenu.

```
YFiles target download pathname
```

Paramètres :

pathname nom complet du fichier à charger, y compris le chemin d'accès.

Retourne :

le contenu du fichier chargé sous forme d'objet binaire

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

files→format_fs()YFiles format_fs YFiles

Rétabli le système de fichier dans on état original, défragmenté.

```
YFiles target 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()YFiles get_advertisedValue

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

YFiles **target 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_filesCount() YFiles
files→filesCount()YFiles get_filesCount

Retourne le nombre de fichiers présents dans le système de fichier.

YFiles **target** **get_filesCount**

Retourne :
un entier représentant le nombre de fichiers présents dans le système de fichier

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

files→get_freeSpace() YFiles
files→freeSpace()YFiles get_freeSpace

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

```
YFiles target 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_list() YFiles
files→list()YFiles get_list

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

```
YFiles target 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()YFiles get_logicalName

Retourne le nom logique du système de fichier.

YFiles **target get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du système de fichier.

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

files→remove()YFiles remove YFiles

Efface un fichier, spécifié par son path complet, du système de fichier.

YFiles **target 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()YFiles set_logicalName

Modifie le nom logique du système de fichier.

```
YFiles target set_logicalName newval
```

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

Paramètres :

newval une chaîne de caractères représentant le nom logique du système de fichier.

Retourne :

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

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

files→upload()YFiles upload YFiles

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

YFiles **target upload pathname content**

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

Paramètres :

pathname nom complet du fichier, y compris le chemin d'accès.

content contenu du fichier à télécharger

Retourne :

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

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

3.18. Interface de la fonction GenericSensor

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
yFindGenericSensor(**func**) Permet de retrouver un capteur générique d'après un identifiant donné.
yFirstGenericSensor() Commence l'énumération des capteurs génériques accessibles par la librairie.
Méthodes des objets `YGenericSensor`
genericsensor→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
genericsensor→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur générique au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
genericsensor→get_advertisedValue() Retourne la valeur courante du capteur générique (pas plus de 6 caractères).
genericsensor→get_currentRawValue() Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).
genericsensor→get_currentValue() Retourne la valeur mesurée actuelle.
genericsensor→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.
genericsensor→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.
genericsensor→get_friendlyName() Retourne un identifiant global du capteur générique au format `NOM_MODULE.NOM_FONCTION`.
genericsensor→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
genericsensor→get_functionId() Retourne l'identifiant matériel du capteur générique, sans référence au module.
genericsensor→get_hardwareId()

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_signalBias()

Retourne le biais du signal électrique pour la correction du point zéro.

genericsensor→get_signalRange()

Retourne la plage de signal électrique utilisée par le capteur.

genericsensor→get_signalUnit()

Retourne l'unité du signal électrique utilisée par le capteur.

genericsensor→get_signalValue()

Retourne la valeur mesurée du signal électrique utilisée par le capteur.

genericsensor→get_unit()

Retourne l'unité dans laquelle la 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→load(**msValidity**)

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

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

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

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

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

genericsensor→nextGenericSensor()

Continue l'énumération des capteurs génériques commencée à l'aide de `yFirstGenericSensor()`.

genericsensor→registerTimedReportCallback(**callback**)

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

genericsensor→registerValueCallback(**callback**)

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

genericsensor→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

genericsensor→set_logFrequency(**newval**)

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

genericsensor→set_logicalName(**newval**)

Modifie le nom logique du capteur générique.

genericsensor→set_lowestValue(**newval**)

Modifie la mémoire de valeur minimale observée.

genericsensor→set_reportFrequency(**newval**)

Modifie la fréquence de notification périodique des valeurs mesurées.

genericsensor→set_resolution(**newval**)

Modifie la résolution des valeurs physique mesurées.

genericsensor→set_signalBias(**newval**)

Modifie le biais du signal électrique pour la correction du point zéro.

genericsensor→set_signalRange(**newval**)

Modifie la plage de signal électrique utilisée par le capteur.

genericsensor→set_unit(**newval**)

Change l'unité dans laquelle la valeur mesurée est exprimée.

genericsensor→set_userData(**data**)

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

genericsensor→set_valueRange(**newval**)

Modifie la plage de valeurs physiques mesurés par le capteur.

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

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

genericsensor→zeroAdjust()

Ajuste le biais du signal de sorte à ce que la valeur actuelle du signal soit interprétée comme zéro (tare).

genericsensor→calibrateFromPoints() YGenericSensor
YGenericSensor calibrateFromPoints

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

YGenericSensor **target calibrateFromPoints rawValues refValues**

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

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

Paramètres :

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

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

Retourne :

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

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

genericsensor→get_advertisedValue() YGenericSensor genericsensor→advertisedValue()YGenericSensor get_advertisedValue

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

YGenericSensor **target 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()YGenericSensor
get_currentRawValue

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

YGenericSensor **target** **get_currentRawValue**

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

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

genericsensor→get_currentValue() YGenericSensor genericsensor→currentValue()YGenericSensor get_currentValue

Retourne la valeur mesurée actuelle.

YGenericSensor **target 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_highestValue() YGenericSensor genericsensor→highestValue()YGenericSensor get_highestValue

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

YGenericSensor **target** **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()YGenericSensor 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.

YGenericSensor **target get_logFrequency**

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

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

genericsensor→get_logicalName() YGenericSensor
genericsensor→logicalName()YGenericSensor
get_logicalName

Retourne le nom logique du capteur générique.

```
YGenericSensor target get_logicalName
```

Retourne :
une chaîne de caractères représentant le nom logique du capteur générique.

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

genericsensor→get_lowestValue() YGenericSensor
genericsensor→lowestValue()YGenericSensor
get_lowestValue

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

YGenericSensor **target 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_recordedData() YGenericSensor
genericsensor→recordedData()YGenericSensor
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.

```
YGenericSensor target get_recordedData startTime endTime
```

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

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

Paramètres :

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

endTime la 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()YGenericSensor get_reportFrequency

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

YGenericSensor **target get_reportFrequency**

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

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

genericsensor→get_resolution() YGenericSensor genericsensor→resolution()YGenericSensor get_resolution

Retourne la résolution des valeurs mesurées.

YGenericSensor **target** **get_resolution**

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

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

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

genericsensor→get_signalBias() YGenericSensor genericsensor→signalBias()YGenericSensor get_signalBias

Retourne le biais du signal électrique pour la correction du point zéro.

YGenericSensor **target 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()YGenericSensor get_signalRange

Retourne la plage de signal électrique utilisée par le capteur.

YGenericSensor **target** **get_signalRange**

Retourne :
une chaîne de caractères représentant la plage de signal électrique utilisée par le capteur

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

genericsensor→get_signalUnit() YGenericSensor
genericsensor→signalUnit()YGenericSensor
get_signalUnit

Retourne l'unité du signal électrique utilisée par le capteur.

YGenericSensor **target get_signalUnit**

Retourne :
une chaîne de caractères représentant l'unité du signal électrique utilisée par le capteur

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

genericsensor→get_signalValue() YGenericSensor genericsensor→signalValue()YGenericSensor get_signalValue

Retourne la valeur mesurée du signal électrique utilisée par le capteur.

YGenericSensor **target** **get_signalValue**

Retourne :
une valeur numérique représentant la valeur mesurée du signal électrique utilisée par le capteur

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

genericsensor→get_unit() YGenericSensor genericsensor→unit()YGenericSensor get_unit

Retourne l'unité dans laquelle la measure est exprimée.

```
YGenericSensor target 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_valueRange() YGenericSensor genericsensor→valueRange()YGenericSensor get_valueRange

Retourne la plage de valeurs physiques mesurés par le capteur.

```
YGenericSensor target get_valueRange
```

Retourne :
une chaîne de caractères représentant la plage de valeurs physiques mesurés par le capteur

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

genericsensor→loadCalibrationPoints() YGenericSensor
YGenericSensor loadCalibrationPoints

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

YGenericSensor **target loadCalibrationPoints rawValues refValues**

Paramètres :

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

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

Retourne :

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

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

genericsensor→set_highestValue() YGenericSensor genericsensor→setHighestValue()YGenericSensor set_highestValue

Modifie la mémoire de valeur maximale observée.

```
YGenericSensor target set_highestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

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

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

genericsensor→set_logFrequency() YGenericSensor genericsensor→setLogFrequency()YGenericSensor set_logFrequency

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

YGenericSensor **target set_logFrequency newval**

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

Paramètres :

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

Retourne :

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

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

genericsensor→set_logicalName() YGenericSensor genericsensor→setLogicalName()YGenericSensor set_logicalName

Modifie le nom logique du capteur générique.

```
YGenericSensor target set_logicalName newval
```

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

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur 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()YGenericSensor
set_lowestValue

Modifie la mémoire de valeur minimale observée.

YGenericSensor **target set_lowestValue newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

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

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

genericsensor→set_reportFrequency() YGenericSensor
genericsensor→setReportFrequency()
YGenericSensor set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

```
YGenericSensor target set_reportFrequency newval
```

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

Paramètres :

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

Retourne :

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

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

genericsensor→set_resolution() YGenericSensor genericsensor→setResolution()YGenericSensor set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YGenericSensor target set_resolution newval
```

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

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

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

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

genericsensor→set_signalBias() YGenericSensor genericsensor→setSignalBias()YGenericSensor set_signalBias

Modifie le biais du signal électrique pour la correction du point zéro.

```
YGenericSensor target 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()YGenericSensor
set_signalRange

Modifie la plage de signal électrique utilisée par le capteur.

YGenericSensor **target set_signalRange newval**

Paramètres :

newval une chaîne de caractères représentant la plage de signal électrique utilisée par le capteur

Retourne :

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

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

genericsensor→set_unit() YGenericSensor genericsensor→setUnit()YGenericSensor set_unit

Change l'unité dans laquelle la valeur mesurée est exprimée.

```
YGenericSensor target 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_valueRange() YGenericSensor genericsensor→setValueRange()YGenericSensor set_valueRange

Modifie la plage de valeurs physiques mesurés par le capteur.

```
YGenericSensor target set_valueRange newval
```

Le changement de plage peut avoir pour effet de bord un changement automatique de la résolution affichée.

Paramètres :

newval une chaîne de caractères représentant la plage de valeurs physiques mesurés par le capteur

Retourne :

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

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

genericsensor→zeroAdjust()YGenericSensor YGenericSensor zeroAdjust

Ajuste le biais du signal de sorte à ce que la valeur actuelle du signal soit interprétée comme zéro (tare).

YGenericSensor **target** **zeroAdjust**

Retourne :

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

3.19. Interface de la fonction Gyro

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
yFindGyro(**func**) Permet de retrouver un gyroscope d'après un identifiant donné.
yFirstGyro() Commence l'énumération des gyroscopes accessibles par la librairie.
Méthodes des objets `YGyro`
gyro→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
gyro→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du gyroscope au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
gyro→get_advertisedValue() Retourne la valeur courante du gyroscope (pas plus de 6 caractères).
gyro→get_currentRawValue() Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés par seconde, sous forme de nombre à virgule.
gyro→get_currentValue() Retourne la valeur actuelle de la vitesse angulaire, en degrés par seconde, sous forme de nombre à virgule.
gyro→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.
gyro→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.
gyro→get_friendlyName() Retourne un identifiant global du gyroscope au format `NOM_MODULE.NOM_FONCTION`.
gyro→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
gyro→get_functionId() Retourne l'identifiant matériel du gyroscope, sans référence au module.
gyro→get_hardwareId()

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_unit()

Retourne l'unité dans laquelle la vitesse angulaire est exprimée.

gyro→get_userData()

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

gyro→get_xValue()

Retourne la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule.

gyro→get_yValue()

Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.

gyro→get_zValue()

Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.

gyro→isOnline()

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

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

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

gyro→load(**msValidity**)

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

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

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

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

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

gyro→nextGyro()

Continue l'énumération des gyroscopes commencée à l'aide de `yFirstGyro()`.

gyro→registerAnglesCallback(**callback**)

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

gyro→registerQuaternionCallback(**callback**)

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

gyro→registerTimedReportCallback(**callback**)

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

gyro→registerValueCallback(**callback**)

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

gyro→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

gyro→set_logFrequency(**newval**)

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

gyro→set_logicalName(**newval**)

Modifie le nom logique du gyroscope.

gyro→set_lowestValue(**newval**)

Modifie la mémoire de valeur minimale observée.

gyro→set_reportFrequency(**newval**)

Modifie la fréquence de notification périodique des valeurs mesurées.

gyro→set_resolution(**newval**)

Modifie la résolution des valeurs physique mesurées.

gyro→set_userData(**data**)

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

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

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

gyro→calibrateFromPoints()YGyro calibrateFromPoints YGyro

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

YGyro **target calibrateFromPoints rawValues refValues**

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

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

Paramètres :

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

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

Retourne :

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

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

gyro→get_advertisedValue() YGyro
gyro→advertisedValue()YGyro get_advertisedValue

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

YGyro **target get_advertisedValue**

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

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

gyro→get_currentRawValue() YGyro
gyro→currentRawValue()YGyro
get_currentRawValue

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

YGyro **target** **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()YGyro get_currentValue

Retourne la valeur actuelle de la vitesse angulaire, en degrés par seconde, sous forme de nombre à virgule.

```
YGyro target 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_highestValue() YGyro
gyro→highestValue()YGyro get_highestValue

Retourne la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module.

YGyro **target** **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()YGyro 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.

YGyro **target get_logFrequency**

Retourne :

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

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

gyro→get_logicalName() YGyro
gyro→logicalName()YGyro get_logicalName

Retourne le nom logique du gyroscope.

YGyro **target** **get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du gyroscope.

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

gyro→get_lowestValue() YGyro
gyro→lowestValue()YGyro get_lowestValue

Retourne la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module.

YGyro **target get_lowestValue**

Retourne :

une valeur numérique représentant la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module

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

gyro→get_recordedData() YGyro gyro→recordedData()YGyro 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.

```
YGyro target get_recordedData startTime endTime
```

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

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

Paramètres :

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

endTime la 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()YGyro get_reportFrequency

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

```
YGyro target get_reportFrequency
```

Retourne :

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

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

gyro→get_resolution() YGyro
gyro→resolution()YGyro get_resolution

Retourne la résolution des valeurs mesurées.

```
YGyro target get_resolution
```

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

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

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

gyro→get_unit() YGyro
gyro→unit()YGyro get_unit

Retourne l'unité dans laquelle la vitesse angulaire est exprimée.

YGyro **target 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→loadCalibrationPoints()YGyro loadCalibrationPoints YGyro

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

YGyro **target loadCalibrationPoints rawValues refValues**

Paramètres :

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

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

Retourne :

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

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

gyro→set_highestValue() YGyro
gyro→setHighestValue()YGyro set_highestValue

Modifie la mémoire de valeur maximale observée.

YGyro **target set_highestValue newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

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

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

gyro→set_logFrequency() YGyro
gyro→setLogFrequency()YGyro set_logFrequency

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

```
YGyro target set_logFrequency newval
```

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

Paramètres :

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

Retourne :

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

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

gyro→set_logicalName() YGyro
gyro→setLogicalName()YGyro set_logicalName

Modifie le nom logique du gyroscope.

```
YGyro target set_logicalName newval
```

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

Paramètres :

newval une chaîne de caractères représentant le nom logique du 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()YGyro set_lowestValue

Modifie la mémoire de valeur minimale observée.

```
YGyro target set_lowestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

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

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

gyro→set_reportFrequency() YGyro
gyro→setReportFrequency()YGyro
set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

```
YGyro target set_reportFrequency newval
```

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

Paramètres :

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

Retourne :

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

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

gyro→set_resolution() YGyro
gyro→setResolution()YGyro set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YGyro target set_resolution newval
```

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

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

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

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

3.20. Interface d'un port de Yocto-hub

Les objets YHubPort permettent de contrôler l'alimentation des ports d'un YoctoHub, ainsi que de détecter si un module y est raccordé et lequel. Un YHubPort reçoit toujours automatiquement comme nom logique le numéro de série unique du module Yoctopuce qui y est connecté.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

yFindHubPort(**func**)

Permet de retrouver un port de Yocto-hub d'après un identifiant donné.

yFirstHubPort()

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

Méthodes des objets `YHubPort`

hubport→describe()

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

hubport→get_advertisedValue()

Retourne la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

hubport→get_baudRate()

Retourne la vitesse de transfert utilisée par le port de Yocto-hub, en kbps.

hubport→get_enabled()

Retourne vrai si le port du Yocto-hub est alimenté, faux sinon.

hubport→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

hubport→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

hubport→get_friendlyName()

Retourne un identifiant global du port de Yocto-hub au format `NOM_MODULE.NOM_FONCTION`.

hubport→get_functionDescriptor()

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

hubport→get_functionId()

Retourne l'identifiant matériel du port de Yocto-hub, sans référence au module.

hubport→get_hardwareId()

Retourne l'identifiant matériel unique du port de Yocto-hub au format `SERIAL.FUNCTIONID`.

hubport→get_logicalName()

Retourne le nom logique du port de Yocto-hub.

hubport→get_module()

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

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

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

hubport→get_portState()

Retourne l'état actuel du port de Yocto-hub.

hubport→get_userData()

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

hubport→isOnline()

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

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

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

hubport→load(**msValidity**)

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

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

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

hubport→nextHubPort()

Continue l'énumération des port de Yocto-hub commencée à l'aide de `yFirstHubPort()`.

hubport→registerValueCallback(**callback**)

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

hubport→set_enabled(**newval**)

Modifie le mode d'activation du port du Yocto-hub.

hubport→set_logicalName(**newval**)

Modifie le nom logique du port de Yocto-hub.

hubport→set_userData(**data**)

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

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

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

hubport→get_advertisedValue() YHubPort
hubport→advertisedValue()YHubPort
get_advertisedValue

Retourne la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

```
YHubPort target get_advertisedValue
```

Retourne :
une chaîne de caractères représentant la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

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

hubport→get_baudRate() YHubPort
hubport→baudRate()YHubPort get_baudRate

Retourne la vitesse de transfert utilisée par le port de Yocto-hub, en kbps.

```
YHubPort target 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()YHubPort get_enabled

Retourne vrai si le port du Yocto-hub est alimenté, faux sinon.

```
YHubPort target 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_logicalName() YHubPort
hubport→logicalName()YHubPort get_logicalName

Retourne le nom logique du port de Yocto-hub.

```
YHubPort target 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_portState() YHubPort
hubport→portState()YHubPort get_portState

Retourne l'état actuel du port de Yocto-hub.

```
YHubPort target 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→set_enabled() YHubPort
hubport→setEnabled()YHubPort set_enabled

Modifie le mode d'activation du port du Yocto-hub.

```
YHubPort target set_enabled newval
```

Si le port est actif, il sera alimenté. Sinon, l'alimentation du module est coupée.

Paramètres :

newval soit `Y_ENABLED_FALSE`, soit `Y_ENABLED_TRUE`, selon le mode d'activation du port du Yocto-hub

Retourne :

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

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

hubport→set_logicalName() YHubPort
hubport→setLogicalName()YHubPort
set_logicalName

Modifie le nom logique du port de Yocto-hub.

```
YHubPort target set_logicalName newval
```

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

Paramètres :

newval une chaîne de caractères représentant le nom logique du port 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.

3.21. Interface de la fonction Humidity

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales
yFindHumidity(**func**) Permet de retrouver un capteur d'humidité d'après un identifiant donné.
yFirstHumidity() Commence l'énumération des capteurs d'humidité accessibles par la librairie.
Méthodes des objets `YHumidity`
humidity→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
humidity→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur d'humidité au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
humidity→get_advertisedValue() Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).
humidity→get_currentRawValue() Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en %RH, sous forme de nombre à virgule.
humidity→get_currentValue() Retourne la valeur actuelle de l'humidité, en %RH, sous forme de nombre à virgule.
humidity→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.
humidity→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.
humidity→get_friendlyName() Retourne un identifiant global du capteur d'humidité au format `NOM_MODULE.NOM_FONCTION`.
humidity→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
humidity→get_functionId() Retourne l'identifiant matériel du capteur d'humidité, sans référence au module.
humidity→get_hardwareId()

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_reportFrequency()

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

humidity→get_resolution()

Retourne la résolution des valeurs mesurées.

humidity→get_unit()

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

humidity→get_userData()

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

humidity→isOnline()

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

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

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

humidity→load(**msValidity**)

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

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

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

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

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

humidity→nextHumidity()

Continue l'énumération des capteurs d'humidité commencée à l'aide de yFirstHumidity().

humidity→registerTimedReportCallback(**callback**)

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

humidity→registerValueCallback(**callback**)

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

humidity→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

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_userData(data)

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

humidity→wait_async(callback, context)

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

humidity→calibrateFromPoints()YHumidity calibrateFromPoints YHumidity

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

YHumidity **target calibrateFromPoints rawValues refValues**

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

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

Paramètres :

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

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

Retourne :

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

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

humidity→get_advertisedValue() YHumidity
humidity→advertisedValue()YHumidity
get_advertisedValue

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

YHumidity **target** **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()YHumidity
get_currentRawValue

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

YHumidity **target 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()YHumidity get_currentValue

Retourne la valeur actuelle de l'humidité, en %RH, sous forme de nombre à virgule.

```
YHumidity target 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_highestValue() YHumidity
humidity→highestValue()YHumidity
get_highestValue

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

YHumidity **target 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()YHumidity
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.

YHumidity **target** **get_logFrequency**

Retourne :

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

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

humidity→get_logicalName() YHumidity
humidity→logicalName()YHumidity get_logicalName

Retourne le nom logique du capteur d'humidité.

YHumidity **target get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du capteur d'humidité.

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

humidity→get_lowestValue() YHumidity
humidity→lowestValue()YHumidity get_lowestValue

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

```
YHumidity target 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_recordedData() YHumidity
humidity→recordedData()YHumidity
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.

```
YHumidity target get_recordedData startTime endTime
```

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

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

Paramètres :

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

endTime la 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_reportFrequency() YHumidity
humidity→reportFrequency()YHumidity
get_reportFrequency

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

```
YHumidity target get_reportFrequency
```

Retourne :

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

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

humidity→get_resolution() YHumidity
humidity→resolution()YHumidity get_resolution

Retourne la résolution des valeurs mesurées.

```
YHumidity target get_resolution
```

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

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

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

humidity→get_unit() YHumidity
humidity→unit()YHumidity get_unit

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

```
YHumidity target get_unit
```

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

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

humidity→loadCalibrationPoints()YHumidity YHumidity loadCalibrationPoints

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

```
YHumidity target loadCalibrationPoints rawValues refValues
```

Paramètres :

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

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→set_highestValue() YHumidity
humidity→setHighestValue()YHumidity
set_highestValue

Modifie la mémoire de valeur maximale observée.

```
YHumidity target set_highestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→set_logFrequency() YHumidity
humidity→setLogFrequency()YHumidity
set_logFrequency

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
YHumidity target set_logFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→set_logicalName() YHumidity
humidity→setLogicalName()YHumidity
set_logicalName

Modifie le nom logique du capteur d'humidité.

```
YHumidity target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur 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()YHumidity
set_lowestValue

Modifie la mémoire de valeur minimale observée.

```
YHumidity target set_lowestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→set_reportFrequency() YHumidity
humidity→setReportFrequency()YHumidity
set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

```
YHumidity target set_reportFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→set_resolution() YHumidity
humidity→setResolution()YHumidity set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YHumidity target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.22. Interface de la fonction Led

La librairie de programmation Yoctopuce permet non seulement d'allumer la led à une intensité donnée, mais aussi de la faire osciller à plusieurs fréquences.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_led.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YLed = yoctolib.YLed;
php     require_once('yocto_led.php');
cpp     #include "yocto_led.h"
m       #import "yocto_led.h"
pas     uses yocto_led;
vb      yocto_led.vb
cs      yocto_led.cs
java    import com.yoctopuce.YoctoAPI.YLed;
py      from yocto_led import *
```

Fonction globales
yFindLed(**func**) Permet de retrouver une led d'après un identifiant donné.
yFirstLed() Commence l'énumération des leds accessibles par la librairie.
Méthodes des objets `YLed`
led→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de la led au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
led→get_advertisedValue() Retourne la valeur courante de la led (pas plus de 6 caractères).
led→get_blinking() Retourne le mode de signalisation de la led.
led→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la led.
led→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la led.
led→get_friendlyName() Retourne un identifiant global de la led au format `NOM_MODULE.NOM_FONCTION`.
led→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
led→get_functionId() Retourne l'identifiant matériel de la led, sans référence au module.
led→get_hardwareId() Retourne l'identifiant matériel unique de la led au format `SERIAL.FUNCTIONID`.
led→get_logicalName() Retourne le nom logique de la led.
led→get_luminosity() Retourne l'intensité de la led en pour cent.
led→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

led→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

led→get_power()

Retourne l'état courant de la led.

led→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

led→isOnline()

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

led→isOnline_async(callback, context)

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

led→load(msValidity)

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

led→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

led→nextLed()

Continue l'énumération des leds commencée à l'aide de `yFirstLed()`.

led→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

led→set_blinking(newval)

Modifie le mode de signalisation de la led.

led→set_logicalName(newval)

Modifie le nom logique de la led.

led→set_luminosity(newval)

Modifie l'intensité lumineuse de la led (en pour cent).

led→set_power(newval)

Modifie l'état courant de la led.

led→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

led→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

led→get_advertisedValue() YLed
led→advertisedValue()YLed get_advertisedValue

Retourne la valeur courante de la led (pas plus de 6 caractères).

```
YLed target get_advertisedValue
```

Retourne :
une chaîne de caractères représentant la valeur courante de la led (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

led→get_blinking() YLed
led→blinking()YLed get_blinking

Retourne le mode de signalisation de la led.

```
YLed target get_blinking
```

Retourne :

une valeur parmi `Y_BLINKING_STILL`, `Y_BLINKING_RELAX`, `Y_BLINKING_AWARE`, `Y_BLINKING_RUN`, `Y_BLINKING_CALL` et `Y_BLINKING_PANIC` représentant le mode de signalisation de la led

En cas d'erreur, déclenche une exception ou retourne `Y_BLINKING_INVALID`.

led→get_logicalName() YLed
led→logicalName()YLed get_logicalName

Retourne le nom logique de la led.

```
YLed target get_logicalName
```

Retourne :
une chaîne de caractères représentant le nom logique de la led.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

led→get_luminosity() YLed
led→luminosity()YLed get_luminosity

Retourne l'intensité de la led en pour cent.

```
YLed target 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_power() YLed
led→power()YLed get_power

Retourne l'état courant de la led.

```
YLed target get_power
```

Retourne :

soit `Y_POWER_OFF`, soit `Y_POWER_ON`, selon l'état courant de la led

En cas d'erreur, déclenche une exception ou retourne `Y_POWER_INVALID`.

led→set_blinking() YLed
led→setBlinking()YLed set_blinking

Modifie le mode de signalisation de la led.

YLed **target** **set_blinking** **newval**

Paramètres :

newval une valeur parmi `Y_BLINKING_STILL`, `Y_BLINKING_RELAX`, `Y_BLINKING_AWARE`, `Y_BLINKING_RUN`, `Y_BLINKING_CALL` et `Y_BLINKING_PANIC` représentant le mode de signalisation de la led

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

led→set_logicalName() YLed led→setLogicalName()YLed set_logicalName

Modifie le nom logique de la led.

```
YLed target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de la led.

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()YLed set_luminosity

Modifie l'intensité lumineuse de la led (en pour cent).

YLed **target** **set_luminosity** **newval**

Paramètres :

newval un entier représentant l'intensité lumineuse de la led (en pour cent)

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

led→set_power() YLed
led→setPower()YLed set_power

Modifie l'état courant de la led.

```
YLed target set_power newval
```

Paramètres :

newval soit `Y_POWER_OFF`, soit `Y_POWER_ON`, selon l'état courant de la led

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.23. Interface de la fonction LightSensor

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_lightsensor.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YLightSensor = yoctolib.YLightSensor;
php     require_once('yocto_lightsensor.php');
cpp     #include "yocto_lightsensor.h"
m       #import "yocto_lightsensor.h"
pas     uses yocto_lightsensor;
vb      yocto_lightsensor.vb
cs      yocto_lightsensor.cs
java    import com.yoctopuce.YoctoAPI.YLightSensor;
py      from yocto_lightsensor import *
```

Fonction globales
yFindLightSensor(**func**) Permet de retrouver un capteur de lumière d'après un identifiant donné.
yFirstLightSensor() Commence l'énumération des capteurs de lumière accessibles par la librairie.
Méthodes des objets `YLightSensor`
lightsensor→calibrate(**calibratedVal**) Modifie le paramètre de calibration spécifique du senseur de sorte à ce que la valeur actuelle corresponde à une consigne donnée (correction linéaire).
lightsensor→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
lightsensor→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de lumière au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
lightsensor→get_advertisedValue() Retourne la valeur courante du capteur de lumière (pas plus de 6 caractères).
lightsensor→get_currentRawValue() Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en lux, sous forme de nombre à virgule.
lightsensor→get_currentValue() Retourne la valeur actuelle de la lumière ambiante, en lux, sous forme de nombre à virgule.
lightsensor→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.
lightsensor→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.
lightsensor→get_friendlyName() Retourne un identifiant global du capteur de lumière au format `NOM_MODULE.NOM_FONCTION`.
lightsensor→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

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_unit()

Retourne l'unité dans laquelle la lumière ambiante est exprimée.

lightsensor→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

lightsensor→isOnline()

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

lightsensor→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

lightsensor→load(**msValidity**)

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

lightsensor→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

lightsensor→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

lightsensor→nextLightSensor()

Continue l'énumération des capteurs de lumière commencée à l'aide de `yFirstLightSensor()`.

lightsensor→registerTimedReportCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

lightsensor→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

lightsensor→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

lightsensor→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

lightsensor→set_logicalName(newval)

Modifie le nom logique du capteur de lumière.

lightsensor→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

lightsensor→set_measureType(newval)

Change le type dde mesure de lumière effectuée par le capteur.

lightsensor→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

lightsensor→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

lightsensor→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

lightsensor→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

lightsensor→calibrate()YLightSensor calibrate YLightSensor

Modifie le paramètre de calibration spécifique du senseur de sorte à ce que la valeur actuelle corresponde à une consigne donnée (correction linéaire).

YLightSensor **target calibrate calibratedVal**

Paramètres :

calibratedVal la consigne de valeur désirée.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→calibrateFromPoints()YLightSensor YLightSensor calibrateFromPoints

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

YLightSensor **target calibrateFromPoints rawValues refValues**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→get_advertisedValue() YLightSensor
lightsensor→advertisedValue()YLightSensor
get_advertisedValue

Retourne la valeur courante du capteur de lumière (pas plus de 6 caractères).

```
YLightSensor target get_advertisedValue
```

Retourne :
une chaîne de caractères représentant la valeur courante du capteur de lumière (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

lightsensor→get_currentRawValue() YLightSensor
lightsensor→currentRawValue()YLightSensor
get_currentRawValue

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en lux, sous forme de nombre à virgule.

YLightSensor **target** **get_currentRawValue**

Retourne :
une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en lux, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

lightsensor→get_currentValue() YLightSensor
lightsensor→currentValue()YLightSensor
get_currentValue

Retourne la valeur actuelle de la lumière ambiante, en lux, sous forme de nombre à virgule.

YLightSensor **target** **get_currentValue**

Retourne :
une valeur numérique représentant la valeur actuelle de la lumière ambiante, en lux, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

lightsensor→get_highestValue() YLightSensor
lightsensor→highestValue()YLightSensor
get_highestValue

Retourne la valeur maximale observée pour la lumière ambiante depuis le démarrage du module.

YLightSensor **target** **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()YLightSensor
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.

YLightSensor **target** **get_logFrequency**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

lightsensor→get_logicalName() YLightSensor
lightsensor→logicalName()YLightSensor
get_logicalName

Retourne le nom logique du capteur de lumière.

```
YLightSensor target get_logicalName
```

Retourne :
une chaîne de caractères représentant le nom logique du capteur de lumière.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

lightsensor→get_lowestValue() YLightSensor
lightsensor→lowestValue()YLightSensor
get_lowestValue

Retourne la valeur minimale observée pour la lumière ambiante depuis le démarrage du module.

YLightSensor **target** **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()YLightSensor
get_measureType

Retourne le type de mesure de lumière utilisé par le module.

YLightSensor **target** **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_recordedData() YLightSensor
lightsensor→recordedData()YLightSensor
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.

```
YLightSensor target get_recordedData startTime endTime
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la 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()YLightSensor
get_reportFrequency

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

YLightSensor **target** **get_reportFrequency**

Retourne :
une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

lightsensor→get_resolution() YLightSensor
lightsensor→resolution()YLightSensor
get_resolution

Retourne la résolution des valeurs mesurées.

YLightSensor **target** **get_resolution**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

lightsensor→get_unit() YLightSensor
lightsensor→unit()YLightSensor get_unit

Retourne l'unité dans laquelle la lumière ambiante est exprimée.

```
YLightSensor target 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→loadCalibrationPoints()YLightSensor YLightSensor loadCalibrationPoints

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

YLightSensor **target loadCalibrationPoints rawValues refValues**

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_highestValue() YLightSensor
lightsensor→setHighestValue()YLightSensor
set_highestValue

Modifie la mémoire de valeur maximale observée.

```
YLightSensor target set_highestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_logFrequency() YLightSensor
lightsensor→setLogFrequency()YLightSensor
set_logFrequency

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

YLightSensor **target set_logFrequency newval**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_logicalName() YLightSensor
lightsensor→setLogicalName()YLightSensor
set_logicalName

Modifie le nom logique du capteur de lumière.

```
YLightSensor target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de lumière.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_lowestValue() YLightSensor
lightsensor→setLowestValue()YLightSensor
set_lowestValue

Modifie la mémoire de valeur minimale observée.

```
YLightSensor target set_lowestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_measureType() YLightSensor
lightsensor→setMeasureType()YLightSensor
set_measureType

Change le type dde mesure de lumière effectuée par le capteur.

```
YLightSensor target 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()YLightSensor set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

```
YLightSensor target set_reportFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_resolution() YLightSensor
lightsensor→setResolution()YLightSensor
set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YLightSensor target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.24. Interface de la fonction Magnetometer

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_magnetometer.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YMagnetometer = yoctolib.YMagnetometer;
php     require_once('yocto_magnetometer.php');
cpp     #include "yocto_magnetometer.h"
m       #import "yocto_magnetometer.h"
pas     uses yocto_magnetometer;
vb      yocto_magnetometer.vb
cs      yocto_magnetometer.cs
java    import com.yoctopuce.YoctoAPI.YMagnetometer;
py      from yocto_magnetometer import *
```

Fonction globales

yFindMagnetometer(**func**)

Permet de retrouver un magnétomètre d'après un identifiant donné.

yFirstMagnetometer()

Commence l'énumération des magnétomètres accessibles par la librairie.

Méthodes des objets `YMagnetometer`

magnetometer→calibrateFromPoints(**rawValues**, **refValues**)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

magnetometer→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du magnétomètre au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

magnetometer→get_advertisedValue()

Retourne la valeur courante du magnétomètre (pas plus de 6 caractères).

magnetometer→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en mT, sous forme de nombre à virgule.

magnetometer→get_currentValue()

Retourne la valeur actuelle du champ magnétique, en mT, sous forme de nombre à virgule.

magnetometer→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

magnetometer→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

magnetometer→get_friendlyName()

Retourne un identifiant global du magnétomètre au format `NOM_MODULE.NOM_FONCTION`.

magnetometer→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

magnetometer→get_functionId()

Retourne l'identifiant matériel du magnétomètre, sans référence au module.

magnetometer→get_hardwareId()

Retourne l'identifiant matériel unique du magnétomètre au format SERIAL.FUNCTIONID.

magnetometer→get_highestValue()

Retourne la valeur maximale observée pour le champ magnétique depuis le démarrage du module.

magnetometer→get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

magnetometer→get_logicalName()

Retourne le nom logique du magnétomètre.

magnetometer→get_lowestValue()

Retourne la valeur minimale observée pour le champ magnétique depuis le démarrage du module.

magnetometer→get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

magnetometer→get_module_async(**callback**, **context**)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

magnetometer→get_recordedData(**startTime**, **endTime**)

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

magnetometer→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

magnetometer→get_resolution()

Retourne la résolution des valeurs mesurées.

magnetometer→get_unit()

Retourne l'unité dans laquelle le champ magnétique est exprimée.

magnetometer→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

magnetometer→get_xValue()

Retourne la composante X du champ magnétique, sous forme de nombre à virgule.

magnetometer→get_yValue()

Retourne la composante Y du champ magnétique, sous forme de nombre à virgule.

magnetometer→get_zValue()

Retourne la composante Z du champ magnétique, sous forme de nombre à virgule.

magnetometer→isOnline()

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

magnetometer→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

magnetometer→load(**msValidity**)

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

magnetometer→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

magnetometer→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

magnetometer→nextMagnetometer()

Continue l'énumération des magnétomètres commencée à l'aide de yFirstMagnetometer().

magnetometer→registerTimedReportCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

magnetometer→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

magnetometer→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

magnetometer→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

magnetometer→set_logicalName(newval)

Modifie le nom logique du magnétomètre.

magnetometer→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

magnetometer→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

magnetometer→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

magnetometer→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

magnetometer→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

magnetometer→calibrateFromPoints() YMagnetometer
YMagnetometer calibrateFromPoints

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

YMagnetometer **target calibrateFromPoints rawValues refValues**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→get_advertisedValue() YMagnetometer magnetometer→advertisedValue()YMagnetometer get_advertisedValue

Retourne la valeur courante du magnétomètre (pas plus de 6 caractères).

YMagnetometer **target get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante du magnétomètre (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

magnetometer→get_currentRawValue() YMagnetometer
magnetometer→currentRawValue()YMagnetometer
get_currentRawValue

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en mT, sous forme de nombre à virgule.

YMagnetometer **target** **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()YMagnetometer
get_currentValue

Retourne la valeur actuelle du champ magnétique, en mT, sous forme de nombre à virgule.

YMagnetometer **target** **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_highestValue() YMagnetometer
magnetometer→highestValue()YMagnetometer
get_highestValue

Retourne la valeur maximale observée pour le champ magnétique depuis le démarrage du module.

YMagnetometer **target** **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()YMagnetometer
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.

YMagnetometer **target** **get_logFrequency**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

magnetometer→get_logicalName() YMagnetometer
magnetometer→logicalName()YMagnetometer
get_logicalName

Retourne le nom logique du magnétomètre.

YMagnetometer **target** **get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du magnétomètre.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

magnetometer→get_lowestValue() YMagnetometer
magnetometer→lowestValue()YMagnetometer
get_lowestValue

Retourne la valeur minimale observée pour le champ magnétique depuis le démarrage du module.

YMagnetometer **target** **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_recordedData() YMagnetometer
magnetometer→recordedData()YMagnetometer
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.

YMagnetometer **target get_recordedData startTime endTime**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la 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()YMagnetometer
get_reportFrequency

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
YMagnetometer target get_reportFrequency
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

magnetometer→get_resolution() YMagnetometer
magnetometer→resolution()YMagnetometer
get_resolution

Retourne la résolution des valeurs mesurées.

```
YMagnetometer target get_resolution
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

magnetometer→get_unit() YMagnetometer
magnetometer→unit()YMagnetometer get_unit

Retourne l'unité dans laquelle le champ magnétique est exprimée.

YMagnetometer **target** **get_unit**

Retourne :
une chaîne de caractères représentant l'unité dans laquelle le champ magnétique est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

magnetometer→get_xValue() YMagnetometer
magnetometer→xValue()YMagnetometer get_xValue

Retourne la composante X du champ magnétique, sous forme de nombre à virgule.

YMagnetometer **target** **get_xValue**

Retourne :

une valeur numérique représentant la composante X du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_XVALUE_INVALID`.

magnetometer→get_yValue() YMagnetometer magnetometer→yValue()YMagnetometer get_yValue

Retourne la composante Y du champ magnétique, sous forme de nombre à virgule.

```
YMagnetometer target get_yValue
```

Retourne :

une valeur numérique représentant la composante Y du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_YVALUE_INVALID`.

magnetometer→get_zValue() YMagnetometer
magnetometer→zValue()YMagnetometer get_zValue

Retourne la composante Z du champ magnétique, sous forme de nombre à virgule.

YMagnetometer **target** **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→loadCalibrationPoints() YMagnetometer
YMagnetometer loadCalibrationPoints

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

YMagnetometer **target loadCalibrationPoints rawValues refValues**

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→set_highestValue() YMagnetometer
magnetometer→setHighestValue()YMagnetometer
set_highestValue

Modifie la mémoire de valeur maximale observée.

YMagnetometer **target** **set_highestValue** **newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→set_logFrequency() YMagnetometer
magnetometer→setLogFrequency()YMagnetometer
set_logFrequency

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

YMagnetometer **target set_logFrequency newval**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→set_logicalName() YMagnetometer
magnetometer→setLogicalName()YMagnetometer
set_logicalName

Modifie le nom logique du magnétomètre.

YMagnetometer **target set_logicalName newval**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du magnétomètre.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→set_lowestValue() YMagnetometer magnetometer→setLowestValue()YMagnetometer set_lowestValue

Modifie la mémoire de valeur minimale observée.

```
YMagnetometer target set_lowestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→set_reportFrequency() YMagnetometer
magnetometer→setReportFrequency()
YMagnetometer set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

```
YMagnetometer target set_reportFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→set_resolution() YMagnetometer magnetometer→setResolution()YMagnetometer set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YMagnetometer target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.25. Valeur mesurée

Les objets YMeasure sont utilisés dans l'interface de programmation Yoctopuce pour représenter une valeur observée un moment donnée. Ces objets sont utilisés en particulier en conjonction avec la classe YDataSet.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_api.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YAPI = yoctolib.YAPI;
        var YModule = yoctolib.YModule;
php     require_once('yocto_api.php');
cpp     #include "yocto_api.h"
m       #import "yocto_api.h"
pas     uses yocto_api;
vb      yocto_api.vb
cs      yocto_api.cs
java    import com.yoctopuce.YoctoAPI.YModule;
py      from yocto_api import *
```

Méthodes des objets `YMeasure`
measure→get_averageValue() Retourne la valeur moyenne observée durant l'intervalle de temps couvert par la mesure.
measure→get_endTimeUTC() Retourne l'heure absolue de la fin de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).
measure→get_maxValue() Retourne la plus grande valeur observée durant l'intervalle de temps couvert par la mesure.
measure→get_minValue() Retourne la plus petite valeur observée durant l'intervalle de temps couvert par la mesure.
measure→get_startTimeUTC() Retourne l'heure absolue du début de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

3.26. Interface de contrôle du module

Cette interface est la même pour tous les modules USB de Yoctopuce. Elle permet de contrôler les paramètres généraux du module, et d'énumérer les fonctions fournies par chaque module.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_api.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YAPI = yoctolib.YAPI;
        var YModule = yoctolib.YModule;
php     require_once('yocto_api.php');
cpp     #include "yocto_api.h"
m       #import "yocto_api.h"
pas     uses yocto_api;
vb      yocto_api.vb
cs      yocto_api.cs
java    import com.yoctopuce.YoctoAPI.YModule;
py      from yocto_api import *
```

Fonction globales
yFindModule(**func**) Permet de retrouver un module d'après son numéro de série ou son nom logique.
yFirstModule() Commence l'énumération des modules accessibles par la librairie.
Méthodes des objets `YModule`
module→checkFirmware(**path**, **onlynew**) Test si le fichié byn est valid pour le module.
module→describe() Retourne un court texte décrivant le module.
module→download(**pathname**) Télécharge le fichier choisi du module et retourne son contenu.
module→functionCount() Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.
module→functionId(**functionIndex**) Retourne l'identifiant matériel de la *n*ième fonction du module.
module→functionName(**functionIndex**) Retourne le nom logique de la *n*ième fonction du module.
module→functionValue(**functionIndex**) Retourne la valeur publiée par la *n*ième fonction du module.
module→get_allSettings() Retourne tous les paramètres du module.
module→get_beacon() Retourne l'état de la balise de localisation.
module→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.
module→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.
module→get_firmwareRelease()

Retourne la version du logiciel embarqué du module.

module→get_hardwareId()
Retourne l'identifiant unique du module.

module→get_icon2d()
Retourne l'icône du module.

module→get_lastLogs()
Retourne une 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_persistentSettings()
Retourne l'état courant des réglages persistents du module.

module→get_productId()
Retourne l'identifiant USB du module, préprogrammé en usine.

module→get_productName()
Retourne le nom commercial du module, préprogrammé en usine.

module→get_productRelease()
Retourne le numéro de version matériel du module, préprogrammé en usine.

module→get_rebootCountdown()
Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

module→get_serialNumber()
Retourne le numéro de série du module, préprogrammé en usine.

module→get_upTime()
Retourne le numbre de millisecondes écoulées depuis la mise sous tension du 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→isOnline()
Vérifie si le module est joignable, sans déclencher d'erreur.

module→isOnline_async(**callback**, **context**)
Vérifie si le module est joignable, sans déclencher d'erreur.

module→load(**msValidity**)
Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

module→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

module→nextModule()
Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

module→reboot(**secBeforeReboot**)
Agende un simple redémarrage du module dans un nombre donné de secondes.

module→registerLogCallback(**callback**)
Enregistre une fonction de callback qui sera appelée à chaque fois le module émet un message de log.

module→revertFromFlash()

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

module→saveToFlash()

Sauve les réglages courants dans la mémoire non volatile du module.

module→set_allSettings(**settings**)

Restore tous les paramètres du module.

module→set_beacon(**newval**)

Allume ou éteint la balise de localisation du module.

module→set_logicalName(**newval**)

Change le nom logique du module.

module→set_luminosity(**newval**)

Modifie la luminosité des leds informatives du module.

module→set_userData(**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**)

Retourne la valeur entière précédemment stockée dans cet attribut.

module→triggerFirmwareUpdate(**secBeforeReboot**)

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

module→updateFirmware(**path**)

Prepare une mise à jour de firmware du module.

module→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

module→checkFirmware()YModule checkFirmware YModule

Test si le fichié byn est valid pour le module.

YModule **target checkFirmware path onlynew**

Cette methode est utile pour tester si il est nécessaire de mettre à jour le module avec un nouveau firmware. Il est possible de passer un répertoire qui contiens plusieurs fichier byn. Dans ce cas cette methode retourne le path du fichier byn compatible le plus récent. Si le parametre onlynew est vrais les firmware équivalent ou plus ancien au firmware installé sont ignorés.

Paramètres :

path le path sur un fichier byn ou un répertoire contenant plusieurs fichier byn

onlynew retourne uniquement les fichier strictement plus récent

Retourne :

: le path du fichier byn a utiliser ou une chaine vide si aucun firmware plus récent est disponible En cas d'erreur, déclenche une exception ou retourne une chaine de caractère qui comment par "error:".

module→download()YModule download — YModule

Télécharge le fichier choisi du module et retourne son contenu.

```
YModule target download pathname
```

Paramètres :

pathname nom complet du fichier

Retourne :

le contenu du fichier chargé

En cas d'erreur, déclenche une exception ou retourne `YAPI_INVALID_STRING`.

module→get_allSettings() YModule
module→allSettings()YModule get_allSettings

Retourne tous les paramètres du module.

YModule **target get_allSettings**

Utile pour sauvgarder les noms logiques et les calibrations du module.

Retourne :
un buffer binaire avec touts les paramètres En cas d'erreur, déclenche une exception ou retourne `YAPI_INVALID_STRING`.

module→get_beacon() YModule
module→beacon()YModule get_beacon

Retourne l'état de la balise de localisation.

YModule **target** **get_beacon**

Retourne :

soit `Y_BEACON_OFF`, soit `Y_BEACON_ON`, selon l'état de la balise de localisation

En cas d'erreur, déclenche une exception ou retourne `Y_BEACON_INVALID`.

module→get_firmwareRelease() YModule
module→firmwareRelease()YModule
get_firmwareRelease

Retourne la version du logiciel embarqué du module.

YModule **target get_firmwareRelease**

Retourne :
une chaîne de caractères représentant la version du logiciel embarqué du module

En cas d'erreur, déclenche une exception ou retourne `Y_FIRMWARERELEASE_INVALID`.

module→get_icon2d() YModule
module→icon2d()YModule get_icon2d

Retourne l'icône du module.

```
YModule target 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()YModule get_lastLogs

Retourne une chaine de charactère contenant les derniers logs du module.

YModule **target get_lastLogs**

Cette methode retourne les derniers logs qui sont encore stocké dans le module.

Retourne :
une chaine de charactère contenant les derniers logs du module. En cas d'erreur, déclenche une exception ou retourne `YAPI_INVALID_STRING`.

module→get_logicalName() YModule
module→logicalName()YModule get_logicalName

Retourne le nom logique du module.

YModule **target** **get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

module→get_luminosity() YModule
module→luminosity()YModule get_luminosity

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

YModule **target 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()YModule
get_persistentSettings

Retourne l'état courant des réglages persistents du module.

YModule **target** **get_persistentSettings**

Retourne :

une valeur parmi `Y_PERSISTENTSETTINGS_LOADED`, `Y_PERSISTENTSETTINGS_SAVED` et `Y_PERSISTENTSETTINGS_MODIFIED` représentant l'état courant des réglages persistents du module

En cas d'erreur, déclenche une exception ou retourne `Y_PERSISTENTSETTINGS_INVALID`.

module→get_productId() YModule
module→productId()YModule get_productId

Retourne l'identifiant USB du module, préprogrammé en usine.

YModule **target 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()YModule get_productName

Retourne le nom commercial du module, préprogrammé en usine.

YModule **target** **get_productName**

Retourne :
une chaîne de caractères représentant le nom commercial du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne `Y_PRODUCTNAME_INVALID`.

module→get_productRelease() YModule
module→productRelease()YModule
get_productRelease

Retourne le numéro de version matériel du module, préprogrammé en usine.

YModule **target 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()YModule
get_rebootCountdown

Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

YModule **target 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()YModule get_serialNumber

Retourne le numéro de série du module, préprogrammé en usine.

YModule **target get_serialNumber**

Retourne :
une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne `Y_SERIALNUMBER_INVALID`.

module→get_upTime() YModule
module→upTime()YModule get_upTime

Retourne le numbre de millisecondes écoulées depuis la mise sous tension du module

YModule **target** **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()YModule get_usbCurrent

Retourne le courant consommé par le module sur le bus USB, en milliampères.

YModule **target get_usbCurrent**

Retourne :
un entier représentant le courant consommé par le module sur le bus USB, en milliampères

En cas d'erreur, déclenche une exception ou retourne `Y_USBCURRENT_INVALID`.

module→get_userVar() YModule
module→userVar()YModule get_userVar

Retourne la valeur entière précédemment stockée dans cet attribut.

```
YModule target 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→reboot()YModule reboot — YModule

Agende un simple redémarrage du module dans un nombre donné de secondes.

```
YModule target reboot secBeforeReboot
```

Paramètres :

secBeforeReboot nombre de secondes avant de redémarrer

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→revertFromFlash()YModule YModule revertFromFlash

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

YModule **target revertFromFlash**

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→saveToFlash()YModule saveToFlash YModule

Sauve les réglages courants dans la mémoire non volatile du module.

```
YModule target 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()YModule set_allSettings

Restore tous les paramètres du module.

```
YModule target set_allSettings settings
```

Utile pour restorer les noms logiques et les calibrations du module depuis un sauvgarde.

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()YModule set_beacon

Allume ou éteint la balise de localisation du module.

YModule **target 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()YModule set_logicalName

Change le nom logique du module.

YModule **target set_logicalName newval**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→set_luminosity() YModule
module→setLuminosity()YModule set_luminosity

Modifie la luminosité des leds informatives du module.

YModule **target 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_userVar() YModule
module→setUserVar()YModule set_userVar

Retourne la valeur entière précédemment stockée dans cet attribut.

```
YModule target set_userVar newval
```

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 triggerFirmwareUpdate

YModule

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

YModule **target triggerFirmwareUpdate secBeforeReboot**

Paramètres :

secBeforeReboot nombre de secondes avant de redémarrer

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→updateFirmware()YModule updateFirmware YModule

Prepare une mise à jour de firmware du module.

```
YModule target updateFirmware path
```

Cette methode un object `YFirmwareUpdate` qui est utilisé pour mettre à jour le firmware du module.

Paramètres :

path le path sur un fichier byn

Retourne :

: Un object `YFirmwareUpdate`

3.27. Interface de la fonction Motor

La librairie de programmation yoctopuce permet de piloter la puissance envoyée au moteur pour le faire tourner aussi bien dans un sens que dans l'autre, mais aussi de piloter des accélérations linéaires: le moteur 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>
nodejs  var yoctolib = require('yoctolib');
        var YMotor = yoctolib.YMotor;
php     require_once('yocto_motor.php');
cpp     #include "yocto_motor.h"
m       #import "yocto_motor.h"
pas     uses yocto_motor;
vb      yocto_motor.vb
cs      yocto_motor.cs
java    import com.yoctopuce.YoctoAPI.YMotor;
py      from yocto_motor import *
```

Fonction globales
yFindMotor(**func**) Permet de retrouver un moteur d'après un identifiant donné.
yFirstMotor() Commence l'énumération des moteur accessibles par la librairie.
Méthodes des objets `YMotor`
motor→brakingForceMove(**targetPower**, **delay**) Modifie progressivement la force de freinage appliquée au moteur sur une durée donnée.
motor→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du moteur au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
motor→drivingForceMove(**targetPower**, **delay**) Modifie progressivement la puissance envoyée au moteur sur une durée donnée.
motor→get_advertisedValue() Retourne la valeur courante du moteur (pas plus de 6 caractères).
motor→get_brakingForce() Retourne la force de freinage appliquée au moteur, sous forme de pourcentage.
motor→get_cutOffVoltage() Retourne la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.
motor→get_drivingForce() Retourne la puissance actuelle envoyée au moteur, sous forme de nombre réel entre -100% et +100%.
motor→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moteur.
motor→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moteur.
motor→get_failSafeTimeout()

Retourne le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

motor→get_frequency()

Retourne la fréquence du signal PWM utilisé pour contrôler le moteur.

motor→get_friendlyName()

Retourne un identifiant global du moteur au format `NOM_MODULE.NOM_FONCTION`.

motor→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

motor→get_functionId()

Retourne l'identifiant matériel du moteur, sans référence au module.

motor→get_hardwareId()

Retourne l'identifiant matériel unique du moteur au format `SERIAL.FUNCTIONID`.

motor→get_logicalName()

Retourne le nom logique du moteur.

motor→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

motor→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

motor→get_motorStatus()

Retourne l'état du contrôleur de moteur.

motor→get_overCurrentLimit()

Retourne la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

motor→get_starterTime()

Retourne la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

motor→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

motor→isOnline()

Vérifie si le module hébergeant le moteur est joignable, sans déclencher d'erreur.

motor→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le moteur est joignable, sans déclencher d'erreur.

motor→keepALive()

Réarme la sécurité failsafe du contrôleur.

motor→load(**msValidity**)

Met en cache les valeurs courantes du moteur, avec une durée de validité spécifiée.

motor→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du moteur, avec une durée de validité spécifiée.

motor→nextMotor()

Continue l'énumération des moteur commencée à l'aide de `yFirstMotor()`.

motor→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

motor→resetStatus()

Réinitialise l'état du contrôleur à IDLE.

motor→set_brakingForce(**newval**)

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→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.

motor→brakingForceMove()YMotor brakingForceMove — YMotor

Modifie progressivement la force de freinage appliquée au moteur sur une durée donnée.

YMotor **target 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→drivingForceMove()YMotor drivingForceMove

YMotor

Modifie progressivement la puissance envoyée au moteur sur une durée donnée.

YMotor **target 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()YMotor
get_advertisedValue

Retourne la valeur courante du moteur (pas plus de 6 caractères).

```
YMotor target 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()YMotor get_brakingForce

Retourne la force de freinage appliquée au moteur, sous forme de pourcentage.

```
YMotor target 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()YMotor get_cutOffVoltage

Retourne la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

YMotor **target 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()YMotor get_drivingForce

Retourne la puissance actuelle envoyée au moteur, sous forme de nombre réel entre -100% et +100%.

YMotor **target** **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_failSafeTimeout() YMotor
motor→failSafeTimeout()YMotor get_failSafeTimeout

Retourne le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

```
YMotor target 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()YMotor get_frequency

Retourne la fréquence du signal PWM utilisé pour contrôler le moteur.

YMotor **target** **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_logicalName() YMotor
motor→logicalName()YMotor get_logicalName

Retourne le nom logique du moteur.

YMotor **target** **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_motorStatus() YMotor
motor→motorStatus()YMotor get_motorStatus

Retourne l'état du contrôleur de moteur.

YMotor **target** **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()YMotor
get_overCurrentLimit

Retourne la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

YMotor **target** **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()YMotor get_starterTime

Retourne la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

YMotor **target** **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→keepALive()YMotor keepALive YMotor

Réarme la sécurité failsafe du contrôleur.

```
YMotor target 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→resetStatus()YMotor resetStatus YMotor

Réinitialise l'état du contrôleur à IDLE.

```
YMotor target resetStatus
```

Cette fonction doit être explicitement appelée après toute condition d'erreur pour permettre au contrôleur de repartir.

motor→set_brakingForce() YMotor
motor→setBrakingForce()YMotor set_brakingForce

Modifie immédiatement la force de freinage appliquée au moteur (en pourcents).

YMotor **target 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()YMotor set_cutOffVoltage

Modifie la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

```
YMotor target 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()YMotor set_drivingForce

Modifie immédiatement la puissance envoyée au moteur.

```
YMotor target 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()YMotor set_failSafeTimeout

Modifie le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

```
YMotor target 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()YMotor set_frequency

Modifie la fréquence du signal PWM utilisée pour contrôler le moteur.

```
YMotor target 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()YMotor set_logicalName

Modifie le nom logique du moteur.

```
YMotor target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du 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()YMotor
set_overCurrentLimit

Modifie la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

```
YMotor target 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()YMotor set_starterTime

Modifie la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

YMotor **target 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.

3.28. Interface de la fonction Network

Les objets YNetwork permettent de contrôler les paramètres TCP/IP des modules Yoctopuce dotés d'une interface réseau.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_network.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YNetwork = yoctolib.YNetwork;
php     require_once('yocto_network.php');
cpp     #include "yocto_network.h"
m       #import "yocto_network.h"
pas     uses yocto_network;
vb      yocto_network.vb
cs      yocto_network.cs
java    import com.yoctopuce.YoctoAPI.YNetwork;
py      from yocto_network import *
```

Fonction globales
yFindNetwork(**func**) Permet de retrouver une interface réseau d'après un identifiant donné.
yFirstNetwork() Commence l'énumération des interfaces réseau accessibles par la librairie.
Méthodes des objets `YNetwork`
network→callbackLogin(**username**, **password**) Contacte le callback de notification et sauvegarde un laisser-passer pour s'y connecter.
network→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
network→get_adminPassword() Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide.
network→get_advertisedValue() Retourne la valeur courante de l'interface réseau (pas plus de 6 caractères).
network→get_callbackCredentials() Retourne une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide.
network→get_callbackEncoding() Retourne l'encodage à utiliser pour représenter les valeurs notifiées par callback.
network→get_callbackMaxDelay() Retourne l'attente maximale entre deux notifications par callback, en secondes.
network→get_callbackMethod() Retourne la méthode HTTP à utiliser pour signaler les changements d'état par callback.
network→get_callbackMinDelay() Retourne l'attente minimale entre deux notifications par callback, en secondes.
network→get_callbackUrl() Retourne l'adresse (URL) de callback à notifier lors de changement d'état significatifs.
network→get_discoverable()

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (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_ipAddress()

Retourne l'adresse IP utilisée par le module Yoctopuce.

network→get_logicalName()

Retourne le nom logique de l'interface réseau.

network→get_macAddress()

Retourne l'adresse MAC de l'interface réseau, unique pour chaque module.

network→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

network→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

network→get_poeCurrent()

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

network→get_primaryDNS()

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

network→get_readiness()

Retourne l'état de fonctionnement atteint par l'interface réseau.

network→get_router()

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

network→get_secondaryDNS()

Retourne l'adresse IP du serveur de noms secondaire que le module doit utiliser.

network→get_subnetMask()

Retourne le masque de sous-réseau utilisé par le module.

network→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

network→get_userPassword()

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

network→get_wwwWatchdogDelay()

Retourne la durée de perte de connection WWW tolérée (en secondes) avant de 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→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

network→nextNetwork()

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

network→ping(host)

Ping str_host pour vérifier la connexion réseau.

network→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

network→set_adminPassword(newval)

Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.

network→set_callbackCredentials(newval)

Modifie le laisser-passer pour se connecter à l'adresse de callback.

network→set_callbackEncoding(newval)

Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.

network→set_callbackMaxDelay(newval)

Modifie l'attente maximale entre deux notifications par callback, en secondes.

network→set_callbackMethod(newval)

Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.

network→set_callbackMinDelay(newval)

Modifie l'attente minimale entre deux notifications par callback, en secondes.

network→set_callbackUrl(newval)

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

network→set_discoverable(newval)

Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

network→set_logicalName(newval)

Modifie le nom logique de l'interface réseau.

network→set_primaryDNS(newval)

Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.

network→set_secondaryDNS(newval)

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

network→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

network→set_userPassword(newval)

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

network→set_wwwWatchdogDelay(newval)

Modifie la durée de perte de connection WWW tolérée (en secondes) avant de déclancher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

network→useDHCP(fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

network→useStaticIP(ipAddress, subnetMaskLen, router)

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

network→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

network→callbackLogin()YNetwork callbackLogin YNetwork

Contacte le callback de notification et sauvegarde un laisser-passer pour s'y connecter.

YNetwork **target callbackLogin username password**

Le mot de passe ne sera pas stocké dans le module, mais seulement une version hashée non réversible. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

username nom d'utilisateur pour s'identifier au callback

password mot de passe pour s'identifier au callback

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→get_adminPassword() YNetwork
network→adminPassword()YNetwork
get_adminPassword

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide.

YNetwork **target** **get_adminPassword**

Retourne :

une chaîne de caractères représentant une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne `Y_ADMINPASSWORD_INVALID`.

network→get_advertisedValue() YNetwork
network→advertisedValue()YNetwork
get_advertisedValue

Retourne la valeur courante de l'interface réseau (pas plus de 6 caractères).

YNetwork **target** **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()YNetwork
get_callbackCredentials

Retourne une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide.

YNetwork **target** **get_callbackCredentials**

Retourne :
une chaîne de caractères représentant une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKCREDENTIALS_INVALID`.

network→get_callbackEncoding() YNetwork
network→callbackEncoding()YNetwork
get_callbackEncoding

Retourne l'encodage à utiliser pour représenter les valeurs notifiées par callback.

YNetwork **target** **get_callbackEncoding**

Retourne :

une valeur parmi `Y_CALLBACKENCODING_FORM`, `Y_CALLBACKENCODING_JSON`, `Y_CALLBACKENCODING_JSON_ARRAY`, `Y_CALLBACKENCODING_CSV` et `Y_CALLBACKENCODING_YOCTO_API` représentant l'encodage à utiliser pour représenter les valeurs notifiées par callback

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKENCODING_INVALID`.

network→get_callbackMaxDelay() YNetwork
network→callbackMaxDelay()YNetwork
get_callbackMaxDelay

Retourne l'attente maximale entre deux notifications par callback, en secondes.

YNetwork **target** **get_callbackMaxDelay**

Retourne :
un entier représentant l'attente maximale entre deux notifications par callback, en secondes

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKMAXDELAY_INVALID`.

network→get_callbackMethod() YNetwork
network→callbackMethod()YNetwork
get_callbackMethod

Retourne la méthode HTTP à utiliser pour signaler les changements d'état par callback.

YNetwork **target** **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()YNetwork
get_callbackMinDelay

Retourne l'attente minimale entre deux notifications par callback, en secondes.

YNetwork **target** **get_callbackMinDelay**

Retourne :
un entier représentant l'attente minimale entre deux notifications par callback, en secondes

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKMINDELAY_INVALID`.

network→get_callbackUrl() YNetwork network→callbackUrl()YNetwork get_callbackUrl

Retourne l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

YNetwork **target get_callbackUrl**

Retourne :

une chaîne de caractères représentant l'adresse (URL) de callback à notifier lors de changement d'état significatifs

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKURL_INVALID`.

network→get_discoverable() YNetwork network→discoverable()YNetwork get_discoverable

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

YNetwork **target** **get_discoverable**

Retourne :

soit `Y_DISCOVERABLE_FALSE`, soit `Y_DISCOVERABLE_TRUE`, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour)

En cas d'erreur, déclenche une exception ou retourne `Y_DISCOVERABLE_INVALID`.

network→get_ipAddress() YNetwork
network→ipAddress()YNetwork get_ipAddress

Retourne l'adresse IP utilisée par le module Yoctopuce.

YNetwork **target** **get_ipAddress**

Il peut s'agir d'une adresse configurée statiquement, ou d'une adresse reçue par un serveur DHCP.

Retourne :
une chaîne de caractères représentant l'adresse IP utilisée par le module Yoctopuce

En cas d'erreur, déclenche une exception ou retourne `Y_IPADDRESS_INVALID`.

network→get_logicalName() YNetwork
network→logicalName()YNetwork get_logicalName

Retourne le nom logique de l'interface réseau.

YNetwork **target** **get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique de l'interface réseau.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

network→get_macAddress() YNetwork
network→macAddress()YNetwork get_macAddress

Retourne l'adresse MAC de l'interface réseau, unique pour chaque module.

```
YNetwork target get_macAddress
```

L'adresse MAC est aussi présente sur un autocollant sur le module, représentée en chiffres et en code-barres.

Retourne :
une chaîne de caractères représentant l'adresse MAC de l'interface réseau, unique pour chaque module

En cas d'erreur, déclenche une exception ou retourne `Y_MACADDRESS_INVALID`.

network→get_poeCurrent() YNetwork
network→poeCurrent()YNetwork get_poeCurrent

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

YNetwork **target** **get_poeCurrent**

La consommation est mesurée après conversion en 5 Volt, et ne doit jamais dépasser 1800 mA.

Retourne :
un entier représentant le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères

En cas d'erreur, déclenche une exception ou retourne `Y_POECURRENT_INVALID`.

network→get_primaryDNS() YNetwork
network→primaryDNS()YNetwork get_primaryDNS

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

YNetwork **target** **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()YNetwork get_readiness

Retourne l'état de fonctionnement atteint par l'interface réseau.

YNetwork **target** **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()YNetwork get_router

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

YNetwork **target** **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()YNetwork
get_secondaryDNS

Retourne l'adresse IP du serveur de noms secondaire que le module doit utiliser.

YNetwork **target** **get_secondaryDNS**

Retourne :
une chaîne de caractères représentant l'adresse IP du serveur de noms secondaire que le module doit utiliser

En cas d'erreur, déclenche une exception ou retourne `Y_SECONDARYDNS_INVALID`.

network→get_subnetMask() YNetwork
network→subnetMask()YNetwork get_subnetMask

Retourne le masque de sous-réseau utilisé par le module.

YNetwork **target** **get_subnetMask**

Retourne :
une chaîne de caractères représentant le masque de sous-réseau utilisé par le module

En cas d'erreur, déclenche une exception ou retourne `Y_SUBNETMASK_INVALID`.

network→get_userPassword() YNetwork
network→userPassword()YNetwork
get_userPassword

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

YNetwork **target get_userPassword**

Retourne :
une chaîne de caractères représentant une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne `Y_USERPASSWORD_INVALID`.

network→get_wwwWatchdogDelay() YNetwork
network→wwwWatchdogDelay()YNetwork
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.

YNetwork **target** **get_wwwWatchdogDelay**

Une valeur nulle désactive le redémarrage automatique en cas de perte de connectivité WWW.

Retourne :

un entier représentant la durée de perte de connection WWW tolérée (en secondes) avant de déclancher un redémarrage automatique pour tenter de récupérer la connectivité Internet

En cas d'erreur, déclenche une exception ou retourne `Y_WWWWATCHDOGDELAY_INVALID`.

network→ping()YNetwork ping — YNetwork

Ping str_host pour vérifier la connexion réseau.

```
YNetwork target ping host
```

Envoie quatre requêtes ICMP ECHO_RESPONSER à la cible str_host depuis le module. Cette méthode retourne une chaîne de caractères avec le résultat des 4 requêtes ICMP ECHO_RESPONSE.

Paramètres :

host le nom d'hôte ou l'adresse IP de la cible

Retourne :

une chaîne de caractères contenant le résultat du ping.

network→set_adminPassword() YNetwork
network→setAdminPassword()YNetwork
set_adminPassword

Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.

```
YNetwork target set_adminPassword newval
```

Si la valeur fournie est une chaîne vide, plus aucun mot de passe n'est nécessaire. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_callbackCredentials() YNetwork
network→setCallbackCredentials()YNetwork
set_callbackCredentials

Modifie le laisser-passer pour se connecter à l'adresse de callback.

YNetwork **target** **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()YNetwork
set_callbackEncoding

Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.

YNetwork **target set_callbackEncoding newval**

Paramètres :

newval une valeur parmi `Y_CALLBACKENCODING_FORM`, `Y_CALLBACKENCODING_JSON`, `Y_CALLBACKENCODING_JSON_ARRAY`, `Y_CALLBACKENCODING_CSV` et `Y_CALLBACKENCODING_YOCTO_API` représentant l'encodage à utiliser pour représenter les valeurs notifiées par callback

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_callbackMaxDelay() YNetwork
network→setCallbackMaxDelay()YNetwork
set_callbackMaxDelay

Modifie l'attente maximale entre deux notifications par callback, en secondes.

YNetwork **target** **set_callbackMaxDelay** **newval**

Paramètres :

newval un entier représentant l'attente maximale entre deux notifications par callback, en secondes

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_callbackMethod() YNetwork
network→setCallbackMethod()YNetwork
set_callbackMethod

Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.

YNetwork **target** **set_callbackMethod** **newval**

Paramètres :

newval une valeur parmi `Y_CALLBACKMETHOD_POST`, `Y_CALLBACKMETHOD_GET` et `Y_CALLBACKMETHOD_PUT` représentant la méthode HTTP à utiliser pour signaler les changements d'état par callback

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_callbackMinDelay() YNetwork
network→setCallbackMinDelay()YNetwork
set_callbackMinDelay

Modifie l'attente minimale entre deux notifications par callback, en secondes.

YNetwork **target** **set_callbackMinDelay** **newval**

Paramètres :

newval un entier représentant l'attente minimale entre deux notifications par callback, en secondes

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_callbackUrl() YNetwork
network→setCallbackUrl()YNetwork set_callbackUrl

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

YNetwork **target** **set_callbackUrl** **newval**

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant l'adresse (URL) de callback à notifier lors de changement d'état significatifs

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_discoverable() YNetwork
network→setDiscoverable()YNetwork
set_discoverable

Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

YNetwork **target set_discoverable newval**

Paramètres :

newval soit `Y_DISCOVERABLE_FALSE`, soit `Y_DISCOVERABLE_TRUE`, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour)

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_logicalName() YNetwork
network→setLogicalName()YNetwork
set_logicalName

Modifie le nom logique de l'interface réseau.

YNetwork **target set_logicalName newval**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'interface réseau.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_primaryDNS() YNetwork
network→setPrimaryDNS()YNetwork
set_primaryDNS

Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.

YNetwork **target 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()YNetwork
set_secondaryDNS

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

```
YNetwork target set_secondaryDNS newval
```

En mode DHCP, si une valeur est spécifiée, elle remplacera celle reçue du serveur DHCP. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

newval une chaîne de caractères représentant l'adresse IP du serveur de nom secondaire que le module doit utiliser

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_userPassword() YNetwork
network→setUserPassword()YNetwork
set_userPassword

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

YNetwork **target** **set_userPassword** **newval**

Si la valeur fournie est une chaîne vide, plus aucun mot de passe n'est nécessaire. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_wwwWatchdogDelay() YNetwork
network→setWwwWatchdogDelay()YNetwork
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.

YNetwork **target 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→useDHCP()YNetwork useDHCP YNetwork

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

YNetwork **target useDHCP fallbackIpAddr fallbackSubnetMaskLen fallbackRouter**

En attendant qu'une adresse soit reçue (et indéfiniment si aucun serveur DHCP ne répond), le module utilisera les paramètres IP spécifiés à cette fonction. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

fallbackIpAddr	adresse IP à utiliser si aucun serveur DHCP ne répond
fallbackSubnetMaskLen	longueur du masque de sous-réseau à utiliser si aucun serveur DHCP ne répond. Par exemple, la valeur 24 représente 255.255.255.0.
fallbackRouter	adresse de la passerelle à utiliser si aucun serveur DHCP ne répond

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→useStaticIP()YNetwork useStaticIP YNetwork

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

YNetwork **target useStaticIP ipAddress subnetMaskLen router**

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

ipAddress	adresse IP à utiliser par le module
subnetMaskLen	longueur du masque de sous-réseau à utiliser. Par exemple, la valeur 24 représente 255.255.255.0.
router	adresse IP de la passerelle à utiliser ("default gateway")

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.29. contrôle d'OS

L'objet OsControl permet de controler le systèm d'exploitation sur lequel tourne un VirtualHub. OsControl n'est disponible que dans le VirtualHub software. Attention, cette fonctionnalité doit être explicitement activé au lancement du VirtualHub, avec l'option -o.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_oscontrol.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YOsControl = yoctolib.YOsControl;
php     require_once('yocto_oscontrol.php');
cpp     #include "yocto_oscontrol.h"
m       #import "yocto_oscontrol.h"
pas     uses yocto_oscontrol;
vb      yocto_oscontrol.vb
cs      yocto_oscontrol.cs
java    import com.yoctopuce.YoctoAPI.YOsControl;
py      from yocto_oscontrol import *
```

Fonction globales
yFindOsControl(**func**) Permet de retrouver un contrôle d'OS d'après un identifiant donné.
yFirstOsControl() Commence l'énumération des contrôle d'OS accessibles par la librairie.
Méthodes des objets `YOsControl`
oscontrol→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'OS au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
oscontrol→get_advertisedValue() Retourne la valeur courante du contrôle d'OS (pas plus de 6 caractères).
oscontrol→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.
oscontrol→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.
oscontrol→get_friendlyName() Retourne un identifiant global du contrôle d'OS au format `NOM_MODULE.NOM_FONCTION`.
oscontrol→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
oscontrol→get_functionId() Retourne l'identifiant matériel du contrôle d'OS, sans référence au module.
oscontrol→get_hardwareId() Retourne l'identifiant matériel unique du contrôle d'OS au format `SERIAL.FUNCTIONID`.
oscontrol→get_logicalName() Retourne le nom logique du contrôle d'OS.
oscontrol→get_module() Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.
oscontrol→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

oscontrol→get_shutdownCountdown()

Retourne le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé.

oscontrol→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

oscontrol→isOnline()

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

oscontrol→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

oscontrol→load(**msValidity**)

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

oscontrol→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

oscontrol→nextOsControl()

Continue l'énumération des contrôle d'OS commencée à l'aide de `yFirstOsControl()`.

oscontrol→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

oscontrol→set_logicalName(**newval**)

Modifie le nom logique du contrôle d'OS.

oscontrol→set_userData(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

oscontrol→shutdown(**secBeforeShutDown**)

Agende un arrêt de l'OS dans un nombre donné de secondes.

oscontrol→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

oscontrol→get_advertisedValue() YOsControl
oscontrol→advertisedValue()YOsControl
get_advertisedValue

Retourne la valeur courante du contrôle d'OS (pas plus de 6 caractères).

YOsControl **target get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante du contrôle d'OS (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

oscontrol→get_logicalName() YOsControl
oscontrol→logicalName()YOsControl
get_logicalName

Retourne le nom logique du contrôle d'OS.

YOsControl **target** **get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du contrôle d'OS.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

oscontrol→get_shutdownCountdown() YOsControl
oscontrol→shutdownCountdown()YOsControl
get_shutdownCountdown

Retourne le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé.

YOsControl **target get_shutdownCountdown**

Retourne :
un entier représentant le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé

En cas d'erreur, déclenche une exception ou retourne `Y_SHUTDOWNCOUNTDOWN_INVALID`.

oscontrol→set_logicalName() YOsControl
oscontrol→setLogicalName()YOsControl
set_logicalName

Modifie le nom logique du contrôle d'OS.

```
YOsControl target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du contrôle d'OS.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

oscontrol→shutdown()YOsControl shutdown YOsControl

Agende un arrêt de l'OS dans un nombre donné de secondes.

YOsControl **target shutdown secBeforeShutDown**

Paramètres :

secBeforeShutDown nombre de secondes avant l'arrêt

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.30. Interface de la fonction Power

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_power.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YPower = yoctolib.YPower;
php     require_once('yocto_power.php');
cpp     #include "yocto_power.h"
m       #import "yocto_power.h"
pas     uses yocto_power;
vb      yocto_power.vb
cs      yocto_power.cs
java    import com.yoctopuce.YoctoAPI.YPower;
py      from yocto_power import *
```

Fonction globales
yFindPower(**func**) Permet de retrouver un capteur de puissance electrique d'après un identifiant donné.
yFirstPower() 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→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_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_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→load(**msValidity**)

Met en cache les valeurs courantes du capteur de puissance electrique, avec une durée de validité spécifiée.

power→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

power→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du capteur de puissance electrique, avec une durée de validité spécifiée.

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→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.

power→calibrateFromPoints()YPower calibrateFromPoints — YPower

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

YPower **target calibrateFromPoints rawValues refValues**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→get_advertisedValue() YPower
power→advertisedValue()YPower
get_advertisedValue

Retourne la valeur courante du capteur de puissance electrique (pas plus de 6 caractères).

YPower **target** **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()YPower get_cosPhi

Retourne le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA).

```
YPower target get_cosPhi
```

Retourne :

une valeur numérique représentant le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA)

En cas d'erreur, déclenche une exception ou retourne `Y_COSPHI_INVALID`.

power→get_currentRawValue() YPower
power→currentRawValue()YPower
get_currentRawValue

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Watt, sous forme de nombre à virgule.

YPower **target** **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()YPower get_currentValue

Retourne la valeur actuelle de la puissance électrique, en Watt, sous forme de nombre à virgule.

YPower **target 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_highestValue() YPower
power→highestValue()YPower get_highestValue

Retourne la valeur maximale observée pour la puissance électrique depuis le démarrage du module.

YPower **target 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()YPower 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.

YPower **target get_logFrequency**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

power→get_logicalName() YPower
power→logicalName()YPower get_logicalName

Retourne le nom logique du capteur de puissance electrique.

YPower **target 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()YPower get_lowestValue

Retourne la valeur minimale observée pour la puissance électrique depuis le démarrage du module.

YPower **target 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()YPower get_meter

Retourne la valeur actuelle du compteur d'energie, calculée par le wattmètre en intégrant la consommation instantanée.

```
YPower target 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()YPower get_meterTimer

Retourne le temps écoulé depuis la dernière initilialisation du compteur d'énergie, en secondes

YPower **target 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_recordedData() YPower power→recordedData()YPower 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.

```
YPower target get_recordedData startTime endTime
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la 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()YPower
get_reportFrequency

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

YPower **target** **get_reportFrequency**

Retourne :
une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

power→get_resolution() YPower
power→resolution()YPower get_resolution

Retourne la résolution des valeurs mesurées.

```
YPower target get_resolution
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

power→get_unit() YPower
power→unit()YPower get_unit

Retourne l'unité dans laquelle la puissance électrique est exprimée.

```
YPower target 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→loadCalibrationPoints()YPower loadCalibrationPoints YPower

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

YPower **target loadCalibrationPoints rawValues refValues**

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→reset()YPower reset YPower

Réinitialise le compteur d'énergie.

```
YPower target reset
```

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→set_highestValue() YPower
power→setHighestValue()YPower set_highestValue

Modifie la mémoire de valeur maximale observée.

YPower **target set_highestValue newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→set_logFrequency() YPower
power→setLogFrequency()YPower
set_logFrequency

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

YPower **target set_logFrequency newval**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→set_logicalName() YPower power→setLogicalName()YPower set_logicalName

Modifie le nom logique du capteur de puissance electrique.

```
YPower target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de 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()YPower set_lowestValue

Modifie la mémoire de valeur minimale observée.

YPower **target set_lowestValue newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→set_reportFrequency() YPower
power→setReportFrequency()YPower
set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

```
YPower target set_reportFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→set_resolution() YPower
power→setResolution()YPower set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YPower target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.31. Interface de la fonction Pressure

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_pressure.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YPressure = yoctolib.YPressure;
php     require_once('yocto_pressure.php');
cpp     #include "yocto_pressure.h"
m       #import "yocto_pressure.h"
pas     uses yocto_pressure;
vb      yocto_pressure.vb
cs      yocto_pressure.cs
java    import com.yoctopuce.YoctoAPI.YPressure;
py      from yocto_pressure import *
```

Fonction globales
yFindPressure(**func**) Permet de retrouver un capteur de pression d'après un identifiant donné.
yFirstPressure() Commence l'énumération des capteurs de pression accessibles par la librairie.
Méthodes des objets `YPressure`
pressure→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
pressure→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de pression au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
pressure→get_advertisedValue() Retourne la valeur courante du capteur de pression (pas plus de 6 caractères).
pressure→get_currentRawValue() Retourne la valeur brute 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_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_unit()

Retourne l'unité dans laquelle la pression est exprimée.

pressure→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

pressure→isOnline()

Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.

pressure→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.

pressure→load(**msValidity**)

Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.

pressure→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

pressure→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.

pressure→nextPressure()

Continue l'énumération des capteurs de pression commencée à l'aide de `yFirstPressure()`.

pressure→registerTimedReportCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

pressure→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

pressure→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

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→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.

pressure→calibrateFromPoints()YPressure calibrateFromPoints YPressure

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

YPressure **target calibrateFromPoints rawValues refValues**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→get_advertisedValue() YPressure
pressure→advertisedValue()YPressure
get_advertisedValue

Retourne la valeur courante du capteur de pression (pas plus de 6 caractères).

YPressure **target get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante du capteur de pression (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

pressure→get_currentRawValue() YPressure
pressure→currentRawValue()YPressure
get_currentRawValue

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en millibar (hPa), sous forme de nombre à virgule.

YPressure **target 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()YPressure
get_currentValue

Retourne la valeur actuelle de la pression, en millibar (hPa), sous forme de nombre à virgule.

YPressure **target 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_highestValue() YPressure
pressure→highestValue()YPressure
get_highestValue

Retourne la valeur maximale observée pour la pression depuis le démarrage du module.

```
YPressure target 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()YPressure
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.

YPressure **target get_logFrequency**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

pressure→get_logicalName() YPressure
pressure→logicalName()YPressure get_logicalName

Retourne le nom logique du capteur de pression.

YPressure **target get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du capteur de pression.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

pressure→get_lowestValue() YPressure pressure→lowestValue()YPressure get_lowestValue

Retourne la valeur minimale observée pour la pression depuis le démarrage du module.

YPressure **target 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_recordedData() YPressure
pressure→recordedData()YPressure
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.

YPressure **target get_recordedData startTime endTime**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la 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()YPressure
get_reportFrequency

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
YPressure target get_reportFrequency
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

pressure→get_resolution() YPressure
pressure→resolution()YPressure get_resolution

Retourne la résolution des valeurs mesurées.

```
YPressure target get_resolution
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

pressure→get_unit() YPressure
pressure→unit()YPressure get_unit

Retourne l'unité dans laquelle la pression est exprimée.

```
YPressure target 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→loadCalibrationPoints()YPressure loadCalibrationPoints — YPressure

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

YPressure **target loadCalibrationPoints rawValues refValues**

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→set_highestValue() YPressure
pressure→setHighestValue()YPressure
set_highestValue

Modifie la mémoire de valeur maximale observée.

YPressure **target** **set_highestValue** **newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→set_logFrequency() YPressure
pressure→setLogFrequency()YPressure
set_logFrequency

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

YPressure **target set_logFrequency newval**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→set_logicalName() YPressure
pressure→setLogicalName()YPressure
set_logicalName

Modifie le nom logique du capteur de pression.

```
YPressure target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de pression.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→set_lowestValue() YPressure
pressure→setLowestValue()YPressure
set_lowestValue

Modifie la mémoire de valeur minimale observée.

```
YPressure target set_lowestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→set_reportFrequency() YPressure
pressure→setReportFrequency()YPressure
set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

YPressure **target set_reportFrequency newval**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→set_resolution() YPressure
pressure→setResolution()YPressure set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YPressure target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.32. Interface de la fonction PwmInput

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_pwminput.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YPwmInput = yoctolib.YPwmInput;
php     require_once('yocto_pwminput.php');
cpp     #include "yocto_pwminput.h"
m       #import "yocto_pwminput.h"
pas     uses yocto_pwminput;
vb      yocto_pwminput.vb
cs      yocto_pwminput.cs
java    import com.yoctopuce.YoctoAPI.YPwmInput;
py      from yocto_pwminput import *
```

Fonction globales
yFindPwmInput(**func**) Permet de retrouver un capteur de tension d'après un identifiant donné.
yFirstPwmInput() Commence l'énumération des capteurs de tension accessibles par la librairie.
Méthodes des objets `YPwmInput`
pwminput→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
pwminput→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
pwminput→get_advertisedValue() Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).
pwminput→get_currentRawValue() Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule.
pwminput→get_currentValue() Retourne la valeur courante de la fonctionnalité PwmInput, sous forme de nombre à virgule.
pwminput→get_dutyCycle() Retourne le duty cycle du PWM, en pour cents.
pwminput→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.
pwminput→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.
pwminput→get_frequency() Retourne la fréquence du PWM en Hz.
pwminput→get_friendlyName() Retourne un identifiant global du capteur de tension au format `NOM_MODULE.NOM_FONCTION`.
pwminput→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

pwminput→get_functionId()

Retourne l'identifiant matériel du capteur de tension, sans référence au module.

pwminput→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de tension au format SERIAL.FUNCTIONID.

pwminput→get_highestValue()

Retourne la valeur maximale observée pour la tension 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 du capteur de tension.

pwminput→get_lowestValue()

Retourne la valeur minimale observée pour la tension 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 changement 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_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 le capteur de tension est joignable, sans déclencher d'erreur.

pwminput→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

pwminput→load(**msValidity**)

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

pwminput→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

pwminput→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

pwminput→nextPwmInput()

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstPwmInput()`.

pwminput→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

pwminput→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

pwminput→resetCounter()

réinitialise le compteur d'impulsions et son timer

pwminput→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

pwminput→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

pwminput→set_logicalName(newval)

Modifie le nom logique du capteur de tension.

pwminput→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

pwminput→set_pwmReportMode(newval)

Change le type de paramètre (fréquence, duty cycle, longueur d'impulsion ou nombre de changement d'état) renvoyé par la fonction get_currentValue et les callback.

pwminput→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

pwminput→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

pwminput→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

pwminput→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

pwminput→calibrateFromPoints()YPwmInput YPwmInput calibrateFromPoints

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

YPwmInput **target calibrateFromPoints rawValues refValues**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→get_advertisedValue() YPwmInput
pwminput→advertisedValue()YPwmInput
get_advertisedValue

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

YPwmInput **target** **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`.

pwminput→get_currentRawValue() YPwmInput
pwminput→currentRawValue()YPwmInput
get_currentRawValue

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule.

YPwmInput **target 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`.

pwminput→get_currentValue() YPwmInput
pwminput→currentValue()YPwmInput
get_currentValue

Retourne la valeur courante de la fonctionnalité PwmInput, sous forme de nombre à virgule.

```
YPwmInput target 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_dutyCycle() YPwmInput
pwminput→dutyCycle()YPwmInput get_dutyCycle

Retourne le duty cycle du PWM, en pour cents.

YPwmInput **target 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_frequency() YPwmInput
pwminput→frequency()YPwmInput get_frequency

Retourne la fréquence du PWM en Hz.

YPwmInput **target** **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_highestValue() YPwmInput
pwminput→highestValue()YPwmInput
get_highestValue

Retourne la valeur maximale observée pour la tension depuis le démarrage du module.

YPwmInput **target 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`.

pwminput→get_logFrequency() YPwmInput
pwminput→logFrequency()YPwmInput
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.

YPwmInput **target** **get_logFrequency**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

pwminput→get_logicalName() YPwmInput
pwminput→logicalName()YPwmInput
get_logicalName

Retourne le nom logique du capteur de tension.

YPwmInput **target 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`.

pwminput→get_lowestValue() YPwmInput
pwminput→lowestValue()YPwmInput
get_lowestValue

Retourne la valeur minimale observée pour la tension depuis le démarrage du module.

YPwmInput **target** **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`.

pwminput→get_period() YPwmInput
pwminput→period()YPwmInput get_period

Retourne la période du PWM en millisecondes.

YPwmInput **target 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()YPwmInput
get_pulseCounter

Retourne la valeur du compteur d'impulsions.

YPwmInput **target** **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()YPwmInput
get_pulseDuration

Retourne la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule.

YPwmInput **target 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_pwmReportMode() YPwmInput
pwminput→pwmReportMode()YPwmInput
get_pwmReportMode

Retourne le type de paramètre (fréquence, duty cycle , longueur d'impulsion ou nombre de changement d'état) renvoyé par la fonction get_currentValue et les callback.

YPwmInput **target** **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 changement d'état) renvoyé par la fonction get_currentValue et les callback

En cas d'erreur, déclenche une exception ou retourne `Y_PWMREPORTMODE_INVALID`.

pwminput→get_recordedData() YPwmInput
pwminput→recordedData()YPwmInput
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.

```
YPwmInput target get_recordedData startTime endTime
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la 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()YPwmInput
get_reportFrequency

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

YPwmInput **target** **get_reportFrequency**

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

pwminput→get_resolution() YPwmInput
pwminput→resolution()YPwmInput get_resolution

Retourne la résolution des valeurs mesurées.

```
YPwmInput target get_resolution
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

pwminput→get_unit() YPwmInput
pwminput→unit()YPwmInput get_unit

Retourne l'unité dans laquelle la valeur retournée par get_currentValue et les callback est exprimée.

```
YPwmInput target 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→loadCalibrationPoints()YPwmInput YPwmInput loadCalibrationPoints

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

YPwmInput **target loadCalibrationPoints rawValues refValues**

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→set_highestValue() YPwmInput
pwminput→setHighestValue()YPwmInput
set_highestValue

Modifie la mémoire de valeur maximale observée.

YPwmInput **target** **set_highestValue** **newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→set_logFrequency() YPwmInput
pwminput→setLogFrequency()YPwmInput
set_logFrequency

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

YPwmInput **target set_logFrequency newval**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→set_logicalName() YPwmInput
pwminput→setLogicalName()YPwmInput
set_logicalName

Modifie le nom logique du capteur de tension.

```
YPwmInput target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de tension.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→set_lowestValue() YPwmInput
pwminput→setLowestValue()YPwmInput
set_lowestValue

Modifie la mémoire de valeur minimale observée.

YPwmInput **target set_lowestValue newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→set_pwmReportMode() YPwmInput
pwminput→setPwmReportMode()YPwmInput
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.

YPwmInput **target** **set_pwmReportMode** **newval**

Seule les six digit de droite du nombre de changement d'état sont transmis, pour les valeurs plus grandes que un million, utiliser get_pulseCounter().

Paramètres :

newval une valeur parmi Y_PWMREPORTMODE_PWM_DUTYCYCLE, Y_PWMREPORTMODE_PWM_FREQUENCY, Y_PWMREPORTMODE_PWM_PULSEDURATION et Y_PWMREPORTMODE_PWM_EDGECOUNT

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→set_reportFrequency() YPwmInput
pwminput→setReportFrequency()YPwmInput
set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

YPwmInput **target set_reportFrequency newval**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→set_resolution() YPwmInput
pwminput→setResolution()YPwmInput
set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YPwmInput target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.33. Interface de la fonction Pwm

La librairie de programmation Yoctopuce permet simplement de configurer, démarrer et arrêter le PWM.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_pwmoutput.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YPwmOutput = yoctolib.YPwmOutput;
php     require_once('yocto_pwmoutput.php');
cpp     #include "yocto_pwmoutput.h"
m       #import "yocto_pwmoutput.h"
pas     uses yocto_pwmoutput;
vb      yocto_pwmoutput.vb
cs      yocto_pwmoutput.cs
java    import com.yoctopuce.YoctoAPI.YPwmOutput;
py      from yocto_pwmoutput import *
```

Fonction globales
yFindPwmOutput(**func**) Permet de retrouver un PWM d'après un identifiant donné.
yFirstPwmOutput() Commence l'énumération des PWM accessibles par la librairie.
Méthodes des objets `YPwmOutput`
pwmoutput→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du PWM au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
pwmoutput→dutyCycleMove(**target**, **ms_duration**) Déclenche une variation progressive de la longueur des impulsions vers une valeur donnée.
pwmoutput→get_advertisedValue() Retourne la valeur courante du PWM (pas plus de 6 caractères).
pwmoutput→get_dutyCycle() Retourne le duty cycle du PWM, en pour cents.
pwmoutput→get_dutyCycleAtPowerOn() Retourne le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100
pwmoutput→get_enabled() Retourne l'état de fonctionnement du PWM.
pwmoutput→get_enabledAtPowerOn() Retourne l'état de fonctionnement du PWM à la mise sous tension du module.
pwmoutput→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du PWM.
pwmoutput→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du PWM.
pwmoutput→get_frequency() Retourne la fréquence du PWM en Hz.
pwmoutput→get_friendlyName() Retourne un identifiant global du PWM au format `NOM_MODULE.NOM_FONCTION`.
pwmoutput→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

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→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

pwmoutput→nextPwmOutput()
Continue l'énumération des PWM commencée à l'aide de yFirstPwmOutput().

pwmoutput→pulseDurationMove(**ms_target**, **ms_duration**)
Déclenche une transition progressive de la longueur des impulsions vers une valeur donnée.

pwmoutput→registerValueCallback(**callback**)
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

pwmoutput→set_dutyCycle(**newval**)
Modifie le duty cycle du PWM, en pour cents.

pwmoutput→set_dutyCycleAtPowerOn(**newval**)
Modifie le duty cycle du PWM au démarrage du module.

pwmoutput→set_enabled(**newval**)
Démarre ou arrête le PWM.

pwmoutput→set_enabledAtPowerOn(**newval**)
Modifie l'état du fonctionnement du PWM à la mise sous tension du module.

pwmoutput→set_frequency(**newval**)
Modifie la fréquence du PWM.

pwmoutput→set_logicalName(**newval**)
Modifie le nom logique du PWM.

pwmoutput→set_period(**newval**)
Modifie la période du PWM 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→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.

pwmoutput→dutyCycleMove()YPwmOutput dutyCycleMove YPwmOutput

Déclenche une variation progressive de la longueur des impulsions vers une valeur donnée.

YPwmOutput **target dutyCycleMove target ms_duration**

Paramètres :

target	nouveau duty cycle à la fin de la transition (nombre flottant, entre 0 et 1)
ms_duration	durée totale de la transition, en millisecondes

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→get_advertisedValue() YPwmOutput
pwmoutput→advertisedValue()YPwmOutput
get_advertisedValue

Retourne la valeur courante du PWM (pas plus de 6 caractères).

YPwmOutput **target** **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()YPwmOutput get_dutyCycle

Retourne le duty cycle du PWM, en pour cents.

YPwmOutput **target 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_enabled() YPwmOutput pwmoutput→enabled()YPwmOutput get_enabled

Retourne l'état de fonctionnement du PWM.

YPwmOutput **target** **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()YPwmOutput
get_enabledAtPowerOn

Retourne l'état de fonctionnement du PWM à la mise sous tension du module.

YPwmOutput **target 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_frequency() YPwmOutput
pwmoutput→frequency()YPwmOutput get_frequency

Retourne la fréquence du PWM en Hz.

YPwmOutput **target** **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_logicalName() YPwmOutput
pwmoutput→logicalName()YPwmOutput
get_logicalName

Retourne le nom logique du PWM.

YPwmOutput **target get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du PWM.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

pwmoutput→get_period() YPwmOutput
pwmoutput→period()YPwmOutput get_period

Retourne la période du PWM en millisecondes.

YPwmOutput **target** **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()YPwmOutput
get_pulseDuration

Retourne la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule.

YPwmOutput **target 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→pulseDurationMove()YPwmOutput pulseDurationMove YPwmOutput

Déclenche une transition progressive de la longueur des impulsions vers une valeur donnée.

YPwmOutput **target 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→set_dutyCycle() YPwmOutput
pwmoutput→setDutyCycle()YPwmOutput
set_dutyCycle

Modifie le duty cycle du PWM, en pour cents.

```
YPwmOutput target 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()YPwmOutput
set_dutyCycleAtPowerOn

Modifie le duty cycle du PWM au démarrage du module.

YPwmOutput **target** **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()YPwmOutput set_enabled

Démarre ou arrête le PWM.

YPwmOutput **target set_enabled newval**

Paramètres :

newval soit `Y_ENABLED_FALSE`, soit `Y_ENABLED_TRUE`

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→set_enabledAtPowerOn() YPwmOutput
pwmoutput→setEnabledAtPowerOn()YPwmOutput
set_enabledAtPowerOn

Modifie l'état du fonctionnement du PWM à la mise sous tension du module.

YPwmOutput **target** **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()YPwmOutput
set_frequency

Modifie la fréquence du PWM.

```
YPwmOutput target 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()YPwmOutput set_logicalName

Modifie le nom logique du PWM.

```
YPwmOutput target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du 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()YPwmOutput set_period

Modifie la période du PWM en millisecondes.

YPwmOutput **target** **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()YPwmOutput set_pulseDuration

Modifie la longueur des impulsions du PWM, en millisecondes.

```
YPwmOutput target 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.

3.34. Interface de la fonction PwmPowerSource

La librairie de programmation Yoctopuce permet de configurer la source de tension utilisée par tous les PWM situés sur un même module.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_pwmpowersource.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YPwmPowerSource = yoctolib.YPwmPowerSource;
php     require_once('yocto_pwmpowersource.php');
cpp     #include "yocto_pwmpowersource.h"
m       #import "yocto_pwmpowersource.h"
pas     uses yocto_pwmpowersource;
vb      yocto_pwmpowersource.vb
cs      yocto_pwmpowersource.cs
java    import com.yoctopuce.YoctoAPI.YPwmPowerSource;
py      from yocto_pwmpowersource import *
```

Fonction globales
yFindPwmPowerSource(**func**) Permet de retrouver une source de tension d'après un identifiant donné.
yFirstPwmPowerSource() Commence l'énumération des Source de tension accessibles par la librairie.
Méthodes des objets `YPwmPowerSource`
pwmpowersource→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de la source de tension au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
pwmpowersource→get_advertisedValue() Retourne la valeur courante de la source de tension (pas plus de 6 caractères).
pwmpowersource→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.
pwmpowersource→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.
pwmpowersource→get_friendlyName() Retourne un identifiant global de la source de tension au format `NOM_MODULE.NOM_FONCTION`.
pwmpowersource→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
pwmpowersource→get_functionId() Retourne l'identifiant matériel de la source de tension, sans référence au module.
pwmpowersource→get_hardwareId() Retourne l'identifiant matériel unique de la source de tension au format `SERIAL.FUNCTIONID`.
pwmpowersource→get_logicalName() Retourne le nom logique de la source de tension.
pwmpowersource→get_module() Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.
pwmpowersource→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

pwmpowersource→get_powerMode()

Retourne la source de tension utilisé par tous les PWM du même module.

pwmpowersource→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

pwmpowersource→isOnline()

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

pwmpowersource→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

pwmpowersource→load(**msValidity**)

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

pwmpowersource→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

pwmpowersource→nextPwmPowerSource()

Continue l'énumération des Source de tension commencée à l'aide de `yFirstPwmPowerSource()`.

pwmpowersource→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

pwmpowersource→set_logicalName(**newval**)

Modifie le nom logique de la source de tension.

pwmpowersource→set_powerMode(**newval**)

Modifie le mode fonctionnement des PWM qui peut sortir du 5 volts isolé issu de l'USB, du 3V isolé issu de l'USB, une tension arbitraire issue de l'alimentation externe.

pwmpowersource→set_userData(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

pwmpowersource→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

pwmpowersource→get_advertisedValue() YPwmPowerSource
pwmpowersource→advertisedValue()
YPwmPowerSource get_advertisedValue

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

```
YPwmPowerSource target get_advertisedValue
```

Retourne :

une chaîne de caractères représentant la valeur courante de la source de tension (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

pwmpowersource→get_logicalName() YPwmPowerSource
pwmpowersource→logicalName()
YPwmPowerSource get_logicalName

Retourne le nom logique de la source de tension.

```
YPwmPowerSource target get_logicalName
```

Retourne :
une chaîne de caractères représentant le nom logique de la source de tension.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

pwmpowersource→set_logicalName() YPwmPowerSource
pwmpowersource→setLogicalName()
YPwmPowerSource set_logicalName

Modifie le nom logique de la source de tension.

```
YPwmPowerSource target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de la source de tension.

Retourne :

`YAPI_SUCCESS` si l'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()
YPwmPowerSource 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.

YPwmPowerSource **target 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.

3.35. 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>
nodejs  var yoctolib = require('yoctolib');
        var YGyro = yoctolib.YGyro;
php     require_once('yocto_gyro.php');
cpp     #include "yocto_gyro.h"
m       #import "yocto_gyro.h"
pas     uses yocto_gyro;
vb      yocto_gyro.vb
cs      yocto_gyro.cs
java    import com.yoctopuce.YoctoAPI.YGyro;
py      from yocto_gyro import *
```

Fonction globales
yFindQt(**func**) Permet de retrouver un élément de quaternion d'après un identifiant donné.
yFirstQt() Commence l'énumération des éléments de quaternion accessibles par la librairie.
Méthodes des objets `YQt`
qt→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
qt→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de l'élément de quaternion au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
qt→get_advertisedValue() Retourne la valeur courante de l'élément de quaternion (pas plus de 6 caractères).
qt→get_currentRawValue() Retourne la valeur brute 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_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_unit()

Retourne l'unité dans laquelle la coordonnée est exprimée.

qt→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

qt→isOnline()

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

qt→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

qt→load(**msValidity**)

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

qt→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

qt→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

qt→nextQt()

Continue l'énumération des éléments de quaternion commencée à l'aide de `yFirstQt()`.

qt→registerTimedReportCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

qt→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

qt→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

qt→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

qt→set_logicalName(newval)

Modifie le nom logique de l'élément de quaternion.

qt→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

qt→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

qt→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

qt→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

qt→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

qt→calibrateFromPoints()YSensor calibrateFromPoints YQt

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

YSensor **target calibrateFromPoints rawValues refValues**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→get_advertisedValue() YQt
qt→advertisedValue()YSensor get_advertisedValue

Retourne la valeur courante de l'élément de quaternion (pas plus de 6 caractères).

YSensor **target** **get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante de l'élément de quaternion (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

qt→get_currentRawValue() YQt
qt→currentRawValue()YSensor get_currentRawValue

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en unités, sous forme de nombre à virgule.

```
YSensor target 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()YSensor get_currentValue

Retourne la valeur actuelle de la coordonnée, en unités, sous forme de nombre à virgule.

YSensor **target** **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_highestValue() YQt
qt→highestValue()YSensor get_highestValue

Retourne la valeur maximale observée pour la coordonnée depuis le démarrage du module.

YSensor **target** **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()YSensor 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.

YSensor **target** **get_logFrequency**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

qt→get_logicalName() YQt
qt→logicalName()YSensor get_logicalName

Retourne le nom logique de l'élément de quaternion.

YSensor **target** **get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique de l'élément de quaternion.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

qt→get_lowestValue() YQt
qt→lowestValue()YSensor get_lowestValue

Retourne la valeur minimale observée pour la coordonnée depuis le démarrage du module.

YSensor **target get_lowestValue**

Retourne :

une valeur numérique représentant la valeur minimale observée pour la coordonnée depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

qt→get_recordedData() YQt
qt→recordedData()YSensor 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.

```
YSensor target get_recordedData startTime endTime
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la 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()YSensor get_reportFrequency

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

YSensor **target** **get_reportFrequency**

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

qt→get_resolution() YQt
qt→resolution()YSensor get_resolution

Retourne la résolution des valeurs mesurées.

```
YSensor target get_resolution
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

qt→get_unit() YQt
qt→unit()YSensor get_unit

Retourne l'unité dans laquelle la coordonnée est exprimée.

YSensor **target** **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→loadCalibrationPoints()YSensor loadCalibrationPoints YQt

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

YSensor **target loadCalibrationPoints rawValues refValues**

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→set_highestValue() YQt
qt→setHighestValue()YSensor set_highestValue

Modifie la mémoire de valeur maximale observée.

YSensor **target set_highestValue newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→set_logFrequency() YQt
qt→setLogFrequency()YSensor set_logFrequency

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

YSensor **target** **set_logFrequency** **newval**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→set_logicalName() YQt
qt→setLogicalName()YSensor set_logicalName

Modifie le nom logique de l'élément de quaternion.

YSensor **target set_logicalName newval**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'élément de quaternion.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→set_lowestValue() YQt
qt→setLowestValue()YSensor set_lowestValue

Modifie la mémoire de valeur minimale observée.

YSensor **target set_lowestValue newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→set_reportFrequency() YQt
qt→setReportFrequency()YSensor
set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

YSensor **target set_reportFrequency newval**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→set_resolution() YQt
qt→setResolution()YSensor set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YSensor target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.36. 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>
nodejs  var yoctolib = require('yoctolib');
        var YRealTimeClock = yoctolib.YRealTimeClock;
php     require_once('yocto_realtimeclock.php');
cpp     #include "yocto_realtimeclock.h"
m       #import "yocto_realtimeclock.h"
pas     uses yocto_realtimeclock;
vb      yocto_realtimeclock.vb
cs      yocto_realtimeclock.cs
java    import com.yoctopuce.YoctoAPI.YRealTimeClock;
py      from yocto_realtimeclock import *
```

Fonction globales
yFindRealTimeClock(**func**) Permet de retrouver une horloge d'après un identifiant donné.
yFirstRealTimeClock() Commence l'énumération des horloge accessibles par la librairie.
Méthodes des objets `YRealTimeClock`
realtimeclock→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de l'horloge au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
realtimeclock→get_advertisedValue() Retourne la valeur courante de l'horloge (pas plus de 6 caractères).
realtimeclock→get_dateTime() Retourne l'heure courante au format "AAAA/MM/JJ hh:mm:ss"
realtimeclock→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.
realtimeclock→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.
realtimeclock→get_friendlyName() Retourne un identifiant global de l'horloge au format `NOM_MODULE.NOM_FONCTION`.
realtimeclock→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
realtimeclock→get_functionId() Retourne l'identifiant matériel de l'horloge, sans référence au module.
realtimeclock→get_hardwareId() Retourne l'identifiant matériel unique de l'horloge au format `SERIAL.FUNCTIONID`.
realtimeclock→get_logicalName() Retourne le nom logique de l'horloge.
realtimeclock→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

realtimeclock→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

realtimeclock→get_timeSet()

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

realtimeclock→get_unixTime()

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

realtimeclock→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

realtimeclock→get_utcOffset()

Retourne le nombre de secondes de 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→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

realtimeclock→nextRealTimeClock()

Continue l'énumération des horloge commencée à l'aide de `yFirstRealTimeClock()`.

realtimeclock→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

realtimeclock→set_logicalName(newval)

Modifie le nom logique de l'horloge.

realtimeclock→set_unixTime(newval)

Modifie l'heure courante.

realtimeclock→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

realtimeclock→set_utcOffset(newval)

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

realtimeclock→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

realtimeclock→get_advertisedValue() YRealTimeClock
realtimeclock→advertisedValue()YRealTimeClock
get_advertisedValue

Retourne la valeur courante de l'horloge (pas plus de 6 caractères).

YRealTimeClock **target get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante de l'horloge (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

realtimeclock→get_logicalName() YRealTimeClock
realtimeclock→logicalName()YRealTimeClock
get_logicalName

Retourne le nom logique de l'horloge.

YRealTimeClock **target** **get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique de l'horloge.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

realtimeclock→get_timeSet() YRealTimeClock
realtimeclock→timeSet()YRealTimeClock
get_timeSet

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

YRealTimeClock **target 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()YRealTimeClock
get_unixTime

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

YRealTimeClock **target** **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_utcOffset() YRealTimeClock
realtimeclock→utcOffset()YRealTimeClock
get_utcOffset

Retourne le nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone).

YRealTimeClock **target 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→set_logicalName() YRealTimeClock
realtimeclock→setLogicalName()YRealTimeClock
set_logicalName

Modifie le nom logique de l'horloge.

```
YRealTimeClock target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'horloge.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

realtimeclock→set_unixTime() YRealTimeClock
realtimeclock→setUnixTime()YRealTimeClock
set_unixTime

Modifie l'heure courante.

```
YRealTimeClock target set_unixTime newval
```

L'heure est passée au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970). Si l'heure UTC est connue, l'attribut utcOffset sera automatiquement ajusté en fonction de l'heure configurée.

Paramètres :

newval un entier représentant l'heure courante

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

realtimeclock→set_utcOffset() YRealTimeClock
realtimeclock→setUtcOffset()YRealTimeClock
set_utcOffset

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

```
YRealTimeClock target 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.

3.37. Configuration du référentiel

Cette classe permet de configurer l'orientation dans laquelle le Yocto-3D est utilisé, afin que les fonctions d'orientation relatives au plan de la surface terrestre utilisent le référentiel approprié. La classe offre aussi un processus de recalibration tridimensionnel des capteurs, permettant de compenser les variations locales de l'accélération terrestre et d'améliorer la précision des capteurs d'inclinaisons.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_refframe.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YRefFrame = yoctolib.YRefFrame;
php     require_once('yocto_refframe.php');
cpp     #include "yocto_refframe.h"
m       #import "yocto_refframe.h"
pas     uses yocto_refframe;
vb      yocto_refframe.vb
cs      yocto_refframe.cs
java    import com.yoctopuce.YoctoAPI.YRefFrame;
py      from yocto_refframe import *
```

Fonction globales
yFindRefFrame(**func**) Permet de retrouver un référentiel d'après un identifiant donné.
yFirstRefFrame() Commence l'énumération des référentiels accessibles par la librairie.
Méthodes des objets `YRefFrame`
refframe→cancel3DCalibration() Annule la calibration tridimensionnelle en cours, et rétabli les réglages normaux.
refframe→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du référentiel au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
refframe→get_3DCalibrationHint() Retourne les instructions à suivre pour procéder à la calibration tridimensionnelle initiée avec la méthode `start3DCalibration`.
refframe→get_3DCalibrationLogMsg() Retourne le dernier message de log produit par le processus de calibration.
refframe→get_3DCalibrationProgress() Retourne l'avancement global du processus de calibration tridimensionnelle initié avec la méthode `start3DCalibration`.
refframe→get_3DCalibrationStage() Retourne l'index de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.
refframe→get_3DCalibrationStageProgress() Retourne l'avancement de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.
refframe→get_advertisedValue() Retourne la valeur courante du référentiel (pas plus de 6 caractères).
refframe→get_bearing()

Retourne le cap de référence utilisé par le compas.

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_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→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→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.

refframe→set_logicalName(newval)

Modifie le nom logique du référentiel.

refframe→set_mountPosition(position, orientation)

Modifie le référentiel de la boussole et des inclinomètres.

refframe→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

refframe→start3DCalibration()

Initie le processus de calibration tridimensionnelle des capteurs.

refframe→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

refframe→cancel3DCalibration()YRefFrame cancel3DCalibration

YRefFrame

Annule la calibration tridimensionnelle en cours, et rétabli les réglages normaux.

YRefFrame **target cancel3DCalibration**

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→get_3DCalibrationHint() YRefFrame
refframe→3DCalibrationHint()YRefFrame
get_3DCalibrationHint

Retourne les instructions à suivre pour procéder à la calibration tridimensionnelle initiée avec la méthode `start3DCalibration`.

```
YRefFrame target get_3DCalibrationHint
```

Retourne :
une chaîne de caractères.

refframe→get_3DCalibrationLogMsg() YRefFrame
refframe→3DCalibrationLogMsg()YRefFrame
get_3DCalibrationLogMsg

Retourne le dernier message de log produit par le processus de calibration.

```
YRefFrame target get_3DCalibrationLogMsg
```

Si aucun nouveau message n'est disponible, retourne une chaîne vide.

Retourne :
une chaîne de caractères.

refframe→get_3DCalibrationProgress() YRefFrame
refframe→3DCalibrationProgress()YRefFrame
get_3DCalibrationProgress

Retourne l'avancement global du processus de calibration tridimensionnelle initié avec la méthode `start3DCalibration`.

YRefFrame **target** **get_3DCalibrationProgress**

Retourne :
une nombre entier entre 0 (pas commencé) et 100 (terminé).

refframe→get_3DCalibrationStage() YRefFrame
refframe→3DCalibrationStage()YRefFrame
get_3DCalibrationStage

Retourne l'index de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.

YRefFrame **target get_3DCalibrationStage**

Retourne :
une nombre entier, croissant au fur et à mesure de la complétion des étapes.

refframe→get_3DCalibrationStageProgress() YRefFrame
refframe→3DCalibrationStageProgress()YRefFrame
get_3DCalibrationStageProgress

Retourne l'avancement de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.

YRefFrame **target** **get_3DCalibrationStageProgress**

Retourne :
une nombre entier entre 0 (pas commencé) et 100 (terminé).

refframe→get_advertisedValue() YRefFrame
refframe→advertisedValue()YRefFrame
get_advertisedValue

Retourne la valeur courante du référentiel (pas plus de 6 caractères).

```
YRefFrame target 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()YRefFrame get_bearing

Retourne le cap de référence utilisé par le compas.

```
YRefFrame target get_bearing
```

Le cap relatif indiqué par le compas est la différence entre le Nord magnétique mesuré et le cap de référence spécifié ici.

Retourne :
une valeur numérique représentant le cap de référence utilisé par le compas

En cas d'erreur, déclenche une exception ou retourne `Y_BEARING_INVALID`.

reframe→get_logicalName() YRefFrame
reframe→logicalName()YRefFrame get_logicalName

Retourne le nom logique du référentiel.

YRefFrame **target get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du référentiel.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

refframe→get_mountOrientation() YRefFrame
refframe→mountOrientation()YRefFrame
get_mountOrientation

Retourne l'orientation à l'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

YRefFrame **target** **get_mountOrientation**

Retourne :

une valeur parmi l'énumération `Y_MOUNTORIENTATION` (`Y_MOUNTORIENTATION_TWELVE`, `Y_MOUNTORIENTATION_THREE`, `Y_MOUNTORIENTATION_SIX`, `Y_MOUNTORIENTATION_NINE`) correspondant à la l'orientation de la flèche "X" sur le module par rapport à un cadran d'horloge vu par un observateur au centre de la boîte. Sur la face BOTTOM le 12h pointe vers l'avant, tandis que sur la face TOP le 12h pointe vers l'arrière.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→get_mountPosition() YRefFrame
refframe→mountPosition()YRefFrame
get_mountPosition

Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

YRefFrame **target 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→more3DCalibration()YRefFrame YRefFrame more3DCalibration

Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode `start3DCalibration`.

YRefFrame **target** **more3DCalibration**

Cette méthode doit être appelée environ 5 fois par secondes après avoir positionné le module selon les instructions fournies par la méthode `get_3DCalibrationHint` (les instructions changent pendant la procédure de calibration). En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→save3DCalibration()YRefFrame save3DCalibration YRefFrame

Applique les paramètres de calibration tridimensionnelle précédemment calculés.

YRefFrame **target save3DCalibration**

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après le redémarrage du module. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→set_bearing() YRefFrame
refframe→setBearing()YRefFrame set_bearing

Modifie le cap de référence utilisé par le compas.

```
YRefFrame target 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()YRefFrame
set_logicalName

Modifie le nom logique du référentiel.

```
YRefFrame target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du référentiel.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→set_mountPosition() YRefFrame
refframe→setMountPosition()YRefFrame
set_mountPosition

Modifie le référentiel de la boussole et des inclinomètres.

YRefFrame **target set_mountPosition position orientation**

La boussole magnétique et les inclinomètres gravitationnels fonctionnent par rapport au plan parallèle à la surface terrestre. Dans les cas oÃ¹ le module n'est pas utilisé horizontalement et à l'endroit, il faut indiquer son orientation de référence (parallèle à la surface terrestre) afin que les mesures soient faites relativement à cette position.

Paramètres :

position une valeur parmi l'énumération `Y_MOUNTPOSITION` (`Y_MOUNTPOSITION_BOTTOM`, `Y_MOUNTPOSITION_TOP`, `Y_MOUNTPOSITION_FRONT`, `Y_MOUNTPOSITION_RIGHT`, `Y_MOUNTPOSITION_REAR`, `Y_MOUNTPOSITION_LEFT`), correspondant à l'installation dans une boîte, sur l'une des six faces.

orientation une valeur parmi l'énumération `Y_MOUNTORIENTATION` (`Y_MOUNTORIENTATION_TWELVE`, `Y_MOUNTORIENTATION_THREE`, `Y_MOUNTORIENTATION_SIX`, `Y_MOUNTORIENTATION_NINE`) correspondant à la l'orientation de la flèche "X" sur le module par rapport à un cadran d'horloge vu par un observateur au centre de la boîte. Sur la face BOTTOM le 12h pointe vers l'avant, tandis que sur la face TOP le 12h pointe vers l'arrière. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

refframe→start3DCalibration()YRefFrame start3DCalibration YRefFrame

Initie le processus de calibration tridimensionnelle des capteurs.

YRefFrame **target start3DCalibration**

Cette calibration est utilisée à bas niveau pour l'estimation innertielle de position et pour améliorer la précision des mesures d'inclinaison. Après avoir appelé cette méthode, il faut positionner le module selon les instructions fournies par la méthode `get_3DCalibrationHint` et appeler `more3DCalibration` environ 5 fois par secondes. La procédure de calibration est terminée lorsque la méthode `get_3DCalibrationProgress` retourne 100. Il est alors possible d'appliquer les paramètres calculés, à l'aide de la méthode `save3DCalibration`. A tout moment, la calibration peut être abandonnée à l'aide de `cancel3DCalibration`. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.38. Interface de la fonction Relay

La librairie de programmation Yoctopuce permet simplement de changer l'état du relais. Le changement d'état n'est pas persistant: le relais retournera spontanément à sa position de repos dès que le module est mis hors tension ou redémarré. La librairie permet aussi de créer des courtes impulsions de durée déterminée. Pour les modules dotés de deux sorties par relais (relai inverseur), les deux sorties sont appelées A et B, la sortie A correspondant 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>
nodejs  var yoctolib = require('yoctolib');
        var YRelay = yoctolib.YRelay;
php     require_once('yocto_relay.php');
cpp     #include "yocto_relay.h"
m       #import "yocto_relay.h"
pas     uses yocto_relay;
vb      yocto_relay.vb
cs      yocto_relay.cs
java    import com.yoctopuce.YoctoAPI.YRelay;
py      from yocto_relay import *
```

Fonction globales
yFindRelay(**func**) Permet de retrouver un relais d'après un identifiant donné.
yFirstRelay() Commence l'énumération des relais accessibles par la librairie.
Méthodes des objets `YRelay`
relay→delayedPulse(**ms_delay**, **ms_duration**) Pré-programme une impulsion
relay→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du relais au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
relay→get_advertisedValue() Retourne la valeur courante du relais (pas plus de 6 caractères).
relay→get_countdown() Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse().
relay→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du relais.
relay→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du relais.
relay→get_friendlyName() Retourne un identifiant global du relais au format `NOM_MODULE.NOM_FONCTION`.
relay→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
relay→get_functionId() Retourne l'identifiant matériel du relais, sans référence au module.

relay→get_hardwareId()

Retourne l'identifiant matériel unique du relais au format `SERIAL.FUNCTIONID`.

relay→get_logicalName()

Retourne le nom logique du relais.

relay→get_maxTimeOnStateA()

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

relay→get_maxTimeOnStateB()

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

relay→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

relay→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

relay→get_output()

Retourne l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

relay→get_pulseTimer()

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

relay→get_state()

Retourne l'état du relais (A pour la position de repos, B pour l'état actif).

relay→get_stateAtPowerOn()

Retourne l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

relay→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

relay→isOnline()

Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.

relay→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.

relay→load(**msValidity**)

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

relay→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

relay→nextRelay()

Continue l'énumération des relais commencée à l'aide de `yFirstRelay()`.

relay→pulse(**ms_duration**)

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

relay→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

relay→set_logicalName(**newval**)

Modifie le nom logique du relais.

relay→set_maxTimeOnStateA(**newval**)

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

relay→set_maxTimeOnStateB(**newval**)

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

relay→set_output(**newval**)

Modifie l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

relay→set_state(**newval**)

Modifie l'état du relais (A pour la position de repos, B pour l'état actif).

relay→set_stateAtPowerOn(**newval**)

Pré-programme l'état du relais au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

relay→set_userData(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

relay→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

relay→delayedPulse()YRelay delayedPulse YRelay

Pré-programme une impulsion

YRelay **target delayedPulse ms_delay ms_duration**

Paramètres :

ms_delay delai d'attente avant l'impulsion, en millisecondes

ms_duration durée de l'impulsion, en millisecondes

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→get_advertisedValue() YRelay
relay→advertisedValue()YRelay get_advertisedValue

Retourne la valeur courante du relais (pas plus de 6 caractères).

YRelay **target** **get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante du relais (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

relay→get_countdown() YRelay
relay→countdown()YRelay get_countdown

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse().

```
YRelay target get_countdown
```

Si aucune impulsion n'est programmée, retourne zéro.

Retourne :
un entier représentant le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse()

En cas d'erreur, déclenche une exception ou retourne `Y_COUNTDOWN_INVALID`.

relay→get_logicalName() YRelay
relay→logicalName()YRelay get_logicalName

Retourne le nom logique du relais.

YRelay **target get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du relais.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

relay→get_maxTimeOnStateA() YRelay
relay→maxTimeOnStateA()YRelay
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.

YRelay **target** **get_maxTimeOnStateA**

Zéro signifie qu'il n'y a pas de limitation

Retourne :
un entier représentant le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B

En cas d'erreur, déclenche une exception ou retourne `Y_MAXTIMEONSTATEA_INVALID`.

relay→get_maxTimeOnStateB() YRelay
relay→maxTimeOnStateB()YRelay
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.

YRelay **target get_maxTimeOnStateB**

Zéro signifie qu'il n'y a pas de limitation

Retourne :

un entier représentant le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A

En cas d'erreur, déclenche une exception ou retourne `Y_MAXTIMEONSTATEB_INVALID`.

relay→get_output() YRelay
relay→output()YRelay get_output

Retourne l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

YRelay **target** **get_output**

Retourne :

soit `Y_OUTPUT_OFF`, soit `Y_OUTPUT_ON`, selon l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur

En cas d'erreur, déclenche une exception ou retourne `Y_OUTPUT_INVALID`.

relay→get_pulseTimer() YRelay
relay→pulseTimer()YRelay 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.

YRelay **target get_pulseTimer**

Si aucune impulsion n'est en cours, retourne zéro.

Retourne :
un entier représentant le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée

En cas d'erreur, déclenche une exception ou retourne `Y_PULSETIMER_INVALID`.

relay→get_state() YRelay
relay→state()YRelay get_state

Retourne l'état du relais (A pour la position de repos, B pour l'état actif).

YRelay **target** **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()YRelay 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).

YRelay **target** **get_stateAtPowerOn**

Retourne :

une valeur parmi `Y_STATEATPOWERON_UNCHANGED`, `Y_STATEATPOWERON_A` et `Y_STATEATPOWERON_B` représentant l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement)

En cas d'erreur, déclenche une exception ou retourne `Y_STATEATPOWERON_INVALID`.

relay→pulse()YRelay 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).

YRelay **target pulse ms_duration**

Paramètres :

ms_duration durée de l'impulsion, en millisecondes

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→set_logicalName() YRelay
relay→setLogicalName()YRelay set_logicalName

Modifie le nom logique du relais.

```
YRelay target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du 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()YRelay
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.

YRelay **target** **set_maxTimeOnStateA** **newval**

Zéro signifie qu'il n'y a pas de limitation

Paramètres :

newval un entier

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→set_maxTimeOnStateB() YRelay relay→setMaxTimeOnStateB()YRelay 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.

```
YRelay target 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()YRelay set_output

Modifie l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

YRelay **target** **set_output** **newval**

Paramètres :

newval soit `Y_OUTPUT_OFF`, soit `Y_OUTPUT_ON`, selon l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→set_state() YRelay
relay→setState()YRelay set_state

Modifie l'état du relais (A pour la position de repos, B pour l'état actif).

YRelay **target set_state newval**

Paramètres :

newval soit `Y_STATE_A`, soit `Y_STATE_B`, selon l'état du relais (A pour la position de repos, B pour l'état actif)

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→set_stateAtPowerOn() YRelay
relay→setStateAtPowerOn()YRelay
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).

YRelay **target set_stateAtPowerOn newval**

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval une valeur parmi `Y_STATEATPOWERON_UNCHANGED`, `Y_STATEATPOWERON_A` et `Y_STATEATPOWERON_B`

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.39. Interface des fonctions de type senseur

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_api.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YAPI = yoctolib.YAPI;
        var YModule = yoctolib.YModule;
php     require_once('yocto_api.php');
cpp     #include "yocto_api.h"
m       #import "yocto_api.h"
pas     uses yocto_api;
vb      yocto_api.vb
cs      yocto_api.cs
java    import com.yoctopuce.YoctoAPI.YModule;
py      from yocto_api import *
```

Fonction globales
yFindSensor(**func**) Permet de retrouver un senseur d'après un identifiant donné.
yFirstSensor() Commence l'énumération des senseurs accessibles par la librairie.
Méthodes des objets `YSensor`
sensor→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
sensor→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du senseur au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
sensor→get_advertisedValue() Retourne la valeur courante du senseur (pas plus de 6 caractères).
sensor→get_currentRawValue() Retourne la valeur brute 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_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_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→load(**msValidity**)

Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.

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→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→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.

sensor→calibrateFromPoints()YSensor YSensor calibrateFromPoints

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

YSensor **target calibrateFromPoints rawValues refValues**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→get_advertisedValue() YSensor
sensor→advertisedValue()YSensor
get_advertisedValue

Retourne la valeur courante du senseur (pas plus de 6 caractères).

YSensor **target** **get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante du senseur (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

sensor→get_currentRawValue() YSensor
sensor→currentRawValue()YSensor
get_currentRawValue

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule.

YSensor **target** **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()YSensor get_currentValue

Retourne la valeur actuelle de la measure, en l'unité spécifiée, sous forme de nombre à virgule.

YSensor **target** **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_highestValue() YSensor
sensor→highestValue()YSensor get_highestValue

Retourne la valeur maximale observée pour la measure depuis le démarrage du module.

YSensor **target 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()YSensor 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.

YSensor **target** **get_logFrequency**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

sensor→get_logicalName() YSensor
sensor→logicalName()YSensor get_logicalName

Retourne le nom logique du senseur.

YSensor **target 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()YSensor get_lowestValue

Retourne la valeur minimale observée pour la measure depuis le démarrage du module.

YSensor **target 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_recordedData() YSensor
sensor→recordedData()YSensor 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.

```
YSensor target get_recordedData startTime endTime
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la 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()YSensor
get_reportFrequency

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

YSensor **target** **get_reportFrequency**

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

sensor→get_resolution() YSensor
sensor→resolution()YSensor get_resolution

Retourne la résolution des valeurs mesurées.

```
YSensor target get_resolution
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

sensor→get_unit() YSensor
sensor→unit()YSensor get_unit

Retourne l'unité dans laquelle la measure est exprimée.

```
YSensor target 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→loadCalibrationPoints()YSensor loadCalibrationPoints YSensor

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

YSensor **target loadCalibrationPoints rawValues refValues**

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→set_highestValue() YSensor
sensor→setHighestValue()YSensor set_highestValue

Modifie la mémoire de valeur maximale observée.

```
YSensor target set_highestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→set_logFrequency() YSensor
sensor→setLogFrequency()YSensor
set_logFrequency

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

YSensor **target** **set_logFrequency** **newval**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→set_logicalName() YSensor
sensor→setLogicalName()YSensor set_logicalName

Modifie le nom logique du senseur.

```
YSensor target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du senseur.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→set_lowestValue() YSensor sensor→setLowestValue()YSensor set_lowestValue

Modifie la mémoire de valeur minimale observée.

YSensor **target** **set_lowestValue** **newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→set_reportFrequency() YSensor
sensor→setReportFrequency()YSensor
set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

```
YSensor target set_reportFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→set_resolution() YSensor sensor→setResolution()YSensor set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YSensor target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.40. Interface de la fonction SerialPort

La fonction SerialPort permet de piloter entièrement un module d'interface série Yoctopuce, pour envoyer et recevoir des données et configurer les paramètres de transmission (vitesse, nombre de bits, parité, contrôle de flux et protocole). Notez que les interfaces série Yoctopuce ne sont pas des visibles comme des ports COM virtuels. Ils sont faits pour être utilisés comme tous les autres modules Yoctopuce.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_serialport.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YSerialPort = yoctolib.YSerialPort;
php     require_once('yocto_serialport.php');
cpp     #include "yocto_serialport.h"
m       #import "yocto_serialport.h"
pas     uses yocto_serialport;
vb      yocto_serialport.vb
cs      yocto_serialport.cs
java    import com.yoctopuce.YoctoAPI.YSerialPort;
py      from yocto_serialport import *
```

Fonction globales
yFindSerialPort(**func**) Permet de retrouver une port série d'après un identifiant donné.
yFirstSerialPort() Commence l'énumération des le port série accessibles par la librairie.
Méthodes des objets `YSerialPort`
serialport→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du port série au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
serialport→get_CTS() Lit l'état de la ligne CTS.
serialport→get_advertisedValue() Retourne la valeur courante du port série (pas plus de 6 caractères).
serialport→get_errCount() Retourne le nombre d'erreurs de communication détectées depuis la dernière mise à zéro.
serialport→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port série.
serialport→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port série.
serialport→get_friendlyName() Retourne un identifiant global du port série au format `NOM_MODULE.NOM_FONCTION`.
serialport→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
serialport→get_functionId() Retourne l'identifiant matériel du port série, sans référence au module.
serialport→get_hardwareId() Retourne l'identifiant matériel unique du port série au format `SERIAL.FUNCTIONID`.

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_msgCount()

Retourne le nombre de messages reçus depuis la dernière mise à zéro.

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_serialMode()

Retourne les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1".

serialport→get_txCount()

Retourne le nombre d'octets transmis 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→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→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→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→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_seek(**rxCountVal**)

Change le pointeur de position courante dans le flux de donnée à la valeur spécifiée.

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→set_RTS(**val**)

Change manuellement l'état de la ligne RTS.

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_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→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→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.

serialport→get_CTS() YSerialPort
serialport→CTS()YSerialPort get_CTS

Lit l'état de la ligne CTS.

```
YSerialPort target 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()YSerialPort
get_advertisedValue

Retourne la valeur courante du port série (pas plus de 6 caractères).

YSerialPort **target 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_errCount() YSerialPort
serialport→errCount()YSerialPort get_errCount

Retourne le nombre d'erreurs de communication détectées depuis la dernière mise à zéro.

YSerialPort **target** **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_lastMsg() YSerialPort
serialport→lastMsg()YSerialPort get_lastMsg

Retourne le dernier message reçu (pour les protocoles de type Line, Frame et Modbus).

YSerialPort **target 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()YSerialPort
get_logicalName

Retourne le nom logique du port série.

YSerialPort **target 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_msgCount() YSerialPort
serialport→msgCount()YSerialPort get_msgCount

Retourne le nombre de messages reçus depuis la dernière mise à zéro.

YSerialPort **target get_msgCount**

Retourne :
un entier représentant le nombre de messages reçus depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne `Y_MSGCOUNT_INVALID`.

serialport→get_protocol() YSerialPort
serialport→protocol()YSerialPort get_protocol

Retourne le type de protocole utilisé sur la communication série, sous forme d'une chaîne de caractères.

```
YSerialPort target 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, "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()YSerialPort get_rxCount

Retourne le nombre d'octets reçus depuis la dernière mise à zéro.

YSerialPort **target 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_serialMode() YSerialPort serialport→serialMode()YSerialPort get_serialMode

Retourne les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1".

YSerialPort **target** **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_txCount() YSerialPort
serialport→txCount()YSerialPort get_txCount

Retourne le nombre d'octets transmis depuis la dernière mise à zéro.

YSerialPort **target 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→modbusReadBits()YSerialPort modbusReadBits

YSerialPort

Lit un ou plusieurs bits contigus depuis un périphérique MODBUS.

YSerialPort **target 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 modbusReadInputBits YSerialPort

Lit un ou plusieurs bits contigus depuis un périphérique MODBUS.

YSerialPort **target 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 YSerialPort modbusReadInputRegisters

Lit un ou plusieurs registres d'entrée (registre enlecture seule) depuis un périphérique MODBUS.

YSerialPort **target 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 modbusReadRegisters YSerialPort

Lit un ou plusieurs registres interne depuis un périphérique MODBUS.

YSerialPort **target 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
YSerialPort modbusWriteAndReadRegisters

Modifie l'état de plusieurs bits (ou relais) contigus sur un périphérique MODBUS.

YSerialPort **target 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 modbusWriteBit YSerialPort

Modifie l'état d'un seul bit (ou relais) sur un périphérique MODBUS.

YSerialPort **target 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 modbusWriteBits

YSerialPort

Modifie l'état de plusieurs bits (ou relais) contigus sur un périphérique MODBUS.

```
YSerialPort target 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 modbusWriteRegister YSerialPort

Modifie la valeur d'un registre interne 16 bits sur un périphérique MODBUS.

YSerialPort **target 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 modbusWriteRegisters YSerialPort

Modifie l'état de plusieurs registres internes 16 bits contigus sur un périphérique MODBUS.

YSerialPort **target 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→queryLine()YSerialPort queryLine YSerialPort

Envoie un message sous forme de ligne de texte sur le port série, et lit la réponse reçue.

```
YSerialPort target queryLine query maxWait
```

Cette fonction ne peut être utilisée que lorsque le module est configuré en protocole 'Line'.

Paramètres :

query le message à envoyer (sans le retour de chariot)

maxWait le temps maximum d'attente pour obtenir une réponse (en millisecondes).

Retourne :

la première ligne de texte reçue après l'envoi du message. Les lignes suivantes peuvent être obtenues avec des appels à readLine ou readMessages.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→queryMODBUS()YSerialPort YSerialPort queryMODBUS

Envoie un message à un périphérique MODBUS esclave connecté au port série, et lit la réponse reçue.

YSerialPort **target 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→readHex()YSerialPort readHex YSerialPort

Lit le contenu du tampon de réception sous forme hexadécimale, à partir de la position courante dans le flux de donnée.

```
YSerialPort target 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()YSerialPort readLine YSerialPort

Lit la prochaine ligne (ou le prochain message) du tampon de réception, à partir de la position courante dans le flux de donnée.

YSerialPort **target readLine**

Cette fonction ne peut être utilisée que lorsque le module est configuré pour gérer un protocole basé message, comme en mode 'Line' ou en protocole MODBUS. Elle ne fonctionne pas dans les modes de flux continu ('Char' et 'Byte'), pour lesquels le début d'un message n'est pas défini.

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction retournera la plus ancienne ligne disponible et déplacera le pointeur de position juste après. Si aucune nouvelle ligne entière n'est disponible, la fonction retourne un chaîne vide.

Retourne :
une chaîne de caractère avec une ligne de texte

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→readMessages()YSerialPort YSerialPort readMessages

Cherche les messages entrants dans le tampon de réception correspondant à un format donné, à partir de la position courante.

YSerialPort **target readMessages pattern maxWait**

Cette fonction ne peut être utilisée que lorsque le module est configuré pour gérer un protocole basé message, comme en mode 'Line' ou en protocole MODBUS. Elle ne fonctionne pas dans les modes de flux continu ('Char' et 'Byte'), pour lesquels le début d'un message n'est pas défini.

La recherche retourne tous les messages trouvés qui correspondent au format. Tant qu'aucun message adéquat n'est trouvé, la fonction attendra, au maximum pour le temps spécifié en argument (en millisecondes).

Paramètres :

pattern une expression régulière limitée décrivant le format de message désiré, ou une chaîne vide si aucun filtrage des messages n'est désiré. Pour les protocoles binaires, le format est appliqué à la représentation hexadécimale du message.

maxWait le temps maximum d'attente pour obtenir un message, tant qu'aucun n'est trouvé dans le tampon de réception (en millisecondes).

Retourne :

un tableau de chaînes de caractères contenant les messages trouvés. Les messages binaires sont convertis automatiquement en représentation hexadécimale.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

serialport→readStr()YSerialPort readStr YSerialPort

Lit le contenu du tampon de réception sous forme de string, à partir de la position courante dans le flux de donnée.

```
YSerialPort target 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_seek()YSerialPort read_seek YSerialPort

Change le pointeur de position courante dans le flux de donnée à la valeur spécifiée.

```
YSerialPort target read_seek rxCountVal
```

Cette fonction n'a pas d'effet sur le module, elle ne fait que changer la valeur stockée dans l'objet YSerialPort qui sera utilisée pour les prochaines operations de lecture.

Paramètres :

rxCountVal l'index de position absolue (valeur de rxCount) pour les opérations de lecture suiantes.

Retourne :

rien du tout.

serialport→reset()YSerialPort reset — YSerialPort

Remet à zéro tous les compteurs et efface les tampons.

YSerialPort **target** **reset**

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→set_RTS() YSerialPort
serialport→setRTS()YSerialPort set_RTS

Change manuellement l'état de la ligne RTS.

```
YSerialPort target 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_logicalName() YSerialPort
serialport→setLogicalName()YSerialPort
set_logicalName

Modifie le nom logique du port série.

YSerialPort **target set_logicalName newval**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du port 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()YSerialPort set_protocol

Modifie le type de protocol utilisé sur la communication série.

```
YSerialPort target 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, "Char" pour un flux ASCII continu ou "Byte" pour un flux binaire continue.

Paramètres :

newval une chaîne de caractères représentant le type de protocol utilisé sur la communication série

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→set_serialMode() YSerialPort
serialport→setSerialMode()YSerialPort
set_serialMode

Modifie les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1".

```
YSerialPort target 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→writeArray()YSerialPort writeArray YSerialPort

Envoie une séquence d'octets (fournie sous forme d'une liste) sur le port série.

YSerialPort **target 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()YSerialPort writeBin YSerialPort

Envoie un objet binaire tel quel sur le port série.

```
YSerialPort target 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→writeHex()YSerialPort writeHex YSerialPort

Envoie une séquence d'octets (fournie sous forme de chaîne hexadécimale) sur le port série.

```
YSerialPort target 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()YSerialPort writeLine YSerialPort

Envoie une chaîne de caractères sur le port série, suivie d'un saut de ligne (CR LF).

```
YSerialPort target 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()YSerialPort writeMODBUS

YSerialPort

Envoie une commande MODBUS (fournie sous forme de chaîne hexadécimale) sur le port série.

```
YSerialPort target 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()YSerialPort writeStr YSerialPort

Envoie une chaîne de caractères telle quelle sur le port série.

```
YSerialPort target 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.

3.41. Interface de la fonction Servo

La librairie de programmation Yoctopuce permet non seulement de déplacer le servo vers une position donnée, mais aussi de spécifier l'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>
nodejs  var yoctolib = require('yoctolib');
        var YServo = yoctolib.YServo;
php     require_once('yocto_servo.php');
cpp     #include "yocto_servo.h"
m       #import "yocto_servo.h"
pas     uses yocto_servo;
vb      yocto_servo.vb
cs      yocto_servo.cs
java    import com.yoctopuce.YoctoAPI.YServo;
py      from yocto_servo import *
```

Fonction globales
yFindServo(**func**) Permet de retrouver un servo d'après un identifiant donné.
yFirstServo() Commence l'énumération des servo accessibles par la librairie.
Méthodes des objets `YServo`
servo→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du servo au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
servo→get_advertisedValue() Retourne la valeur courante du servo (pas plus de 6 caractères).
servo→get_enabled() Retourne l'état de fonctionnement du $FUNCTION$.
servo→get_enabledAtPowerOn() Retourne l'état du générateur de signal de commande du servo au démarrage du module.
servo→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du servo.
servo→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du servo.
servo→get_friendlyName() Retourne un identifiant global du servo au format `NOM_MODULE.NOM_FONCTION`.
servo→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
servo→get_functionId() Retourne l'identifiant matériel du servo, sans référence au module.
servo→get_hardwareId() Retourne l'identifiant matériel unique du servo au format `SERIAL.FUNCTIONID`.
servo→get_logicalName() Retourne le nom logique du servo.

servo→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

servo→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

servo→get_neutral()

Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.

servo→get_position()

Retourne la position courante du servo.

servo→get_positionAtPowerOn()

Retourne la position du servo au démarrage du module.

servo→get_range()

Retourne la plage d'utilisation du servo.

servo→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

servo→isOnline()

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

servo→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

servo→load(**msValidity**)

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

servo→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

servo→move(**target**, **ms_duration**)

Déclenche un mouvement à vitesse constante vers une position donnée.

servo→nextServo()

Continue l'énumération des servo commencée à l'aide de `yFirstServo()`.

servo→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

servo→set_enabled(**newval**)

Démarre ou arrête le $FUNCTION$.

servo→set_enabledAtPowerOn(**newval**)

Configure l'état du générateur de signal de commande du servo au démarrage du module.

servo→set_logicalName(**newval**)

Modifie le nom logique du servo.

servo→set_neutral(**newval**)

Modifie la durée de l'impulsion correspondant à la position neutre du servo.

servo→set_position(**newval**)

Modifie immédiatement la consigne de position du servo.

servo→set_positionAtPowerOn(**newval**)

Configure la position du servo au démarrage du module.

servo→set_range(**newval**)

Modifie la plage d'utilisation du servo, en pourcents.

servo→set_userData(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

servo→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

servo→get_advertisedValue() YServo
servo→advertisedValue()YServo
get_advertisedValue

Retourne la valeur courante du servo (pas plus de 6 caractères).

YServo **target get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante du servo (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

servo→get_enabled() YServo
servo→enabled()YServo get_enabled

Retourne l'état de fonctionnement du $FUNCTION$.

YServo **target** **get_enabled**

Retourne :

soit `Y_ENABLED_FALSE`, soit `Y_ENABLED_TRUE`, selon l'état de fonctionnement du $FUNCTION$

En cas d'erreur, déclenche une exception ou retourne `Y_ENABLED_INVALID`.

servo→get_enabledAtPowerOn() YServo
servo→enabledAtPowerOn()YServo
get_enabledAtPowerOn

Retourne l'état du générateur de signal de commande du servo au démarrage du module.

YServo **target** **get_enabledAtPowerOn**

Retourne :

soit `Y_ENABLEDATPOWERON_FALSE`, soit `Y_ENABLEDATPOWERON_TRUE`, selon l'état du générateur de signal de commande du servo au démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_ENABLEDATPOWERON_INVALID`.

servo→get_logicalName() YServo
servo→logicalName()YServo get_logicalName

Retourne le nom logique du servo.

```
YServo target get_logicalName
```

Retourne :
une chaîne de caractères représentant le nom logique du servo.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

servo→get_neutral() YServo
servo→neutral()YServo get_neutral

Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.

YServo **target** **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()YServo get_position

Retourne la position courante du servo.

```
YServo target get_position
```

Retourne :
un entier représentant la position courante du servo

En cas d'erreur, déclenche une exception ou retourne `Y_POSITION_INVALID`.

servo→get_positionAtPowerOn() YServo servo→positionAtPowerOn()YServo get_positionAtPowerOn

Retourne la position du servo au démarrage du module.

YServo **target** **get_positionAtPowerOn**

Retourne :
un entier représentant la position du servo au démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_POSITIONATPOWERON_INVALID`.

servo→get_range() YServo
servo→range()YServo get_range

Retourne la plage d'utilisation du servo.

```
YServo target 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→move()YServo move YServo

Déclenche un mouvement à vitesse constante vers une position donnée.

YServo **target 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→set_enabled() YServo
servo→setEnabled()YServo set_enabled

Démarre ou arrête le $FUNCTION$.

YServo **target** **set_enabled** **newval**

Paramètres :

newval soit `Y_ENABLED_FALSE`, soit `Y_ENABLED_TRUE`

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→set_enabledAtPowerOn() YServo servo→setEnabledAtPowerOn()YServo set_enabledAtPowerOn

Configure l'état du générateur de signal de commande du servo au démarrage du module.

YServo **target 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()YServo set_logicalName

Modifie le nom logique du servo.

```
YServo target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du servo.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→set_neutral() YServo
servo→setNeutral()YServo set_neutral

Modifie la durée de l'impulsion correspondant à la position neutre du servo.

```
YServo target set_neutral newval
```

La durée est spécifiée en microsecondes, et la valeur standard est 1500 [us]. Ce réglage permet de décaler la plage d'utilisation du servo. Attention, l'utilisation d'une plage supérieure aux caractéristiques du servo risque fortement d'endommager le servo.

Paramètres :

newval un entier représentant la durée de l'impulsion correspondant à la position neutre du servo

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→set_position() YServo
servo→setPosition()YServo set_position

Modifie immédiatement la consigne de position du servo.

YServo **target** **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()YServo set_positionAtPowerOn

Configure la position du servo au démarrage du module.

YServo **target 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()YServo set_range

Modifie la plage d'utilisation du servo, en pourcents.

```
YServo target set_range newval
```

La valeur 100% correspond à un signal de commande standard, variant de 1 [ms] à 2 [ms]. Pour les servos supportent une plage double, de 0.5 [ms] à 2.5 [ms], vous pouvez utiliser une valeur allant jusqu'à 200%. Attention, l'utilisation d'une plage supérieure aux caractéristiques du servo risque fortement d'endommager le servo.

Paramètres :

newval un entier représentant la plage d'utilisation du servo, en pourcents

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.42. Interface de la fonction Temperature

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_temperature.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YTemperature = yoctolib.YTemperature;
php     require_once('yocto_temperature.php');
cpp     #include "yocto_temperature.h"
m       #import "yocto_temperature.h"
pas     uses yocto_temperature;
vb      yocto_temperature.vb
cs      yocto_temperature.cs
java    import com.yoctopuce.YoctoAPI.YTemperature;
py      from yocto_temperature import *
```

Fonction globales
yFindTemperature(**func**) Permet de retrouver un capteur de température d'après un identifiant donné.
yFirstTemperature() Commence l'énumération des capteurs de température accessibles par la librairie.
Méthodes des objets `YTemperature`
temperature→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
temperature→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de température au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
temperature→get_advertisedValue() Retourne la valeur courante du capteur de température (pas plus de 6 caractères).
temperature→get_currentRawValue() Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés Celsius, sous forme de nombre à virgule.
temperature→get_currentValue() Retourne la valeur actuelle de la température, en degrés Celsius, sous forme de nombre à virgule.
temperature→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.
temperature→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.
temperature→get_friendlyName() Retourne un identifiant global du capteur de température au format `NOM_MODULE.NOM_FONCTION`.
temperature→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
temperature→get_functionId() Retourne l'identifiant matériel du capteur de température, sans référence au module.

temperature→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de température au format `SERIAL.FUNCTIONID`.

temperature→get_highestValue()

Retourne la valeur maximale observée pour la température depuis le démarrage du module.

temperature→get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

temperature→get_logicalName()

Retourne le nom logique du capteur de température.

temperature→get_lowestValue()

Retourne la valeur minimale observée pour la température depuis le démarrage du module.

temperature→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

temperature→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

temperature→get_recordedData(**startTime**, **endTime**)

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

temperature→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

temperature→get_resolution()

Retourne la résolution des valeurs mesurées.

temperature→get_sensorType()

Retourne le type de capteur de température utilisé par le module

temperature→get_unit()

Retourne l'unité dans laquelle la température est exprimée.

temperature→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

temperature→isOnline()

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

temperature→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

temperature→load(**msValidity**)

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

temperature→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

temperature→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

temperature→nextTemperature()

Continue l'énumération des capteurs de température commencée à l'aide de `yFirstTemperature()`.

temperature→registerTimedReportCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

temperature→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

temperature→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

temperature→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

temperature→set_logicalName(newval)

Modifie le nom logique du capteur de température.

temperature→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

temperature→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

temperature→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

temperature→set_sensorType(newval)

Change le type de senseur utilisé par le module.

temperature→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

temperature→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

temperature→calibrateFromPoints()YTemperature YTemperature calibrateFromPoints

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

YTemperature **target calibrateFromPoints rawValues refValues**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→get_advertisedValue() YTemperature
temperature→advertisedValue()YTemperature
get_advertisedValue

Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

YTemperature **target get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante du capteur de température (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

temperature→get_currentRawValue() YTemperature
temperature→currentRawValue()YTemperature
get_currentRawValue

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés Celsius, sous forme de nombre à virgule.

```
YTemperature target 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()YTemperature
get_currentValue

Retourne la valeur actuelle de la température, en degrés Celsius, sous forme de nombre à virgule.

YTemperature **target 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_highestValue() YTemperature
temperature→highestValue()YTemperature
get_highestValue

Retourne la valeur maximale observée pour la température depuis le démarrage du module.

YTemperature **target 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()YTemperature
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.

YTemperature **target get_logFrequency**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

temperature→get_logicalName() YTemperature
temperature→logicalName()YTemperature
get_logicalName

Retourne le nom logique du capteur de température.

YTemperature **target get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du capteur de température.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

temperature→get_lowestValue() YTemperature
temperature→lowestValue()YTemperature
get_lowestValue

Retourne la valeur minimale observée pour la température depuis le démarrage du module.

YTemperature **target get_lowestValue**

Retourne :
une valeur numérique représentant la valeur minimale observée pour la température depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

temperature→get_recordedData() YTemperature
temperature→recordedData()YTemperature
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.

```
YTemperature target get_recordedData startTime endTime
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la 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()YTemperature
get_reportFrequency

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
YTemperature target get_reportFrequency
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

temperature→get_resolution() YTemperature
temperature→resolution()YTemperature
get_resolution

Retourne la résolution des valeurs mesurées.

YTemperature **target get_resolution**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

temperature→get_sensorType() YTemperature
temperature→sensorType()YTemperature
get_sensorType

Retourne le type de capteur de température utilisé par le module

YTemperature **target get_sensorType**

Retourne :

une valeur parmi Y_SENSORTYPE_DIGITAL, Y_SENSORTYPE_TYPE_K, Y_SENSORTYPE_TYPE_E, Y_SENSORTYPE_TYPE_J, Y_SENSORTYPE_TYPE_N, Y_SENSORTYPE_TYPE_R, Y_SENSORTYPE_TYPE_S, Y_SENSORTYPE_TYPE_T, Y_SENSORTYPE_PT100_4WIRES, Y_SENSORTYPE_PT100_3WIRES et Y_SENSORTYPE_PT100_2WIRES représentant le type de capteur de température utilisé par le module

En cas d'erreur, déclenche une exception ou retourne Y_SENSORTYPE_INVALID.

temperature→get_unit() YTemperature
temperature→unit()YTemperature get_unit

Retourne l'unité dans laquelle la température est exprimée.

```
YTemperature target get_unit
```

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la température est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

temperature→loadCalibrationPoints()YTemperature YTemperature loadCalibrationPoints

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
YTemperature target loadCalibrationPoints rawValues refValues
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_highestValue() YTemperature
temperature→setHighestValue()YTemperature
set_highestValue

Modifie la mémoire de valeur maximale observée.

YTemperature **target set_highestValue newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_logFrequency() YTemperature
temperature→setLogFrequency()YTemperature
set_logFrequency

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

YTemperature **target set_logFrequency newval**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_logicalName() YTemperature
temperature→setLogicalName()YTemperature
set_logicalName

Modifie le nom logique du capteur de température.

```
YTemperature target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de 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()YTemperature
set_lowestValue

Modifie la mémoire de valeur minimale observée.

```
YTemperature target set_lowestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_reportFrequency() YTemperature
temperature→setReportFrequency()YTemperature
set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

YTemperature **target set_reportFrequency newval**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_resolution() YTemperature
temperature→setResolution()YTemperature
set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YTemperature target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_sensorType() YTemperature
temperature→setSensorType()YTemperature
set_sensorType

Change le type de senseur utilisé par le module.

YTemperature **target 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. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur parmi `Y_SENSORTYPE_DIGITAL`, `Y_SENSORTYPE_TYPE_K`, `Y_SENSORTYPE_TYPE_E`, `Y_SENSORTYPE_TYPE_J`, `Y_SENSORTYPE_TYPE_N`, `Y_SENSORTYPE_TYPE_R`, `Y_SENSORTYPE_TYPE_S`, `Y_SENSORTYPE_TYPE_T`, `Y_SENSORTYPE_PT100_4WIRES`, `Y_SENSORTYPE_PT100_3WIRES` et `Y_SENSORTYPE_PT100_2WIRES`

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.43. Interface de la fonction Tilt

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_tilt.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YTilt = yoctolib.YTilt;
php     require_once('yocto_tilt.php');
cpp     #include "yocto_tilt.h"
m       #import "yocto_tilt.h"
pas     uses yocto_tilt;
vb      yocto_tilt.vb
cs      yocto_tilt.cs
java    import com.yoctopuce.YoctoAPI.YTilt;
py      from yocto_tilt import *
```

Fonction globales
yFindTilt(**func**) Permet de retrouver un inclinomètre d'après un identifiant donné.
yFirstTilt() Commence l'énumération des inclinomètres accessibles par la librairie.
Méthodes des objets `YTilt`
tilt→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
tilt→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de l'inclinomètre au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
tilt→get_advertisedValue() Retourne la valeur courante de l'inclinomètre (pas plus de 6 caractères).
tilt→get_currentRawValue() Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés, sous forme de nombre à virgule.
tilt→get_currentValue() Retourne la valeur actuelle de l'inclinaison, en degrés, sous forme de nombre à virgule.
tilt→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.
tilt→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.
tilt→get_friendlyName() Retourne un identifiant global de l'inclinomètre au format `NOM_MODULE.NOM_FONCTION`.
tilt→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
tilt→get_functionId() Retourne l'identifiant matériel de l'inclinomètre, sans référence au module.
tilt→get_hardwareId()

Retourne l'identifiant matériel unique de l'inclinomètre au format `SERIAL.FUNCTIONID`.

tilt→get_highestValue()

Retourne la valeur maximale observée pour l'inclinaison depuis le démarrage du module.

tilt→get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

tilt→get_logicalName()

Retourne le nom logique de l'inclinomètre.

tilt→get_lowestValue()

Retourne la valeur minimale observée pour l'inclinaison depuis le démarrage du module.

tilt→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

tilt→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

tilt→get_recordedData(**startTime**, **endTime**)

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

tilt→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

tilt→get_resolution()

Retourne la résolution des valeurs mesurées.

tilt→get_unit()

Retourne l'unité dans laquelle l'inclinaison est exprimée.

tilt→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

tilt→isOnline()

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

tilt→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

tilt→load(**msValidity**)

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

tilt→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

tilt→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

tilt→nextTilt()

Continue l'énumération des inclinomètres commencée à l'aide de `yFirstTilt()`.

tilt→registerTimedReportCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

tilt→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

tilt→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

tilt→set_logFrequency(**newval**)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

tilt→set_logicalName(**newval**)

Modifie le nom logique de l'inclinomètre.

tilt→set_lowestValue(**newval**)

Modifie la mémoire de valeur minimale observée.

tilt→set_reportFrequency(**newval**)

Modifie la fréquence de notification périodique des valeurs mesurées.

tilt→set_resolution(**newval**)

Modifie la résolution des valeurs physique mesurées.

tilt→set_userData(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

tilt→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

tilt→calibrateFromPoints()YTilt calibrateFromPoints YTilt

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

YTilt **target calibrateFromPoints rawValues refValues**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→get_advertisedValue() YTilt
tilt→advertisedValue()YTilt get_advertisedValue

Retourne la valeur courante de l'inclinomètre (pas plus de 6 caractères).

```
YTilt target get_advertisedValue
```

Retourne :

une chaîne de caractères représentant la valeur courante de l'inclinomètre (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

tilt→get_currentRawValue() YTilt
tilt→currentRawValue()YTilt get_currentRawValue

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés, sous forme de nombre à virgule.

```
YTilt target 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()YTilt get_currentValue

Retourne la valeur actuelle de l'inclinaison, en degrés, sous forme de nombre à virgule.

YTilt **target 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_highestValue() YTilt
tilt→highestValue()YTilt get_highestValue

Retourne la valeur maximale observée pour l'inclinaison depuis le démarrage du module.

YTilt **target get_highestValue**

Retourne :
une valeur numérique représentant la valeur maximale observée pour l'inclinaison depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

tilt→get_logFrequency() YTilt
tilt→logFrequency()YTilt 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.

```
YTilt target get_logFrequency
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

tilt→get_logicalName() YTilt
tilt→logicalName()YTilt get_logicalName

Retourne le nom logique de l'inclinomètre.

YTilt **target get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique de l'inclinomètre.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

tilt→get_lowestValue() YTilt
tilt→lowestValue()YTilt get_lowestValue

Retourne la valeur minimale observée pour l'inclinaison depuis le démarrage du module.

YTilt **target get_lowestValue**

Retourne :

une valeur numérique représentant la valeur minimale observée pour l'inclinaison depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

tilt→get_recordedData() YTilt
tilt→recordedData()YTilt 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.

```
YTilt target get_recordedData startTime endTime
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la 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()YTilt get_reportFrequency

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
YTilt target get_reportFrequency
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

tilt→get_resolution() YTilt
tilt→resolution()YTilt get_resolution

Retourne la résolution des valeurs mesurées.

```
YTilt target get_resolution
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

tilt→get_unit() YTilt
tilt→unit()YTilt get_unit

Retourne l'unité dans laquelle l'inclinaison est exprimée.

```
YTilt target get_unit
```

Retourne :
une chaîne de caractères représentant l'unité dans laquelle l'inclinaison est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

tilt→loadCalibrationPoints()YTilt loadCalibrationPoints — YTilt

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
YTilt target loadCalibrationPoints rawValues refValues
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→set_highestValue() YTilt
tilt→setHighestValue()YTilt set_highestValue

Modifie la mémoire de valeur maximale observée.

YTilt **target set_highestValue newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→set_logFrequency() YTilt
tilt→setLogFrequency()YTilt set_logFrequency

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
YTilt target set_logFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→set_logicalName() YTilt
tilt→setLogicalName()YTilt set_logicalName

Modifie le nom logique de l'inclinomètre.

```
YTilt target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'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()YTilt set_lowestValue

Modifie la mémoire de valeur minimale observée.

YTilt **target set_lowestValue newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→set_reportFrequency() YTilt
tilt→setReportFrequency()YTilt set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

```
YTilt target set_reportFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→set_resolution() YTilt
tilt→setResolution()YTilt set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YTilt target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.44. Interface de la fonction Voc

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_voc.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YVoc = yoctolib.YVoc;
php     require_once('yocto_voc.php');
cpp     #include "yocto_voc.h"
m       #import "yocto_voc.h"
pas     uses yocto_voc;
vb      yocto_voc.vb
cs      yocto_voc.cs
java    import com.yoctopuce.YoctoAPI.YVoc;
py      from yocto_voc import *
```

Fonction globales

yFindVoc(**func**)

Permet de retrouver un capteur de Composés Organiques Volatils d'après un identifiant donné.

yFirstVoc()

Commence l'énumération des capteurs de Composés Organiques Volatils accessibles par la librairie.

Méthodes des objets YVoc

voc→calibrateFromPoints(**rawValues**, **refValues**)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

voc→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de Composés Organiques Volatils au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

voc→get_advertisedValue()

Retourne la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères).

voc→get_currentRawValue()

Retourne la valeur brute 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_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

voc→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

voc→get_friendlyName()

Retourne un identifiant global du capteur de Composés Organiques Volatils au format `NOM_MODULE.NOM_FONCTION`.

voc→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

voc→get_functionId()

Retourne l'identifiant matériel du capteur de Composés Organiques Volatils, sans référence au module.

voc→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de Composés Organiques Volatils au format `SERIAL.FUNCTIONID`.

voc→get_highestValue()

Retourne la valeur maximale observée pour le taux de VOC estimé depuis le démarrage du module.

voc→get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

voc→get_logicalName()

Retourne le nom logique du capteur de Composés Organiques Volatils.

voc→get_lowestValue()

Retourne la valeur minimale observée pour le taux de VOC estimé depuis le démarrage du module.

voc→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

voc→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

voc→get_recordedData(**startTime**, **endTime**)

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

voc→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

voc→get_resolution()

Retourne la résolution des valeurs mesurées.

voc→get_unit()

Retourne l'unité dans laquelle le taux de VOC estimé est exprimée.

voc→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

voc→isOnline()

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

voc→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

voc→load(**msValidity**)

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

voc→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

voc→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

voc→nextVoc()

Continue l'énumération des capteurs de Composés Organiques Volatils commencée à l'aide de `yFirstVoc()`.

voc→registerTimedReportCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

voc→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

voc→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

voc→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

voc→set_logicalName(newval)

Modifie le nom logique du capteur de Composés Organiques Volatils.

voc→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

voc→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

voc→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

voc→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

voc→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

voc→calibrateFromPoints()YVoc calibrateFromPoints YVoc

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

YVoc **target calibrateFromPoints rawValues refValues**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→get_advertisedValue() YVoc voc→advertisedValue()YVoc get_advertisedValue

Retourne la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères).

YVoc **target** **get_advertisedValue**

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

voc→get_currentRawValue() YVoc
voc→currentRawValue()YVoc get_currentRawValue

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en ppm (vol), sous forme de nombre à virgule.

```
YVoc target 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()YVoc get_currentValue

Retourne la valeur actuelle du taux de VOC estimé, en ppm (vol), sous forme de nombre à virgule.

YVoc **target** **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_highestValue() YVoc
voc→highestValue()YVoc get_highestValue

Retourne la valeur maximale observée pour le taux de VOC estimé depuis le démarrage du module.

YVoc **target 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()YVoc 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.

YVoc **target** **get_logFrequency**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

voc→get_logicalName() YVoc
voc→logicalName()YVoc get_logicalName

Retourne le nom logique du capteur de Composés Organiques Volatils.

YVoc **target get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du capteur de Composés Organiques Volatils.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

voc→get_lowestValue() YVoc
voc→lowestValue()YVoc get_lowestValue

Retourne la valeur minimale observée pour le taux de VOC estimé depuis le démarrage du module.

YVoc **target** **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_recordedData() YVoc
voc→recordedData()YVoc 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.

```
YVoc target get_recordedData startTime endTime
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la 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()YVoc get_reportFrequency

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

YVoc **target** **get_reportFrequency**

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

voc→get_resolution() YVoc
voc→resolution()YVoc get_resolution

Retourne la résolution des valeurs mesurées.

```
YVoc target get_resolution
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

voc→get_unit() YVoc
voc→unit()YVoc get_unit

Retourne l'unité dans laquelle le taux de VOC estimé est exprimée.

```
YVoc target 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→loadCalibrationPoints()YVoc loadCalibrationPoints — YVoc

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

YVoc **target loadCalibrationPoints rawValues refValues**

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→set_highestValue() YVoc voc→setHighestValue()YVoc set_highestValue

Modifie la mémoire de valeur maximale observée.

```
YVoc target set_highestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→set_logFrequency() YVoc
voc→setLogFrequency()YVoc set_logFrequency

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
YVoc target set_logFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→set_logicalName() YVoc voc→setLogicalName()YVoc set_logicalName

Modifie le nom logique du capteur de Composés Organiques Volatils.

```
YVoc target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de Composés Organiques Volatils.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→set_lowestValue() YVoc
voc→setLowestValue()YVoc set_lowestValue

Modifie la mémoire de valeur minimale observée.

YVoc **target set_lowestValue newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→set_reportFrequency() YVoc
voc→setReportFrequency()YVoc
set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

```
YVoc target set_reportFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→set_resolution() YVoc
voc→setResolution()YVoc set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YVoc target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.45. Interface de la fonction Voltage

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_voltage.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YVoltage = yoctolib.YVoltage;
php     require_once('yocto_voltage.php');
cpp     #include "yocto_voltage.h"
m       #import "yocto_voltage.h"
pas     uses yocto_voltage;
vb      yocto_voltage.vb
cs      yocto_voltage.cs
java    import com.yoctopuce.YoctoAPI.YVoltage;
py      from yocto_voltage import *
```

Fonction globales
yFindVoltage(**func**) Permet de retrouver un capteur de tension d'après un identifiant donné.
yFirstVoltage() Commence l'énumération des capteurs de tension accessibles par la librairie.
Méthodes des objets `YVoltage`
voltage→calibrateFromPoints(**rawValues**, **refValues**) Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.
voltage→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
voltage→get_advertisedValue() Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).
voltage→get_currentRawValue() Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule.
voltage→get_currentValue() Retourne la valeur actuelle de la tension, en Volt, sous forme de nombre à virgule.
voltage→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.
voltage→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.
voltage→get_friendlyName() Retourne un identifiant global du capteur de tension au format `NOM_MODULE.NOM_FONCTION`.
voltage→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
voltage→get_functionId() Retourne l'identifiant matériel du capteur de tension, sans référence au module.
voltage→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de tension au format `SERIAL.FUNCTIONID`.

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_unit()

Retourne l'unité dans laquelle la tension est exprimée.

voltage→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

voltage→isOnline()

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

voltage→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

voltage→load(**msValidity**)

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

voltage→loadCalibrationPoints(**rawValues**, **refValues**)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

voltage→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

voltage→nextVoltage()

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstVoltage()`.

voltage→registerTimedReportCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

voltage→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

voltage→set_highestValue(**newval**)

Modifie la mémoire de valeur maximale observée.

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→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.

voltage→calibrateFromPoints()YVoltage calibrateFromPoints YVoltage

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

YVoltage **target calibrateFromPoints rawValues refValues**

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→get_advertisedValue() YVoltage
voltage→advertisedValue()YVoltage
get_advertisedValue

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

YVoltage **target** **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()YVoltage
get_currentRawValue

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule.

```
YVoltage target 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()YVoltage get_currentValue

Retourne la valeur actuelle de la tension, en Volt, sous forme de nombre à virgule.

```
YVoltage target 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_highestValue() YVoltage
voltage→highestValue()YVoltage get_highestValue

Retourne la valeur maximale observée pour la tension depuis le démarrage du module.

YVoltage **target** **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()YVoltage 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.

YVoltage **target get_logFrequency**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `Y_LOGFREQUENCY_INVALID`.

voltage→get_logicalName() YVoltage
voltage→logicalName()YVoltage get_logicalName

Retourne le nom logique du capteur de tension.

YVoltage **target 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()YVoltage get_lowestValue

Retourne la valeur minimale observée pour la tension depuis le démarrage du module.

YVoltage **target** **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_recordedData() YVoltage
voltage→recordedData()YVoltage 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.

```
YVoltage target get_recordedData startTime endTime
```

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la 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()YVoltage
get_reportFrequency

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
YVoltage target get_reportFrequency
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne `Y_REPORTFREQUENCY_INVALID`.

voltage→get_resolution() YVoltage
voltage→resolution()YVoltage get_resolution

Retourne la résolution des valeurs mesurées.

YVoltage **target get_resolution**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

voltage→get_unit() YVoltage
voltage→unit()YVoltage get_unit

Retourne l'unité dans laquelle la tension est exprimée.

YVoltage **target** **get_unit**

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la tension est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

voltage→loadCalibrationPoints()YVoltage loadCalibrationPoints YVoltage

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
YVoltage target loadCalibrationPoints rawValues refValues
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→set_highestValue() YVoltage
voltage→setHighestValue()YVoltage
set_highestValue

Modifie la mémoire de valeur maximale observée.

YVoltage **target** **set_highestValue** **newval**

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→set_logFrequency() YVoltage
voltage→setLogFrequency()YVoltage
set_logFrequency

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

YVoltage **target set_logFrequency newval**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→set_logicalName() YVoltage
voltage→setLogicalName()YVoltage set_logicalName

Modifie le nom logique du capteur de tension.

```
YVoltage target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de 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()YVoltage set_lowestValue

Modifie la mémoire de valeur minimale observée.

```
YVoltage target set_lowestValue newval
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→set_reportFrequency() YVoltage
voltage→setReportFrequency()YVoltage
set_reportFrequency

Modifie la fréquence de notification périodique des valeurs mesurées.

YVoltage **target** **set_reportFrequency** **newval**

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→set_resolution() YVoltage
voltage→setResolution()YVoltage set_resolution

Modifie la résolution des valeurs physique mesurées.

```
YVoltage target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.46. 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.

vsource→get_advertisedValue() YVSource
vsource→advertisedValue()YVSource
get_advertisedValue

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

YVSource **target** **get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante de la source de tension (pas plus de 6 caractères)

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

vsource→get_extPowerFailure() YVSource
vsource→extPowerFailure()YVSource
get_extPowerFailure

Rend TRUE si le voltage de l'alimentation externe est trop bas.

YVSource **target get_extPowerFailure**

Retourne :

soit `Y_EXTPOWERFAILURE_FALSE`, soit `Y_EXTPOWERFAILURE_TRUE`

En cas d'erreur, déclenche une exception ou retourne `Y_EXTPOWERFAILURE_INVALID`.

vsource→get_failure() YVSource
vsource→failure()YVSource get_failure

Indique si le module est en condition d'erreur.

YVSource **target get_failure**

Il possible de savoir de quelle erreur il s'agit en testant get_overheat, get_overcurrent etc... Lorsqu'un condition d'erreur est rencontée, la tension de sortie est mise à zéro est ne peut pas être changée tant la fonction reset() n'aura pas appellée.

Retourne :

soit `Y_FAILURE_FALSE`, soit `Y_FAILURE_TRUE`

En cas d'erreur, déclenche une exception ou retourne `Y_FAILURE_INVALID`.

vsource→get_logicalName() YVSource
vsource→logicalName()YVSource get_logicalName

Retourne le nom logique de la source de tension.

YVSource **target get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique de la source de tension

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

vsource→get_overCurrent() YVSource vsource→overCurrent()YVSource get_overCurrent

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

YVSource **target** **get_overCurrent**

Retourne :

soit `Y_OVERCURRENT_FALSE`, soit `Y_OVERCURRENT_TRUE`

En cas d'erreur, déclenche une exception ou retourne `Y_OVERCURRENT_INVALID`.

vsource→get_overHeat() YVSource
vsource→overHeat()YVSource get_overHeat

Rend TRUE si le module est en surchauffe.

YVSource **target get_overHeat**

Retourne :

soit `Y_OVERHEAT_FALSE`, soit `Y_OVERHEAT_TRUE`

En cas d'erreur, déclenche une exception ou retourne `Y_OVERHEAT_INVALID`.

vsource→get_overLoad() YVSource
vsource→overLoad()YVSource get_overLoad

Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

YVSource **target** **get_overLoad**

Retourne :

soit `Y_OVERLOAD_FALSE`, soit `Y_OVERLOAD_TRUE`

En cas d'erreur, déclenche une exception ou retourne `Y_OVERLOAD_INVALID`.

vsource→get_regulationFailure() YVSource
vsource→regulationFailure()YVSource
get_regulationFailure

Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

YVSource **target get_regulationFailure**

Retourne :

soit `Y_REGULATIONFAILURE_FALSE`, soit `Y_REGULATIONFAILURE_TRUE`

En cas d'erreur, déclenche une exception ou retourne `Y_REGULATIONFAILURE_INVALID`.

vsource→get_unit() YVSource
vsource→unit()YVSource get_unit

Retourne l'unité dans laquelle la tension est exprimée.

YVSource **target** **get_unit**

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la tension est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

vsource→pulse()YVSource pulse YVSource

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

YVSource **target pulse voltage ms_duration**

Paramètres :

voltage	tension demandée, en millivolts
ms_duration	durée de l'impulsion, en millisecondes

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

vsource→set_logicalName() YVSource
vsource→setLogicalName()YVSource
set_logicalName

Modifie le nom logique de la source de tension.

YVSource **target set_logicalName newval**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de la source de tension

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

vsource→set_voltage() YVSource
vsource→setVoltage()YVSource set_voltage

Règle la tension de sortie du module (en milliVolts).

YVSource **target set_voltage newval**

Paramètres :

newval un entier

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

vsource→voltageMove()YVSource voltageMove YVSource

Déclenche une variation constante de la sortie vers une valeur donnée.

YVSource **target voltageMove target ms_duration**

Paramètres :

target	nouvelle valeur de sortie à la fin de la transition, en milliVolts.
ms_duration	durée de la transition, en millisecondes

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.47. Interface de la fonction WakeUpMonitor

La fonction WakeUpMonitor prend en charge le contrôle global de toutes les sources de réveil possibles ainsi que les mises en sommeil automatiques.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_wakeupmonitor.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YWakeUpMonitor = yoctolib.YWakeUpMonitor;
php     require_once('yocto_wakeupmonitor.php');
cpp     #include "yocto_wakeupmonitor.h"
m       #import "yocto_wakeupmonitor.h"
pas     uses yocto_wakeupmonitor;
vb      yocto_wakeupmonitor.vb
cs      yocto_wakeupmonitor.cs
java    import com.yoctopuce.YoctoAPI.YWakeUpMonitor;
py      from yocto_wakeupmonitor import *
```

Fonction globales
yFindWakeUpMonitor(**func**) Permet de retrouver un moniteur d'après un identifiant donné.
yFirstWakeUpMonitor() Commence l'énumération des Moniteurs accessibles par la librairie.
Méthodes des objets `YWakeUpMonitor`
wakeupmonitor→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du moniteur au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
wakeupmonitor→get_advertisedValue() Retourne la valeur courante du moniteur (pas plus de 6 caractères).
wakeupmonitor→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.
wakeupmonitor→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.
wakeupmonitor→get_friendlyName() Retourne un identifiant global du moniteur au format `NOM_MODULE.NOM_FONCTION`.
wakeupmonitor→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
wakeupmonitor→get_functionId() Retourne l'identifiant matériel du moniteur, sans référence au module.
wakeupmonitor→get_hardwareId() Retourne l'identifiant matériel unique du moniteur au format `SERIAL.FUNCTIONID`.
wakeupmonitor→get_logicalName() Retourne le nom logique du moniteur.
wakeupmonitor→get_module() Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.
wakeupmonitor→get_module_async(**callback**, **context**) Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

wakeupmonitor→get_nextWakeUp()

Retourne la prochaine date/heure de réveil agendée (format UNIX)

wakeupmonitor→get_powerDuration()

Retourne le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

wakeupmonitor→get_sleepCountdown()

Retourne le temps avant le prochain sommeil.

wakeupmonitor→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

wakeupmonitor→get_wakeUpReason()

Renvoie la raison du dernier réveil.

wakeupmonitor→get_wakeUpState()

Revoie l'état actuel du moniteur

wakeupmonitor→isOnline()

Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.

wakeupmonitor→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.

wakeupmonitor→load(**msValidity**)

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

wakeupmonitor→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

wakeupmonitor→nextWakeUpMonitor()

Continue l'énumération des Moniteurs commencée à l'aide de `yFirstWakeUpMonitor()`.

wakeupmonitor→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

wakeupmonitor→resetSleepCountDown()

Réinitialise le compteur de mise en sommeil.

wakeupmonitor→set_logicalName(**newval**)

Modifie le nom logique du moniteur.

wakeupmonitor→set_nextWakeUp(**newval**)

Modifie les jours de la semaine où un réveil doit avoir lieu.

wakeupmonitor→set_powerDuration(**newval**)

Modifie le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

wakeupmonitor→set_sleepCountdown(**newval**)

Modifie le temps avant le prochain sommeil .

wakeupmonitor→set_userData(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

wakeupmonitor→sleep(**secBeforeSleep**)

Déclenche une mise en sommeil jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

wakeupmonitor→sleepFor(**secUntilWakeUp**, **secBeforeSleep**)

Déclenche une mise en sommeil pour un temps donné ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

wakeupmonitor→sleepUntil(**wakeUpTime**, **secBeforeSleep**)

Déclenche une mise en sommeil jusqu'à une date donnée ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

wakeupmonitor→wait_async(**callback**, **context**)

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.

wakeupmonitor→get_advertisedValue() YWakeUpMonitor wakeupmonitor→advertisedValue()YWakeUpMonitor get_advertisedValue

Retourne la valeur courante du moniteur (pas plus de 6 caractères).

YWakeUpMonitor **target** **get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante du moniteur (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

wakeupmonitor→get_logicalName() YWakeUpMonitor
wakeupmonitor→logicalName()YWakeUpMonitor
get_logicalName

Retourne le nom logique du moniteur.

YWakeUpMonitor **target get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du moniteur.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

wakeupmonitor→get_powerDuration() YWakeUpMonitor wakeupmonitor→powerDuration()YWakeUpMonitor get_powerDuration

Retourne le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

YWakeUpMonitor **target** **get_powerDuration**

Retourne :
un entier représentant le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement

En cas d'erreur, déclenche une exception ou retourne `Y_POWERDURATION_INVALID`.

wakeupmonitor→get_sleepCountdown() YWakeUpMonitor
wakeupmonitor→sleepCountdown()YWakeUpMonitor
get_sleepCountdown

Retourne le temps avant le prochain sommeil.

YWakeUpMonitor **target 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_wakeUpReason() YWakeUpMonitor
wakeupmonitor→wakeUpReason()YWakeUpMonitor
get_wakeUpReason

Renvoie la raison du dernier réveil.

YWakeUpMonitor **target** **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→resetSleepCountDown() YWakeUpMonitor
YWakeUpMonitor resetSleepCountDown

Réinitialise le compteur de mise en sommeil.

YWakeUpMonitor **target** **resetSleepCountDown**

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set_logicalName() YWakeUpMonitor wakeupmonitor→setLogicalName()YWakeUpMonitor set_logicalName

Modifie le nom logique du moniteur.

```
YWakeUpMonitor target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du moniteur.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set_nextWakeUp() YWakeUpMonitor
wakeupmonitor→setNextWakeUp()YWakeUpMonitor
set_nextWakeUp

Modifie les jours de la semaine où un réveil doit avoir lieu.

```
YWakeUpMonitor target 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()
YWakeUpMonitor set_powerDuration

Modifie le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

```
YWakeUpMonitor target set_powerDuration newval
```

Paramètres :

newval un entier représentant le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set_sleepCountdown() YWakeUpMonitor
wakeupmonitor→setSleepCountdown()
YWakeUpMonitor set_sleepCountdown

Modifie le temps avant le prochain sommeil .

```
YWakeUpMonitor target set_sleepCountdown newval
```

Paramètres :

newval un entier représentant le temps avant le prochain sommeil

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→sleep()YWakeUpMonitor sleep YWakeUpMonitor

Déclenche une mise en sommeil jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

YWakeUpMonitor **target sleep secBeforeSleep**

Paramètres :

secBeforeSleep nombre de seconde avant la mise en sommeil

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→sleepFor()YWakeUpMonitor YWakeUpMonitor 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.

YWakeUpMonitor **target 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 sleepUntil YWakeUpMonitor

Déclenche une mise en sommeil jusqu'à une date donnée ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

YWakeUpMonitor **target sleepUntil wakeUpTime secBeforeSleep**

Le compte à rebours avant la mise en sommeil peut être annulé grâce à resetSleepCountDown.

Paramètres :

wakeUpTime	date/heure du réveil (format UNIX)
secBeforeSleep	nombre de secondes avant la mise en sommeil

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→wakeUp()YWakeUpMonitor wakeUp YWakeUpMonitor

Force un réveil.

YWakeUpMonitor **target wakeUp**

3.48. Interface de la fonction WakeUpSchedule

La fonction WakeUpSchedule implémente une condition de réveil. Le réveil est spécifiée par un ensemble de mois et/ou jours et/ou heures et/ou minutes où il doit se produire.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_wakeupschedule.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YWakeUpSchedule = yoctolib.YWakeUpSchedule;
php     require_once('yocto_wakeupschedule.php');
cpp     #include "yocto_wakeupschedule.h"
m       #import "yocto_wakeupschedule.h"
pas     uses yocto_wakeupschedule;
vb      yocto_wakeupschedule.vb
cs      yocto_wakeupschedule.cs
java    import com.yoctopuce.YoctoAPI.YWakeUpSchedule;
py      from yocto_wakeupschedule import *
```

Fonction globales

yFindWakeUpSchedule(**func**)

Permet de retrouver un réveil agendé d'après un identifiant donné.

yFirstWakeUpSchedule()

Commence l'énumération des réveils agendés accessibles par la librairie.

Méthodes des objets `YWakeUpSchedule`

wakeupschedule→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du réveil agendé au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

wakeupschedule→get_advertisedValue()

Retourne la valeur courante du réveil agendé (pas plus de 6 caractères).

wakeupschedule→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

wakeupschedule→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

wakeupschedule→get_friendlyName()

Retourne un identifiant global du réveil agendé au format `NOM_MODULE.NOM_FONCTION`.

wakeupschedule→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

wakeupschedule→get_functionId()

Retourne l'identifiant matériel du réveil agendé, sans référence au module.

wakeupschedule→get_hardwareId()

Retourne l'identifiant matériel unique du réveil agendé au format `SERIAL.FUNCTIONID`.

wakeupschedule→get_hours()

Retourne les heures où le réveil est actif..

wakeupschedule→get_logicalName()

Retourne le nom logique du réveil agendé.

wakeupschedule→get_minutes()

Retourne toutes les minutes de chaque heure où le réveil est actif.

wakeupschedule→get_minutesA()

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→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

wakeupschedule→nextWakeUpSchedule()
Continue l'énumération des réveils agendés commencée à l'aide de `yFirstWakeUpSchedule()`.

wakeupschedule→registerValueCallback(**callback**)
Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

wakeupschedule→set_hours(**newval**, **newval**)
Modifie les heures où un réveil doit avoir lieu

wakeupschedule→set_logicalName(**newval**)
Modifie le nom logique du réveil agendé.

wakeupschedule→set_minutes(**bitmap**)
Modifie toutes les minutes où un réveil doit avoir lieu

wakeupschedule→set_minutesA(**newval**, **newval**)
Modifie les minutes de l'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**, **newval**)
Modifie les jours du mois où un réveil doit avoir lieu

wakeupschedule→set_months(**newval**, **newval**)
Modifie les mois où un réveil doit avoir lieu

wakeupschedule→set_userData(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

wakeupschedule→set_weekDays(newval, newval)

Modifie les jours de la semaine où un réveil doit avoir lieu

wakeupschedule→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

wakeupschedule→get_advertisedValue() YWakeUpSchedule
wakeupschedule→advertisedValue()
YWakeUpSchedule get_advertisedValue

Retourne la valeur courante du réveil agendé (pas plus de 6 caractères).

```
YWakeUpSchedule target get_advertisedValue
```

Retourne :
une chaîne de caractères représentant la valeur courante du réveil agendé (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

wakeupschedule→get_hours() YWakeUpSchedule
wakeupschedule→hours()YWakeUpSchedule
get_hours

Retourne les heures où le réveil est actif..

YWakeUpSchedule **target get_hours**

Retourne :
un entier représentant les heures où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne `Y_HOURS_INVALID`.

wakeupschedule→get_logicalName() YWakeUpSchedule
wakeupschedule→logicalName()YWakeUpSchedule
get_logicalName

Retourne le nom logique du réveil agendé.

YWakeUpSchedule **target get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique du réveil agendé.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

wakeupschedule→get_minutes() YWakeUpSchedule
wakeupschedule→minutes()YWakeUpSchedule
get_minutes

Retourne toutes les minutes de chaque heure où le réveil est actif.

```
YWakeUpSchedule target get_minutes
```

wakeupschedule→get_minutesA() YWakeUpSchedule
wakeupschedule→minutesA()YWakeUpSchedule
get_minutesA

Retourne les minutes de l'interval 00-29 de chaque heures où le réveil est actif.

YWakeUpSchedule **target** **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()YWakeUpSchedule
get_minutesB

Retourne les minutes de l'interval 30-59 de chaque heure où le réveil est actif.

YWakeUpSchedule **target 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_monthDays() YWakeUpSchedule
wakeupschedule→monthDays()YWakeUpSchedule
get_monthDays

Retourne les jours du mois où le réveil est actif..

YWakeUpSchedule **target get_monthDays**

Retourne :
un entier représentant les jours du mois où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne `Y_MONTHDAYS_INVALID`.

wakeupschedule→get_months() YWakeUpSchedule
wakeupschedule→months()YWakeUpSchedule
get_months

Retourne les mois où le réveil est actif..

YWakeUpSchedule **target 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_weekDays() YWakeUpSchedule
wakeupschedule→weekDays()YWakeUpSchedule
get_weekDays

Retourne les jours de la semaine où le réveil est actif..

YWakeUpSchedule **target get_weekDays**

Retourne :
un entier représentant les jours de la semaine où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne `Y_WEEKDAYS_INVALID`.

wakeupschedule→set_hours() YWakeUpSchedule
wakeupschedule→setHours()YWakeUpSchedule
set_hours

Modifie les heures où un réveil doit avoir lieu

YWakeUpSchedule **target set_hours newval**

Paramètres :

newval un entier représentant les heures où un réveil doit avoir lieu

newval un entier

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set_logicalName() YWakeUpSchedule
wakeupschedule→setLogicalName()
YWakeUpSchedule set_logicalName

Modifie le nom logique du réveil agendé.

YWakeUpSchedule **target set_logicalName newval**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du réveil agendé.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set_minutes() YWakeUpSchedule
wakeupschedule→setMinutes()YWakeUpSchedule
set_minutes

Modifie toutes les minutes où un réveil doit avoir lieu

YWakeUpSchedule **target 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()YWakeUpSchedule
set_minutesA

Modifie les minutes de l'interval 00-29 où un réveil doit avoir lieu

```
YWakeUpSchedule target set_minutesA newval
```

Paramètres :

newval un entier représentant les minutes de l'interval 00-29 où un réveil doit avoir lieu

newval un entier

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set_minutesB() YWakeUpSchedule
wakeupschedule→setMinutesB()YWakeUpSchedule
set_minutesB

Modifie les minutes de l'interval 30-59 où un réveil doit avoir lieu.

YWakeUpSchedule **target 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()
YWakeUpSchedule set_monthDays

Modifie les jours du mois où un réveil doit avoir lieu

```
YWakeUpSchedule target set_monthDays newval
```

Paramètres :

newval un entier représentant les jours du mois où un réveil doit avoir lieu

newval un entier

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set_months() YWakeUpSchedule
wakeupschedule→setMonths()YWakeUpSchedule
set_months

Modifie les mois où un réveil doit avoir lieu

YWakeUpSchedule **target** **set_months** **newval**

Paramètres :

newval un entier représentant les mois où un réveil doit avoir lieu

newval un entier

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set_weekDays() YWakeUpSchedule
wakeupschedule→setWeekDays()YWakeUpSchedule
set_weekDays

Modifie les jours de la semaine où un réveil doit avoir lieu

```
YWakeUpSchedule target set_weekDays newval
```

Paramètres :

newval un entier représentant les jours de la semaine où un réveil doit avoir lieu

newval un entier

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.49. Interface de la fonction Watchdog

La fonction WatchDog est gérée comme un relais qui couperait brièvement l'alimentation d'un appareil après un d'attente temps donné afin de provoquer une réinitialisation complète de cet appareil. Il suffit d'appeler le watchdog à 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>
nodejs  var yoctolib = require('yoctolib');
        var YWatchdog = yoctolib.YWatchdog;
php     require_once('yocto_watchdog.php');
cpp     #include "yocto_watchdog.h"
m       #import "yocto_watchdog.h"
pas     uses yocto_watchdog;
vb      yocto_watchdog.vb
cs      yocto_watchdog.cs
java    import com.yoctopuce.YoctoAPI.YWatchdog;
py      from yocto_watchdog import *
```

Fonction globales
yFindWatchdog(**func**) Permet de retrouver un watchdog d'après un identifiant donné.
yFirstWatchdog() Commence l'énumération des watchdog accessibles par la librairie.
Méthodes des objets `YWatchdog`
watchdog→delayedPulse(**ms_delay**, **ms_duration**) Pré-programme une impulsion
watchdog→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du watchdog au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
watchdog→get_advertisedValue() Retourne la valeur courante du watchdog (pas plus de 6 caractères).
watchdog→get_autoStart() Retourne l'état du watchdog à la mise sous tension du module.
watchdog→get_countdown() Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse().
watchdog→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.
watchdog→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.
watchdog→get_friendlyName() Retourne un identifiant global du watchdog au format `NOM_MODULE.NOM_FONCTION`.
watchdog→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
watchdog→get_functionId()

Retourne l'identifiant matériel du watchdog, sans référence au module.

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→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

watchdog→nextWatchdog()

Continue l'énumération des watchdog commencée à l'aide de `yFirstWatchdog()`.

watchdog→pulse(**ms_duration**)

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

watchdog→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

watchdog→resetWatchdog()

Réinitialise le WatchDog.

watchdog→set_autoStart(newval)

Modifie l'état du watching au démarrage du module.

watchdog→set_logicalName(newval)

Modifie le nom logique du watchdog.

watchdog→set_maxTimeOnStateA(newval)

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

watchdog→set_maxTimeOnStateB(newval)

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

watchdog→set_output(newval)

Modifie l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

watchdog→set_running(newval)

Modifie manuellement l'état de fonctionnement du watchdog.

watchdog→set_state(newval)

Modifie l'état du watchdog (A pour la position de repos, B pour l'état actif).

watchdog→set_stateAtPowerOn(newval)

Pré-programme l'état du watchdog au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

watchdog→set_triggerDelay(newval)

Modifie le délai d'attente avant qu'un reset ne soit généré par le 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→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.

watchdog→delayedPulse()YWatchdog delayedPulse YWatchdog

Pré-programme une impulsion

YWatchdog **target delayedPulse ms_delay ms_duration**

Paramètres :

ms_delay delai d'attente avant l'impulsion, en millisecondes

ms_duration durée de l'impulsion, en millisecondes

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→get_advertisedValue() YWatchdog
watchdog→advertisedValue()YWatchdog
get_advertisedValue

Retourne la valeur courante du watchdog (pas plus de 6 caractères).

YWatchdog **target** **get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante du watchdog (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

watchdog→get_autoStart() YWatchdog
watchdog→autoStart()YWatchdog get_autoStart

Retourne l'état du watchdog à la mise sous tension du module.

YWatchdog **target 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()YWatchdog get_countdown

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse().

YWatchdog **target** **get_countdown**

Si aucune impulsion n'est programmée, retourne zéro.

Retourne :

un entier représentant le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à delayedPulse()

En cas d'erreur, déclenche une exception ou retourne `Y_COUNTDOWN_INVALID`.

watchdog→get_logicalName() YWatchdog
watchdog→logicalName()YWatchdog
get_logicalName

Retourne le nom logique du watchdog.

YWatchdog **target 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()YWatchdog
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.

```
YWatchdog target get_maxTimeOnStateA
```

Zéro signifie qu'il n'y a pas de limitation

Retourne :

un entier représentant le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B

En cas d'erreur, déclenche une exception ou retourne `Y_MAXTIMEONSTATEA_INVALID`.

watchdog→get_maxTimeOnStateB() YWatchdog
watchdog→maxTimeOnStateB()YWatchdog
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.

YWatchdog **target get_maxTimeOnStateB**

Zéro signifie qu'il n'y a pas de limitation

Retourne :
un entier représentant le temps maximal (en ms) pendant lequel le 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_output() YWatchdog
watchdog→output()YWatchdog get_output

Retourne l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

YWatchdog **target** **get_output**

Retourne :

soit `Y_OUTPUT_OFF`, soit `Y_OUTPUT_ON`, selon l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur

En cas d'erreur, déclenche une exception ou retourne `Y_OUTPUT_INVALID`.

watchdog→get_pulseTimer() YWatchdog
watchdog→pulseTimer()YWatchdog 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.

```
YWatchdog target get_pulseTimer
```

Si aucune impulsion n'est en cours, retourne zéro.

Retourne :
un entier représentant le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée

En cas d'erreur, déclenche une exception ou retourne `Y_PULSETIMER_INVALID`.

watchdog→get_running() YWatchdog
watchdog→running()YWatchdog get_running

Retourne l'état du watchdog.

YWatchdog **target** **get_running**

Retourne :

soit `Y_RUNNING_OFF`, soit `Y_RUNNING_ON`, selon l'état du watchdog

En cas d'erreur, déclenche une exception ou retourne `Y_RUNNING_INVALID`.

watchdog→get_state() YWatchdog
watchdog→state()YWatchdog get_state

Retourne l'état du watchdog (A pour la position de repos, B pour l'état actif).

YWatchdog **target** **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()YWatchdog
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).

YWatchdog **target** **get_stateAtPowerOn**

Retourne :

une valeur parmi `Y_STATEATPOWERON_UNCHANGED`, `Y_STATEATPOWERON_A` et `Y_STATEATPOWERON_B` représentant l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement)

En cas d'erreur, déclenche une exception ou retourne `Y_STATEATPOWERON_INVALID`.

watchdog→get_triggerDelay() YWatchdog
watchdog→triggerDelay()YWatchdog
get_triggerDelay

Retourne le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchog, en millisecondes.

YWatchdog **target 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()YWatchdog
get_triggerDuration

Retourne la durée d'un reset généré par le watchog, en millisecondes.

YWatchdog **target** **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→pulse()YWatchdog 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).

YWatchdog **target pulse ms_duration**

Paramètres :

ms_duration durée de l'impulsion, en millisecondes

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→resetWatchdog()YWatchdog YWatchdog resetWatchdog

Réinitialise le WatchDog.

```
YWatchdog target 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()YWatchdog set_autoStart

Modifie l'état du watching au démarrage du module.

YWatchdog **target** **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()YWatchdog
set_logicalName

Modifie le nom logique du watchdog.

YWatchdog **target set_logicalName newval**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du 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()YWatchdog
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.

YWatchdog **target set_maxTimeOnStateA newval**

Zéro signifie qu'il n'y a pas de limitation

Paramètres :

newval un entier

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→set_maxTimeOnStateB() YWatchdog watchdog→setMaxTimeOnStateB()YWatchdog 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.

```
YWatchdog target 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()YWatchdog set_output

Modifie l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

YWatchdog **target** **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()YWatchdog set_running

Modifie manuellement l'état de fonctionnement du watchdog.

YWatchdog **target** **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()YWatchdog set_state

Modifie l'état du watchdog (A pour la position de repos, B pour l'état actif).

YWatchdog **target 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()YWatchdog
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).

YWatchdog **target** **set_stateAtPowerOn** **newval**

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval une valeur parmi `Y_STATEATPOWERON_UNCHANGED`, `Y_STATEATPOWERON_A` et `Y_STATEATPOWERON_B`

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→set_triggerDelay() YWatchdog
watchdog→setTriggerDelay()YWatchdog
set_triggerDelay

Modifie le délai d'attente avant qu'un reset ne soit généré par le watchog, en millisecondes.

YWatchdog **target 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()YWatchdog
set_triggerDuration

Modifie la durée des resets générés par le watchog, en millisecondes.

```
YWatchdog target 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.

3.50. Interface de la fonction Wireless

La fonction YWireless permet de configurer et de contrôler la configuration du réseau sans fil sur les modules Yoctopuce qui en sont dotés.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_wireless.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YWireless = yoctolib.YWireless;
php     require_once('yocto_wireless.php');
cpp     #include "yocto_wireless.h"
m       #import "yocto_wireless.h"
pas     uses yocto_wireless;
vb      yocto_wireless.vb
cs      yocto_wireless.cs
java    import com.yoctopuce.YoctoAPI.YWireless;
py      from yocto_wireless import *
```

Fonction globales

yFindWireless(**func**)

Permet de retrouver une interface réseau sans fil d'après un identifiant donné.

yFirstWireless()

Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.

Méthodes des objets `YWireless`

wireless→adhocNetwork(**ssid**, **securityKey**)

Modifie la configuration de l'interface réseau sans fil pour créer un réseau sans fil sans point d'accès, en mode "ad-hoc".

wireless→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau sans fil au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

wireless→get_advertisedValue()

Retourne la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).

wireless→get_channel()

Retourne le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.

wireless→get_detectedWlans()

Retourne une liste d'objets objet YFileRecord qui décrivent les réseaux sans fils détectés.

wireless→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

wireless→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

wireless→get_friendlyName()

Retourne un identifiant global de l'interface réseau sans fil au format `NOM_MODULE.NOM_FONCTION`.

wireless→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

wireless→get_functionId()

Retourne l'identifiant matériel de l'interface réseau sans fil, sans référence au module.

wireless→get_hardwareId()

Retourne l'identifiant matériel unique de l'interface réseau sans fil au format `SERIAL.FUNCTIONID`.

wireless→get_linkQuality()

Retourne la qualité de la connection, exprimée en pourcents.

wireless→get_logicalName()

Retourne le nom logique de l'interface réseau sans fil.

wireless→get_message()

Retourne le dernier message de diagnostique de l'interface au réseau sans fil.

wireless→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

wireless→get_module_async(**callback**, **context**)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

wireless→get_security()

Retourne l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné.

wireless→get_ssid()

Retourne le nom (SSID) du réseau sans-fil sélectionné.

wireless→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

wireless→isOnline()

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

wireless→isOnline_async(**callback**, **context**)

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

wireless→joinNetwork(**ssid**, **securityKey**)

Modifie la configuration de l'interface réseau sans fil pour se connecter à un point d'accès sans fil existant (mode "infrastructure").

wireless→load(**msValidity**)

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

wireless→load_async(**msValidity**, **callback**, **context**)

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

wireless→nextWireless()

Continue l'énumération des interfaces réseau sans fil commencée à l'aide de `yFirstWireless()`.

wireless→registerValueCallback(**callback**)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

wireless→set_logicalName(**newval**)

Modifie le nom logique de l'interface réseau sans fil.

wireless→set_userData(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

wireless→softAPNetwork(**ssid**, **securityKey**)

Modifie la configuration de l'interface réseau sans fil pour créer un pseudo point d'accès sans fil ("Soft AP").

wireless→wait_async(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

wireless→adhocNetwork()YWireless adhocNetwork YWireless

Modifie la configuration de l'interface réseau sans fil pour créer un réseau sans fil sans point d'accès, en mode "ad-hoc".

YWireless **target 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→get_advertisedValue() YWireless
wireless→advertisedValue()YWireless
get_advertisedValue

Retourne la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).

YWireless **target** **get_advertisedValue**

Retourne :
une chaîne de caractères représentant la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

wireless→get_channel() YWireless
wireless→channel()YWireless get_channel

Retourne le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.

YWireless **target 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()YWireless
get_detectedWlans

Retourne une liste d'objets objet YFileRecord qui décrivent les réseaux sans fils détectés.

YWireless **target** **get_detectedWlans**

La liste n'est pas mise à jour quand le module est déjà connecté à un accès sans fil (mode "infrastructure"). Pour forcer la détection des réseaux sans fil, il faut appeler `addhocNetwork()` pour se déconnecter du réseau actuel. L'appelant est responsable de la désallocation de la liste retournée.

Retourne :

une liste d'objets `YWlanRecord`, contenant le SSID, le canal, la qualité du signal, et l'algorithme de sécurité utilisé par le réseau sans-fil

En cas d'erreur, déclenche une exception ou retourne une liste vide.

wireless→get_linkQuality() YWireless
wireless→linkQuality()YWireless get_linkQuality

Retourne la qualité de la connection, exprimée en pourcents.

YWireless **target get_linkQuality**

Retourne :
un entier représentant la qualité de la connection, exprimée en pourcents

En cas d'erreur, déclenche une exception ou retourne `Y_LINKQUALITY_INVALID`.

wireless→get_logicalName() YWireless
wireless→logicalName()YWireless get_logicalName

Retourne le nom logique de l'interface réseau sans fil.

YWireless **target** **get_logicalName**

Retourne :
une chaîne de caractères représentant le nom logique de l'interface réseau sans fil.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

wireless→get_message() YWireless
wireless→message()YWireless get_message

Retourne le dernier message de diagnostique de l'interface au réseau sans fil.

YWireless **target 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_security() YWireless wireless→security()YWireless get_security

Retourne l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné.

YWireless **target** **get_security**

Retourne :

une valeur parmi `Y_SECURITY_UNKNOWN`, `Y_SECURITY_OPEN`, `Y_SECURITY_WEP`, `Y_SECURITY_WPA` et `Y_SECURITY_WPA2` représentant l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné

En cas d'erreur, déclenche une exception ou retourne `Y_SECURITY_INVALID`.

wireless→get_ssid() YWireless
wireless→ssid()YWireless get_ssid

Retourne le nom (SSID) du réseau sans-fil sélectionné.

YWireless **target 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→joinNetwork()YWireless joinNetwork YWireless

Modifie la configuration de l'interface réseau sans fil pour se connecter à un point d'accès sans fil existant (mode "infrastructure").

```
YWireless target joinNetwork ssid securityKey
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

ssid	nom du réseau sans fil à utiliser
securityKey	clé d'accès au réseau, sous forme de chaîne de caractères

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wireless→set_logicalName() YWireless
wireless→setLogicalName()YWireless
set_logicalName

Modifie le nom logique de l'interface réseau sans fil.

YWireless **target set_logicalName newval**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'interface réseau sans fil.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wireless→softAPNetwork()YWireless softAPNetwork YWireless

Modifie la configuration de l'interface réseau sans fil pour créer un pseudo point d'accès sans fil ("Soft AP").

YWireless **target 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.

Index

H

I

J

K

L

M

N

O

P

Q

R

S