Référence de l'API Objective-C

Table des matières

1. Introduction

Ce manuel est votre référence pour l'utilisation de la librairie Objective-C 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 Objective-C

Objective-C est le langage de prédilection pour programmer sous Mac OS X, en raison de son intégration avec le générateur d'interfaces Cocoa. Pour pouvoir utiliser la libraire Objective-C vous aurez impérativement besoin de XCode 4.2, qui est disponible gratuitement sous Lion. Si vous êtes encore sous Snow Leopard il vous faudra être enregistré comme développeur auprès d'Apple pour pourvoir télécharger XCode 4.2. La librairie Yoctopuce est compatible ARC. Il vous sera donc possible de coder vos projet soit en utilisant la traditionnelle méthode de *retain / release*, soit en activant l'*Automatic Reference Counting*.

Les librairies Yoctopuce[1] pour Objective-C vous sont fournies au format source dans leur intégralité. Une partie de la librairie de bas-niveau est écrite en C pur sucre, mais vous n'aurez à priori pas besoin d'interagir directement avec elle: tout a été fait pour que l'interaction soit le plus simple possible depuis Objective-C.

Vous allez rapidement vous rendre compte que l'API Objective-C définit beaucoup de fonctions qui retournent des objets. Vous ne devez jamais désallouer ces objets vous-même. Ils seront désalloués automatiquement par l'API à la fin de l'application.

Afin des les garder simples, tous les exemples fournis dans cette documentation sont des applications consoles. Il va de soit que que les fonctionnement des librairies est strictement identiques si vous les intégrez dans une application dotée d'une interface graphique. Vous trouverez sur le blog de Yoctopuce un exemple détaillé[2] avec des séquences vidéo montrant comment intégrer les fichiers de la librairie à vos projets.

2.1. Contrôle de la fonction Led

Lancez Xcode 4.2 et ouvrez le projet exemple correspondant, fourni dans le répertoire **Examples/ Doc-GettingStarted-Yocto-Demo** de la librairie Yoctopuce.

```
#import <Foundation/Foundation.h>
#import "yocto_api.h"
#import "yocto_led.h"


static void usage(void)
{
    NSLog(@"usage: demo <serial_number>  [ on | off ]");
    NSLog(@"       demo <logical_name> [ on | off ]");
    NSLog(@"       demo any [ on | off ]                (use any discovered device)");
    exit(1);
```

[1] www.yoctopuce.com/FR/libraries.php
[2] www.yoctopuce.com/FR/article/nouvelle-librairie-objective-c-pour-mac-os-x

```
}


int main(int argc, const char * argv[])
{
    NSError *error;
    if(argc < 3) {
        usage();
    }

    @autoreleasepool {
        NSString *target = [NSString stringWithUTF8String:argv[1]];
        NSString *on_off = [NSString stringWithUTF8String:argv[2]];
        YLed     *led;

        if([YAPI RegisterHub:@"usb": &error] != YAPI_SUCCESS) {
            NSLog(@"RegisterHub error: %@", [error localizedDescription]);
            return 1;
        }
        if([target isEqualToString:@"any"]){
            led =  [YLed FirstLed];
        }else{
            led =  [YLed FindLed:[target stringByAppendingString:@".led"]];
        }
        if ([led isOnline]) {
            if ([on_off isEqualToString:@"on"])
                [led set_power:Y_POWER_ON];
            else
                [led set_power:Y_POWER_OFF];
        } else {
            NSLog(@"Module not connected (check identification and USB cable)\n");
        }
    }
    return 0;
}
```

Il n'y a que peu de lignes véritablement importantes dans le code précédent. Nous allons les expliquer en détail.

yocto_api.h et yocto_led.h

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

[YAPI RegisterHub]

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

[Led FindLed]

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

```
YLed *led = [YLed FindLed:@"YCTOPOC1-123456.led"];
YLed *led = [YLed FindLed:@"YCTOPOC1-123456.MaFonction"];
YLed *led = [YLed FindLed:@"MonModule.led"];
YLed *led = [YLed FindLed:@"MonModule.MaFonction"];
YLed *led = [YLed FindLed:@"MaFonction"];
```

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

isOnline

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

set_power

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

2.2. Contrôle de la partie module

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

```
#import <Foundation/Foundation.h>
#import "yocto_api.h"

static void usage(const char *exe)
{
    NSLog(@"usage: %s <serial or logical name> [ON/OFF]\n",exe);
    exit(1);
}


int main (int argc, const char * argv[])
{
    NSError *error;

    @autoreleasepool {
        // Setup the API to use local USB devices
        if([YAPI RegisterHub:@"usb": &error] != YAPI_SUCCESS) {
            NSLog(@"RegisterHub error: %@", [error localizedDescription]);
            return 1;
        }
        if(argc < 2)
            usage(argv[0]);
        NSString *serial_or_name =[NSString stringWithUTF8String:argv[1]];
        // use serial or logical name
        YModule *module = [YModule FindModule:serial_or_name];
        if ([module isOnline]) {
            if (argc > 2) {
                if (strcmp(argv[2], "ON")==0)
                    [module setBeacon:Y_BEACON_ON];
                else
                    [module setBeacon:Y_BEACON_OFF];
            }
            NSLog(@"serial:       %@\n", [module serialNumber]);
            NSLog(@"logical name: %@\n", [module logicalName]);
            NSLog(@"luminosity:   %d\n", [module luminosity]);
            NSLog(@"beacon:       ");
            if ([module beacon] == Y_BEACON_ON)
               NSLog(@"ON\n");
            else
               NSLog(@"OFF\n");
            NSLog(@"upTime:       %ld sec\n", [module upTime]/1000);
            NSLog(@"USB current:  %d mA\n",  [module usbCurrent]);
            NSLog(@"logs:  %@\n",  [module get_lastLogs]);
        } else {
            NSLog(@"%@ not connected (check identification and USB cable)\n",
                serial_or_name);
        }
    }
    return 0;
}
```

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

Modifications des réglages du module

Lorsque que vous souhaitez modifier les réglages d'un module, il suffit d'appeler la fonction `set_xxx:` correspondante, cependant cette modification n'a lieu que dans la mémoire vive du module: si le module redémarre, les modifications seront perdues. Pour qu'elle soient mémorisées de manière persistante, il est nécessaire de demander au module de sauvegarder sa configuration courante dans sa mémoire non volatile. Pour cela il faut utiliser la méthode `saveToFlash`. Inversement il est possible de forcer le module à oublier ses réglages courants en utilisant la méthode `revertFromFlash`. Ce petit exemple ci-dessous vous permet changer le nom logique d'un module.

```
#import <Foundation/Foundation.h>
#import "yocto_api.h"

static void usage(const char *exe)
{
    NSLog(@"usage: %s <serial> <newLogicalName>\n",exe);
    exit(1);
}


int main (int argc, const char * argv[])
{
    NSError *error;

    @autoreleasepool {
        // Setup the API to use local USB devices
        if([YAPI RegisterHub:@"usb" :&error] != YAPI_SUCCESS) {
            NSLog(@"RegisterHub error: %@", [error localizedDescription]);
            return 1;
        }

        if(argc < 2)
            usage(argv[0]);

        NSString *serial_or_name =[NSString stringWithUTF8String:argv[1]];
        // use serial or logical name
        YModule *module = [YModule FindModule:serial_or_name];

        if (module.isOnline) {
            if (argc >= 3){
                NSString *newname =  [NSString stringWithUTF8String:argv[2]];
                if (![YAPI CheckLogicalName:newname]){
                    NSLog(@"Invalid name (%@)\n", newname);
                    usage(argv[0]);
                }
                module.logicalName = newname;
                [module saveToFlash];
            }
            NSLog(@"Current name: %@\n", module.logicalName);
        } else {
            NSLog(@"%@ not connected (check identification and USB cable)\n",
                serial_or_name);
        }
    }
    return 0;
}
```

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

Enumeration des modules

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

```
#import <Foundation/Foundation.h>
#import "yocto_api.h"

int main (int argc, const char * argv[])
{
    NSError *error;

    @autoreleasepool {
        // Setup the API to use local USB devices
        if([YAPI RegisterHub:@"usb" :&error] != YAPI_SUCCESS) {
            NSLog(@"RegisterHub error: %@\n", [error localizedDescription]);
            return 1;
        }

        NSLog(@"Device list:\n");

        YModule *module = [YModule FirstModule];
        while (module != nil) {
            NSLog(@"%@ %@",module.serialNumber, module.productName);
            module = [module nextModule];
        }
    }
    return 0;
}
```

2.3. Gestion des erreurs

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

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

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

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

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

cadre attendu pour la valeur retournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI_SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

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

3. Reference

3.1. Fonctions générales

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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).

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

reframe→set_logicalName(newval)

Modifie le nom logique du référentiel.

reframe→set_mountPosition(position, orientation)

Modifie le référentiel de la boussole et des inclinomètres.

reframe→set_userData(data)

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

reframe→start3DCalibration()

Initie le processus de calibration tridimensionnelle des capteurs.

reframe→wait_async(callback, context)

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

3.38. 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Index