



Référence de l'API JAVA pour Android

Table des matières

1. Introduction	1
2. Utilisation du Yocto-Demo avec Android	3
2.1. Accès Natif et Virtual Hub.	3
2.2. Préparation	3
2.3. Compatibilité	3
2.4. Activer le port USB sous Android	4
2.5. Contrôle de la fonction Led	6
2.6. Contrôle de la partie module	8
2.7. Gestion des erreurs	13
Blueprint	16
3. Reference	16
3.1. Fonctions générales	17
3.2. Interface de la fonction Accelerometer	36
3.3. Interface de la fonction Altitude	78
3.4. Interface de la fonction AnButton	120
3.5. Interface de la fonction CarbonDioxide	158
3.6. Interface de la fonction ColorLed	197
3.7. Interface de la fonction Compass	226
3.8. Interface de la fonction Current	266
3.9. Interface de la fonction DataLogger	305
3.10. Séquence de données mise en forme	339
3.11. Séquence de données enregistrées	349
3.12. Séquence de données enregistrées brute	361
3.13. Interface de la fonction DigitalIO	376
3.14. Interface de la fonction Display	420
3.15. Interface des objets DisplayLayer	467
3.16. Interface de contrôle de l'alimentation	499
3.17. Interface de la fonction Files	524
3.18. Interface de la fonction GenericSensor	551
3.19. Interface de la fonction Gyro	600
3.20. Interface d'un port de Yocto-hub	651
3.21. Interface de la fonction Humidity	676
3.22. Interface de la fonction Led	715

3.23. Interface de la fonction LightSensor	742
3.24. Interface de la fonction Magnetometer	784
3.25. Valeur mesurée	826
3.26. Interface de contrôle du module	832
3.27. Interface de la fonction Motor	872
3.28. Interface de la fonction Network	913
3.29. contrôle d'OS	970
3.30. Interface de la fonction Power	993
3.31. Interface de la fonction Pressure	1036
3.32. Interface de la fonction PwmInput	1075
3.33. Interface de la fonction Pwm	1123
3.34. Interface de la fonction PwmPowerSource	1161
3.35. Interface du quaternion	1184
3.36. Interface de la fonction Horloge Temps Real	1223
3.37. Configuration du référentiel	1250
3.38. Interface de la fonction Relay	1286
3.39. Interface des fonctions de type senseur	1322
3.40. Interface de la fonction SerialPort	1361
3.41. Interface de la fonction Servo	1418
3.42. Interface de la fonction Temperature	1453
3.43. Interface de la fonction Tilt	1494
3.44. Interface de la fonction Voc	1533
3.45. Interface de la fonction Voltage	1572
3.46. Interface de la fonction Source de tension	1611
3.47. Interface de la fonction WakeUpMonitor	1643
3.48. Interface de la fonction WakeUpSchedule	1678
3.49. Interface de la fonction Watchdog	1715
3.50. Interface de la fonction Wireless	1760

Index	1791
--------------------	-------------

1. Introduction

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

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

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

2. Utilisation du Yocto-Demo avec Android

A vrai dire, Android n'est pas un langage de programmation, c'est un système d'exploitation développé par Google pour les appareils portables tels que smart phones et tablettes. Mais il se trouve que sous Android tout est programmé avec le même langage de programmation: Java. En revanche les paradigmes de programmation et les possibilités d'accès au hardware sont légèrement différentes par rapport au Java classique, ce qui justifie un chapitre à part sur la programmation Android.

2.1. Accès Natif et Virtual Hub.

Contrairement à l'API Java classique, l'API Java pour Android accède aux modules USB de manière native. En revanche, comme il n'existe pas de VirtualHub tournant sous Android, il n'est pas possible de prendre le contrôle à distance de modules Yoctopuce pilotés par une machine sous Android. Bien sûr, l'API Java pour Android reste parfaitement capable de se connecter à un VirtualHub tournant sur un autre OS.

2.2. Préparation

Connectez-vous sur le site de Yoctopuce et téléchargez la librairie de programmation pour Java pour Android¹. La librairie est disponible en fichiers sources, mais elle aussi disponible sous la forme d'un fichier jar. Branchez vos modules, décompressez les fichiers de la librairie dans le répertoire de votre choix. Et configurez votre environnement de programmation Android pour qu'il puisse les trouver.

Afin de les garder simples, tous les exemples fournis dans cette documentation sont des fragments d'application Android. Vous devrez les intégrer dans vos propres applications Android pour les faire fonctionner. En revanche vous pourrez trouver des applications complètes dans les exemples fournis avec la librairie Java pour Android.

2.3. Compatibilité

Dans un monde idéal, il suffirait d'avoir un téléphone sous Android pour pouvoir faire fonctionner des modules Yoctopuce. Malheureusement, la réalité est légèrement différente, un appareil tournant sous Android doit répondre à un certain nombre d'exigences pour pouvoir faire fonctionner des modules USB Yoctopuce en natif.

¹ www.yoctopuce.com/FR/libraries.php

Android 4.x

Android 4.0 (api 14) et suivants sont officiellement supportés. Théoriquement le support USB *host* fonctionne depuis Android 3.1. Mais sachez que Yoctopuce ne teste régulièrement l'API Java pour Android qu'à partir de Android 4.

Support USB *host*

Il faut bien sûr que votre machine dispose non seulement d'un port USB, mais il faut aussi que ce port soit capable de tourner en mode *host*. En mode *host*, la machine prend littéralement le contrôle des périphériques qui lui sont raccordés. Les ports USB d'un ordinateur bureau, par exemple, fonctionnent mode *host*. Le pendant du mode *host* est le mode *device*. Les clefs USB par exemple fonctionnent en mode *device*: elles ne peuvent qu'être contrôlées par un *host*. Certains ports USB sont capables de fonctionner dans les deux modes, ils s'agit de ports *OTG (On The Go)*. Il se trouve que beaucoup d'appareils portables ne fonctionnent qu'en mode "device": ils sont conçus pour être branchés à chargeur ou un ordinateur de bureau, rien de plus. Il est donc fortement recommandé de lire attentivement les spécifications techniques d'un produit fonctionnant sous Android avant d'espérer le voir fonctionner avec des modules Yoctopuce.

Disposer d'une version correcte d'Android et de ports USB fonctionnant en mode *host* ne suffit malheureusement pas pour garantir un bon fonctionnement avec des modules Yoctopuce sous Android. En effet certains constructeurs configurent leur image Android afin que les périphériques autres que clavier et mass storage soit ignorés, et cette configuration est difficilement détectable. En l'état actuel des choses, le meilleur moyen de savoir avec certitude si un matériel Android spécifique fonctionne avec les modules Yoctopuce consiste à essayer.

Matériel supporté

La librairie est testée et validée sur les machines suivantes:

- Samsung Galaxy S3
- Samsung Galaxy Note 2
- Google Nexus 5
- Google Nexus 7
- Acer Iconia Tab A200
- Asus Tranformer Pad TF300T
- Kurio 7

Si votre machine Android n'est pas capable de faire fonctionner nativement des modules Yoctopuce, il vous reste tout de même la possibilité de contrôler à distance des modules pilotés par un VirtualHub sur un autre OS ou un YoctoHub².

2.4. Activer le port USB sous Android

Par défaut Android n'autorise pas une application à accéder aux périphériques connectés au port USB. Pour que votre application puisse interagir avec un module Yoctopuce branché directement sur votre tablette sur un port USB quelques étapes supplémentaires sont nécessaires. Si vous comptez uniquement interagir avec des modules connectés sur une autre machine par IP, vous pouvez ignorer cette section.

Il faut déclarer dans son `AndroidManifest.xml` l'utilisation de la fonctionnalité "USB Host" en ajoutant le tag `<uses-feature android:name="android.hardware.usb.host" />` dans la section `manifest`.

```
<manifest ...>
...
<uses-feature android:name="android.hardware.usb.host" />;
...
```

² Les YoctoHub sont un moyen simple et efficace d'ajouter une connectivité réseau à vos modules Yoctopuce. <http://www.yoctopuce.com/FR/products/category/extensions-et-reseau>


```
</manifest>
```

Lors du premier accès à un module Yoctopuce, Android va ouvrir une fenêtre pour informer l'utilisateur que l'application va accéder module connecté. L'utilisateur peut refuser ou autoriser l'accès au périphérique. Si l'utilisateur accepte, l'application pourra accéder au périphérique connecté jusqu'à la prochaine déconnexion du périphérique. Pour que la librairie Yoctopuce puisse gérer correctement ces autorisations, il faut lui fournir un pointeur sur le contexte de l'application en appelant la méthode `EnableUSBHost` de la classe `YAPI` avant le premier accès USB. Cette fonction prend en argument un objet de la classe `android.content.Context` (ou d'une sous-classe). Comme la classe `Activity` est une sous-classe de `Context`, le plus simple est de d'appeler `YAPI.EnableUSBHost(this)` ; dans la méthode `onCreate` de votre application. Si l'objet passé en paramètre n'est pas du bon type, une exception `YAPI_Exception` sera générée.

```
...
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    try {
        // Pass the application Context to the Yoctopuce Library
        YAPI.EnableUSBHost(this);
    } catch (YAPI_Exception e) {
        Log.e("Yocto", e.getLocalizedMessage());
    }
}
...
```

Lancement automatique

Il est possible d'enregistrer son application comme application par défaut pour un module USB, dans ce cas dès qu'un module sera connecté au système, l'application sera lancée automatiquement. Il faut ajouter `<action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"/>` dans la section `<intent-filter>` de l'activité principale. La section `<activity>` doit contenir un pointeur sur un fichier xml qui contient la liste des modules USB qui peuvent lancer l'application.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
...
<uses-feature android:name="android.hardware.usb.host" />
...
<application ... >
    <activity
        android:name=".MainActivity" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>

        <meta-data
            android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
            android:resource="@xml/device_filter" />
        </activity>
    </application>
</manifest>
```

Le fichier XML qui contient la liste des modules qui peuvent lancer l'application doit être sauvé dans le répertoire `res/xml`. Ce fichier contient une liste de `vendorId` et `deviceId` USB en décimal. L'exemple suivant lance l'application dès qu'un Yocto-Relay ou un Yocto-PowerRelay est connecté. Vous pouvez trouver le `vendorId` et `deviceId` des modules Yoctopuce dans la section caractéristiques de la documentation.

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
    <usb-device vendor-id="9440" product-id="12" />
    <usb-device vendor-id="9440" product-id="13" />
</resources>
```

2.5. Contrôle de la fonction Led

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

```
[...]

// On récupère l'objet représentant le module (ici connecté en local sur USB)
YAPI.EnableUSBHost(this);
YAPI.RegisterHub("usb");
led = YLed.FindLed("YCTOPOC1-123456.led");

//Pour gérer le hot-plug, on vérifie que le module est là
if (led.isOnline())
{ //Use led.set_power()
  ...
}

[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

YAPI.EnableUSBHost

La fonction `YAPI.EnableUSBHost` initialise l'API avec le Context de l'application courante. Cette fonction prend en argument un objet de la classe `android.content.Context` (ou d'une sous-classe). Si vous comptez uniquement vous connecter à d'autres machines par IP vous cette fonction est facultative.

YAPI.RegisterHub

La fonction `YAPI.RegisterHub` initialise l'API de Yoctopuce en indiquant où les modules doivent être recherchés. Le paramètre est l'adresse du virtual hub capable de voir les modules. Si l'on passe la chaîne de caractère "usb", l'API va travailler avec les modules connectés localement à la machine. Si l'initialisation se passe mal, une exception sera générée.

YLed.FindLed

La fonction `YLed.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):

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

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

isOnline

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

set_power

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

Un exemple réel

Lancez votre environnement java et ouvrez le projet correspondant, fourni dans le répertoire **Exemples/Doc-Exemples** de la librairie Yoctopuce.

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

```
package com.yoctopuce.doc_examples;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;

import com.yoctopuce.YoctoAPI.YAPI;
import com.yoctopuce.YoctoAPI.YAPI_Exception;
import com.yoctopuce.YoctoAPI.YLed;

public class GettingStarted_Yocto_Demo extends Activity implements OnItemClickListener
{

    private YLed led = null;
    private ArrayAdapter<String> aa;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.gettingstarted_yocto_demo);
        Spinner my_spin = (Spinner) findViewById(R.id.spinner1);
        my_spin.setOnItemClickListener(this);
        aa = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item);
        aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        my_spin.setAdapter(aa);
    }

    @Override
    protected void onStart()
    {
        super.onStart();

        try {
            aa.clear();
            YAPI.EnableUSBHost(this);
            YAPI.RegisterHub("usb");
            YLed r = YLed.FirstLed();
            while (r != null) {
                String hwid = r.get_hardwareId();
                aa.add(hwid);
                r = r.nextLed();
            }
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
        // refresh Spinner with detected relay
        aa.notifyDataSetChanged();
    }

    @Override
    protected void onStop()
    {
        super.onStop();
        YAPI.FreeAPI();
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int pos, long id)
    {
        String hwid = parent.getItemAtPosition(pos).toString();
        led = YLed.FindLed(hwid);
    }
}
```

```

@Override
public void onNothingSelected(AdapterView<?> arg0)
{
}

/** Called when the user touches the button State A */
public void setLedOn(View view)
{
    // Do something in response to button click
    if (led != null)
        try {
            led.setPower(YLed.POWER_ON);
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
}

/** Called when the user touches the button State B */
public void setLedOff(View view)
{
    // Do something in response to button click
    if (led != null)
        try {
            led.setPower(YLed.POWER_OFF);
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
}
}

```

2.6. 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.

```

package com.yoctopuce.doc_examples;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Switch;
import android.widget.TextView;

import com.yoctopuce.YoctoAPI.YAPI;
import com.yoctopuce.YoctoAPI.YAPI_Exception;
import com.yoctopuce.YoctoAPI.YModule;

public class ModuleControl extends Activity implements OnItemClickListener
{
    private ArrayAdapter<String> aa;
    private YModule module = null;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.modulecontrol);
        Spinner my_spin = (Spinner) findViewById(R.id.spinner1);
        my_spin.setOnItemClickListener(this);
        aa = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item);
        aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        my_spin.setAdapter(aa);
    }

    @Override

```

```

protected void onStart()
{
    super.onStart();

    try {
        aa.clear();
        YAPI.EnableUSBHost(this);
        YAPI.RegisterHub("usb");
        YModule r = YModule.FirstModule();
        while (r != null) {
            String hwid = r.get_hardwareId();
            aa.add(hwid);
            r = r.nextModule();
        }
    } catch (YAPI_Exception e) {
        e.printStackTrace();
    }
    // refresh Spinner with detected relay
    aa.notifyDataSetChanged();
}

@Override
protected void onStop()
{
    super.onStop();
    YAPI.FreeAPI();
}

private void DisplayModuleInfo()
{
    TextView field;
    if (module == null)
        return;
    try {
        field = (TextView) findViewById(R.id.serialfield);
        field.setText(module.getSerialNumber());
        field = (TextView) findViewById(R.id.logicalnamefield);
        field.setText(module.getLogicalName());
        field = (TextView) findViewById(R.id.luminosityfield);
        field.setText(String.format("%d%", module.getLuminosity()));
        field = (TextView) findViewById(R.id.uptimefield);
        field.setText(module.getUpTime() / 1000 + " sec");
        field = (TextView) findViewById(R.id.usbcurrentfield);
        field.setText(module.getUsbCurrent() + " mA");
        Switch sw = (Switch) findViewById(R.id.beaconswitch);
        Log.d("switch", "beacon" + module.get_beacon());
        sw.setChecked(module.getBeacon() == YModule.BEACON_ON);
        field = (TextView) findViewById(R.id.logs);
        field.setText(module.get_lastLogs());

    } catch (YAPI_Exception e) {
        e.printStackTrace();
    }
}

@Override
public void onItemSelected(AdapterView<?> parent, View view, int pos, long id)
{
    String hwid = parent.getItemAtPosition(pos).toString();
    module = YModule.FindModule(hwid);
    DisplayModuleInfo();
}

@Override
public void onNothingSelected(AdapterView<?> arg0)
{
}

public void refreshInfo(View view)
{
    DisplayModuleInfo();
}

public void toggleBeacon(View view)
{
    if (module == null)
        return;
    boolean on = ((Switch) view).isChecked();

```

```

    try {
        if (on) {
            module.setBeacon(YModule.BEACON_ON);
        } else {
            module.setBeacon(YModule.BEACON_OFF);
        }
    } catch (YAPI_Exception e) {
        e.printStackTrace();
    }
}
}

```

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

Modifications des réglages du module

Lorsque que vous souhaitez modifier les réglages d'un module, il suffit d'appeler la fonction `YModule.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 `YModule.saveToFlash()`. Inversement il est possible de forcer le module à oublier ses réglages courants en utilisant la méthode `YModule.revertFromFlash()`. Ce petit exemple ci-dessous vous permet changer le nom logique d'un module.

```

package com.yoctopuce.doc_examples;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

import com.yoctopuce.YoctoAPI.YAPI;
import com.yoctopuce.YoctoAPI.YAPI_Exception;
import com.yoctopuce.YoctoAPI.YModule;

public class SaveSettings extends Activity implements OnItemClickListener
{
    private ArrayAdapter<String> aa;
    private YModule module = null;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.savesettings);
        Spinner my_spin = (Spinner) findViewById(R.id.spinner1);
        my_spin.setOnItemClickListener(this);
        aa = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item);
        aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        my_spin.setAdapter(aa);
    }

    @Override
    protected void onStart()
    {
        super.onStart();

        try {
            aa.clear();
            YAPI.EnableUSBHost(this);
            YAPI.RegisterHub("usb");

```

```

        YModule r = YModule.FirstModule();
        while (r != null) {
            String hwid = r.get_hardwareId();
            aa.add(hwid);
            r = r.nextModule();
        }
    } catch (YAPI_Exception e) {
        e.printStackTrace();
    }
    // refresh Spinner with detected relay
    aa.notifyDataSetChanged();
}

@Override
protected void onStop()
{
    super.onStop();
    YAPI.FreeAPI();
}

private void DisplayModuleInfo()
{
    TextView field;
    if (module == null)
        return;
    try {
        YAPI.UpdateDeviceList(); // fixme
        field = (TextView) findViewById(R.id.logicalnamefield);
        field.setText(module.getLogicalName());
    } catch (YAPI_Exception e) {
        e.printStackTrace();
    }
}

@Override
public void onItemSelected(AdapterView<?> parent, View view, int pos, long id)
{
    String hwid = parent.getItemAtPosition(pos).toString();
    module = YModule.FindModule(hwid);
    DisplayModuleInfo();
}

@Override
public void onNothingSelected(AdapterView<?> arg0)
{
}

public void saveName(View view)
{
    if (module == null)
        return;

    EditText edit = (EditText) findViewById(R.id.newname);
    String newname = edit.getText().toString();
    try {
        if (!YAPI.CheckLogicalName(newname)) {
            Toast.makeText(getApplicationContext(), "Invalid name (" + newname + ")",
                Toast.LENGTH_LONG).show();
            return;
        }
        module.set_logicalName(newname);
        module.saveToFlash(); // do not forget this
        edit.setText("");
    } catch (YAPI_Exception ex) {
        ex.printStackTrace();
    }
    DisplayModuleInfo();
}
}

```

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 alentours de 100000 cycles. Pour résumer vous ne pouvez employer la fonction `YModule.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 `YModule.yFirstModule()` qui renvoie le premier module trouvé, il suffit ensuite d'appeler la méthode `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 modules connectés

```
package com.yoctopuce.doc_examples;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.TextView;

import com.yoctopuce.YoctoAPI.YAPI;
import com.yoctopuce.YoctoAPI.YAPI_Exception;
import com.yoctopuce.YoctoAPI.YModule;

public class Inventory extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.inventory);
    }

    public void refreshInventory(View view)
    {
        LinearLayout layout = (LinearLayout) findViewById(R.id.inventoryList);
        layout.removeAllViews();

        try {
            YAPI.UpdateDeviceList();
            YModule module = YModule.FirstModule();
            while (module != null) {
                String line = module.get_serialNumber() + " (" + module.get_productName() +
                ")";

                TextView tx = new TextView(this);
                tx.setText(line);
                layout.addView(tx);
                module = module.nextModule();
            }
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    protected void onStart()
    {
        super.onStart();
        try {
            YAPI.EnableUSBHost(this);
            YAPI.RegisterHub("usb");
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
        refreshInventory(null);
    }

    @Override
    protected void onStop()
    {
        super.onStop();
        YAPI.FreeAPI();
    }
}
```


2.7. 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.

Dans l'API java pour Android, le traitement d'erreur est implémenté au moyen d'exceptions. Vous devrez donc intercepter et traiter correctement ces exceptions si vous souhaitez avoir un projet fiable qui ne crashera pas dès que vous débrancherez un module.

3. Reference

3.1. Fonctions générales

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

yCheckLogicalName(name)

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

yDisableExceptions()

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

yEnableExceptions()

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

yEnableUSBHost(osContext)

Cette fonction est utilisée uniquement sous Android.

yFreeAPI()

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

yGetAPIVersion()

Retourne la version de la librairie Yoctopuce utilisée.

yGetTickCount()

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

yHandleEvents(errmsg)

Maintient la communication de la librairie avec les modules Yoctopuce.

yInitAPI(mode, errmsg)

Initialise la librairie de programmation de Yoctopuce explicitement.

yPreregisterHub(url, errmsg)

Alternative plus tolérante à `RegisterHub()`.

yRegisterDeviceArrivalCallback(arrivalCallback)

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

yRegisterDeviceRemovalCallback(removalCallback)

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

yRegisterHub(url, errmsg)

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

yRegisterHubDiscoveryCallback(hubDiscoveryCallback)

3. Reference

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

yRegisterLogFunction(logfun)

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

ySelectArchitecture(arch)

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

ySetDelegate(object)

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

ySetTimeout(callback, ms_timeout, arguments)

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

ySleep(ms_duration, errmsg)

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

yTriggerHubDiscovery(errmsg)

Relance une détection des hubs réseau.

yUnregisterHub(url)

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

yUpdateDeviceList(errmsg)

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

yUpdateDeviceList_async(callback, context)

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

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

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

`boolean` **CheckLogicalName**(`String` `name`)

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

Paramètres :

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

Retourne :

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

YAPI.EnableUSBHost()**YAPI****yEnableUSBHost()**`YAPI.EnableUSBHost ()`

Cette fonction est utilisée uniquement sous Android.

```
void EnableUSBHost( Object osContext)
```

Avant d'appeler `yRegisterHub("usb")` il faut activer le port USB host du systeme. Cette fonction prend en argument un objet de la classe `android.content.Context` (ou d'une sous-classe). Il n'est pas nécessaire d'appeler cette fonction pour accéder au modules à travers le réseau.

Paramètres :

osContext un objet de classe `android.content.Context` (ou une sous-classe)

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

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

void FreeAPI()

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

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

Retourne la version de la librairie Yoctopuce utilisée.

String **GetAPIVersion()**

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

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

Retourne :

une chaîne de caractères décrivant la version de la librairie.

YAPI.GetTickCount()**YAPI****yGetTickCount()**`YAPI.GetTickCount()`

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

long **GetTickCount()**

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

Retourne :

un long entier contenant la valeur du compteur de millisecondes.

YAPI.HandleEvents()**YAPI****yHandleEvents()**`YAPI.HandleEvents()`

Maintient la communication de la librairie avec les modules Yoctopuce.

int `HandleEvents()`

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

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

YAPI.InitAPI()**YAPI****yInitAPI()**`YAPI.InitAPI()`

Initialise la librairie de programmation de Yoctopuce explicitement.

```
int InitAPI( int mode)
```

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

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

Paramètres :

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

Retourne :

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

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

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

Alternative plus tolérante à RegisterHub().

```
int PreregisterHub( String url)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

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

```
void RegisterDeviceArrivalCallback( DeviceArrivalCallback arrivalCallback)
```

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

Paramètres :

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

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

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

```
void RegisterDeviceRemovalCallback( DeviceRemovalCallback removalCallback)
```

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

Paramètres :

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

YAPI.RegisterHub()**YAPI****yRegisterHub()**`YAPI.RegisterHub()`

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

```
int RegisterHub( String url)
```

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

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

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

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

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

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

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

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

Paramètres :

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

Retourne :

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

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

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

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

```
void RegisterHubDiscoveryCallback( HubDiscoveryCallback hubDiscoveryCallback)
```

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

Paramètres :

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

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

YAPI

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

```
void RegisterLogFunction( LogCallback logfun)
```

Utile pour débbugger le fonctionnement de l'API.

Paramètres :

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

YAPI.Sleep()**YAPI****ySleep()**`YAPI.Sleep()`

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

```
int Sleep( long ms_duration)
```

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

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

Paramètres :

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

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

Retourne :

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

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

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

YAPI

Relance une détection des hubs réseau.

int **TriggerHubDiscovery()**

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

Paramètres :

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

Retourne :

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

YAPI.UnregisterHub()

YAPI

yUnregisterHub() `YAPI.UnregisterHub()`

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

```
void UnregisterHub( String url)
```

Paramètres :

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

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

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

`int UpdateDeviceList()`

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

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

Paramètres :

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

Retourne :

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

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

3.2. Interface de la fonction Accelerometer

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes 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');
c++	#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)`

3. Reference

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

accelerometer→registerValueCallback(callback)

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

accelerometer→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

accelerometer→set_logFrequency(newval)

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

accelerometer→set_logicalName(newval)

Modifie le nom logique de l'accéléromètre.

accelerometer→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

accelerometer→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

accelerometer→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

accelerometer→set_userData(data)

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

accelerometer→wait_async(callback, context)

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

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

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

`YAccelerometer` **FindAccelerometer**(`String func`)

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

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

Retourne :

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

YAccelerometer.FirstAccelerometer()

YAccelerometer

yFirstAccelerometer()

YAccelerometer.FirstAccelerometer()

Commence l'énumération des accéléromètres accessibles par la librairie.

YAccelerometer **FirstAccelerometer()**

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

Retourne :

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

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

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

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

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

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

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

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

String describe()

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

Retourne :

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

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

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

String **get_advertisedValue()**

Retourne :

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

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

accelerometer→**get_currentRawValue()**

YAccelerometer

accelerometer→**currentRawValue()**

accelerometer.get_currentRawValue()

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

double **get_currentRawValue()**

Retourne :

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

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

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

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

double **get_currentValue()**

Retourne :

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

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

accelerometer→get_errorMessage()

YAccelerometer

accelerometer→errorMessage()

accelerometer.get_errorMessage()

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

String get_errorMessage()

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

Retourne :

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

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

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

```
int get_errorType( )
```

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

Retourne :

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

accelerometer→get_friendlyName()

YAccelerometer

accelerometer→friendlyName()

accelerometer.get_friendlyName()

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

String **get_friendlyName()**

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

Retourne :

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

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

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

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

String **get_functionDescriptor()**

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

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

accelerometer→**get_functionId()**

YAccelerometer

accelerometer→**functionId()**

accelerometer.get_functionId()

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

String **get_functionId()** ()

Par exemple relay1.

Retourne :

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

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

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

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

`String` **get_hardwareId()**

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

Retourne :

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

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

accelerometer→get_highestValue()

YAccelerometer

accelerometer→highestValue()

accelerometer.get_highestValue()

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

double get_highestValue()

Retourne :

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

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

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

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

String **get_logFrequency()**

Retourne :

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

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

accelerometer→get_logicalName()

YAccelerometer

accelerometer→logicalName()

accelerometer.get_logicalName()

Retourne le nom logique de l'accéléromètre.

String get_logicalName()

Retourne :

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

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

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

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

double **get_lowestValue()**

Retourne :

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

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

accelerometer→get_module()

YAccelerometer

accelerometer→module()

accelerometer.get_module()

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

YModule **get_module()**

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

Retourne :

une instance de YModule

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

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

YDataSet **get_recordedData(** long **startTime**, long **endTime****)**

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

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

Paramètres :

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

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

Retourne :

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

accelerometer→get_reportFrequency()

YAccelerometer

accelerometer→reportFrequency()

accelerometer.get_reportFrequency()

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

String **get_reportFrequency()**

Retourne :

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

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

accelerometer→**get_resolution()****YAccelerometer****accelerometer**→**resolution()****accelerometer.get_resolution()**

Retourne la résolution des valeurs mesurées.

double **get_resolution()** ()

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

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

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

accelerometer→**get_unit()**

YAccelerometer

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

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

String **get_unit()** ()

Retourne :

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

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

accelerometer→get_userdata()**YAccelerometer****accelerometer→userData()****accelerometer.get_userdata()**

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

Object **get_userdata()**

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

Retourne :

l'objet stocké précédemment par l'appelant.

accelerometer→get_xValue()

YAccelerometer

accelerometer→xValue()

accelerometer.get_xValue()

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

double **get_xValue()** ()

Retourne :

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

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

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

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

double **get_yValue()**

Retourne :

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

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

accelerometer→get_zValue()

YAccelerometer

accelerometer→zValue()

accelerometer.get_zValue()

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

double **get_zValue()**

Retourne :

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

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

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

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

boolean **isOnline**()

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

Retourne :

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

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

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

```
int load( long msValidity)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

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

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

accelerometer→**nextAccelerometer()**

YAccelerometer

accelerometer.nextAccelerometer()

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

`YAccelerometer` **nextAccelerometer()**

Retourne :

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

accelerometer→registerTimedReportCallback()**YAccelerometer**

```
accelerometer.registerTimedReportCallback(  
)
```

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

```
int registerTimedReportCallback( TimedReportCallback callback)
```

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

Paramètres :

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

accelerometer→registerValueCallback()

YAccelerometer

accelerometer.registerValueCallback()

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

```
int registerValueCallback( UpdateCallback callback)
```

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

Paramètres :

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

accelerometer→set_highestValue()**YAccelerometer****accelerometer→setHighestValue()****accelerometer.set_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

accelerometer→set_logFrequency()

YAccelerometer

accelerometer→setLogFrequency()

accelerometer.set_logFrequency()

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

```
int set_logFrequency( String newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

accelerometer→**set_logicalName()**
accelerometer→**setLogicalName()**
accelerometer.set_logicalName()

YAccelerometer

Modifie le nom logique de l'accéléromètre.

```
int set_logicalName( String newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

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

accelerometer→set_lowestValue()

YAccelerometer

accelerometer→setLowestValue()

accelerometer.set_lowestValue()

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

accelerometer→set_reportFrequency()**YAccelerometer****accelerometer→setReportFrequency()****accelerometer.set_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

accelerometer→**set_resolution()**

YAccelerometer

accelerometer→**setResolution()**

accelerometer.set_resolution()

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

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

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

accelerometer→set_userdata()**YAccelerometer****accelerometer→setUserData()****accelerometer.set_userdata()**

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

```
void set_userdata( Object data)
```

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

Paramètres :

data objet quelconque à mémoriser

3.3. Interface de la fonction Altitude

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_altitude.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YAltitude = yoctolib.YAltitude;
php	require_once('yocto_altitude.php');
c++	#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)

3. Reference

Modifie l'altitude actuelle supposée.

altitude→**set_highestValue(newval)**

Modifie la mémoire de valeur maximale observée.

altitude→**set_logFrequency(newval)**

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

altitude→**set_logicalName(newval)**

Modifie le nom logique de l'altimètre.

altitude→**set_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

altitude→**set_qnh(newval)**

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

altitude→**set_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

altitude→**set_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

altitude→**set_userData(data)**

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

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

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

YAltitude.FindAltitude()**YAltitude****yFindAltitude()**`YAltitude.FindAltitude()`

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

```
YAltitude FindAltitude( String func)
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

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

Retourne :

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

YAltitude.FirstAltitude()

YAltitude

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

Commence l'énumération des altimètres accessibles par la librairie.

YAltitude **FirstAltitude()**

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

Retourne :

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

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

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

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                        ArrayList<Double> refValues)
```

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

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

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

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

String describe()

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

Retourne :

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

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

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

String **get_advertisedValue()**

Retourne :

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

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

altitude→**get_currentRawValue()**

YAltitude

altitude→**currentRawValue()**

altitude.get_currentRawValue()

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

double **get_currentRawValue()**

Retourne :

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

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

altitude→**get_currentValue()**
altitude→**currentValue()**
altitude.get_currentValue()

YAltitude

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

```
double get_currentValue()
```

Retourne :

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

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

altitude→**get_errorMessage()**

YAltitude

altitude→**errorMessage()**

altitude.get_errorMessage()

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

String **get_errorMessage()**

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

Retourne :

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

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

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

int **get_errorType()**

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

Retourne :

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

altitude→get_friendlyName()

YAltitude

altitude→friendlyName()

altitude.get_friendlyName()

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

String get_friendlyName()

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

Retourne :

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

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

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

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

String **get_functionDescriptor()**

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

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

altitude→**get_functionId()**

YAltitude

altitude→**functionId()****altitude.get_functionId()**

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

String **get_functionId()**

Par exemple `relay1`.

Retourne :

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

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

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

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

String **get_hardwareId()**

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

Retourne :

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

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

altitude→**get_highestValue()**

YAltitude

altitude→**highestValue()**

altitude.get_highestValue()

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

double **get_highestValue()**

Retourne :

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

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

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

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.

String **get_logFrequency()**

Retourne :

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

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

altitude→get_logicalName()

YAltitude

altitude→logicalName()

altitude.get_logicalName()

Retourne le nom logique de l'altimètre.

String **get_logicalName()**

Retourne :

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

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

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

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

double **get_lowestValue()**

Retourne :

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

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

altitude→**get_module()**

YAltitude

altitude→**module()**`altitude.get_module()`

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

`YModule` **get_module()**

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

Retourne :

une instance de `YModule`

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

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

double **get_qnh()**

Retourne :

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

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

altitude→get_recordedData()

YAltitude

altitude→recordedData()

altitude.get_recordedData()

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

YDataSet **get_recordedData**(long **startTime**, long **endTime**)

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

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

Paramètres :

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

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

Retourne :

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

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

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

String **get_reportFrequency()**

Retourne :

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

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

altitude→**get_resolution()**

YAltitude

altitude→**resolution()****altitude.get_resolution()**

Retourne la résolution des valeurs mesurées.

double **get_resolution()** ()

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

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

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

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

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

String **get_unit()**

Retourne :

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

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

altitude→**get_userData()**

YAltitude

altitude→**userData()****altitude.userData()**

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

Object **get_userData()**

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

Retourne :

l'objet stocké précédemment par l'appelant.

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

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

boolean **isOnline**()

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

Retourne :

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

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

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

```
int load( long msValidity)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

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

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

YAltitude

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

YAltitude **nextAltitude()**

Retourne :

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

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

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

```
int registerTimedReportCallback( TimedReportCallback callback)
```

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

Paramètres :

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

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

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

```
int registerValueCallback( UpdateCallback callback)
```

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

Paramètres :

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

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

YAltitude

Modifie l'altitude actuelle supposée.

```
int set_currentValue( double newval)
```

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

Paramètres :

newval une valeur numérique représentant l'altitude actuelle supposée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→**set_highestValue()**

YAltitude

altitude→**setHighestValue()**

altitude.set_highestValue()

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→**set_logFrequency()**
altitude→**setLogFrequency()**
altitude.set_logFrequency()

YAltitude

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

```
int set_logFrequency( String newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→**set_logicalName()**

YAltitude

altitude→**setLogicalName()**

altitude.set_logicalName()

Modifie le nom logique de l'altimètre.

```
int set_logicalName( String newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

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

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

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→**set_qnh()**

YAltitude

altitude→**setQnh()****altitude.set_qnh()**

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

```
int set_qnh( double newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→set_reportFrequency()

YAltitude

altitude→setReportFrequency()

altitude.set_reportFrequency()

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→**set_resolution()**

YAltitude

altitude→**setResolution()**

altitude.set_resolution()

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

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

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→**set_userdata()****YAltitude****altitude**→**setUserData()****altitude.set_userdata()**

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

```
void set_userdata( Object data )
```

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

Paramètres :

data objet quelconque à mémoriser

3.4. Interface de la fonction AnButton

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

yFindAnButton(func)

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

yFirstAnButton()

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

Méthodes des objets YAnButton

anbutton→describe()

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

anbutton→get_advertisedValue()

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

anbutton→get_analogCalibration()

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

anbutton→get_calibratedValue()

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

anbutton→get_calibrationMax()

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

anbutton→get_calibrationMin()

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

anbutton→get_errorMessage()

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

anbutton→get_errorType()

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

anbutton→get_friendlyName()

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

anbutton→get_functionDescriptor()

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

`anbutton→get_functionId()`

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

`anbutton→get_hardwareId()`

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

`anbutton→get_isPressed()`

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

`anbutton→get_lastTimePressed()`

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

`anbutton→get_lastTimeReleased()`

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

`anbutton→get_logicalName()`

Retourne le nom logique de l'entrée analogique.

`anbutton→get_module()`

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

`anbutton→get_module_async(callback, context)`

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

`anbutton→get_pulseCounter()`

Retourne la valeur du compteur d'impulsions.

`anbutton→get_pulseTimer()`

Retourne le timer du compteur d'impulsions (ms)

`anbutton→get_rawValue()`

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

`anbutton→get_sensitivity()`

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

`anbutton→get_userData()`

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

`anbutton→isOnline()`

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

`anbutton→isOnline_async(callback, context)`

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

`anbutton→load(msValidity)`

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

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

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

`anbutton→nextAnButton()`

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

`anbutton→registerValueCallback(callback)`

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

`anbutton→resetCounter()`

réinitialise le compteur d'impulsions et son timer

`anbutton→set_analogCalibration(newval)`

Enclenche ou déclenche le procédure de calibration.

`anbutton→set_calibrationMax(newval)`

3. Reference

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

anbutton→**set_calibrationMin(newval)**

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

anbutton→**set_logicalName(newval)**

Modifie le nom logique de l'entrée analogique.

anbutton→**set_sensitivity(newval)**

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

anbutton→**set_userData(data)**

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

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

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

YAnButton.FindAnButton()**YAnButton****yFindAnButton()**`YAnButton.FindAnButton()`

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

`YAnButton FindAnButton(String func)`

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

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

Retourne :

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

YAnButton.FirstAnButton()

YAnButton

yFirstAnButton()`YAnButton.FirstAnButton()`

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

`YAnButton` **FirstAnButton()**

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

Retourne :

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

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

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

String describe()

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

Retourne :

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

anbutton→get_advertisedValue()

YAnButton

anbutton→advertisedValue()

anbutton.get_advertisedValue()

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

String **get_advertisedValue()**

Retourne :

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

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

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

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

int **get_analogCalibration()** ()

Retourne :

soit Y_ANALOGCALIBRATION_OFF, soit Y_ANALOGCALIBRATION_ON

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

anbutton→**get_calibratedValue()**

YAnButton

anbutton→**calibratedValue()**

anbutton.get_calibratedValue()

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

int **get_calibratedValue()**

Retourne :

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

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

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

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

int **get_calibrationMax()**

Retourne :

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

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

anbutton→**get_calibrationMin()**

YAnButton

anbutton→**calibrationMin()**

anbutton.get_calibrationMin()

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

int **get_calibrationMin()**

Retourne :

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

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

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

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

String **errorMessage()**

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

Retourne :

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

anbutton→**get_errorType()**

YAnButton

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

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

int **get_errorType()**

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

Retourne :

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

anbutton→get_friendlyName()
anbutton→friendlyName()
anbutton.get_friendlyName()

YAnButton

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

String `get_friendlyName()`

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

Retourne :

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

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

anbutton→get_functionDescriptor()

YAnButton

anbutton→functionDescriptor()

anbutton.get_functionDescriptor()

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

String **get_functionDescriptor()**

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

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

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

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

String **get_functionId()** ()

Par exemple `relay1`.

Retourne :

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

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

anbutton→**get_hardwareId()**

YAnButton

anbutton→**hardwareId()**

anbutton.get_hardwareId()

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

String **get_hardwareId()** ()

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

Retourne :

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

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

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

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

int **get_isPressed()**

Retourne :

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

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

anbutton→**get_lastTimePressed()**

YAnButton

anbutton→**lastTimePressed()**

anbutton.get_lastTimePressed()

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).

long **get_lastTimePressed()**

Retourne :

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

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

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

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

```
long get_lastTimeReleased( )
```

Retourne :

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

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

anbutton→get_logicalName()

YAnButton

anbutton→logicalName()

anbutton.get_logicalName()

Retourne le nom logique de l'entrée analogique.

String **get_logicalName()**

Retourne :

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

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

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

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

`YModule` **get_module()**

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

Retourne :

une instance de `YModule`

anbutton→get_pulseCounter()

YAnButton

anbutton→pulseCounter()

anbutton.get_pulseCounter()

Retourne la valeur du compteur d'impulsions.

long get_pulseCounter()

Retourne :

un entier représentant la valeur du compteur d'impulsions

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

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

Retourne le timer du compteur d'impulsions (ms)

```
long get_pulseTimer( )
```

Retourne :

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

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

anbutton→**get_rawValue()**

YAnButton

anbutton→**rawValue()****anbutton.getRawValue()**

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

int **get_rawValue()**

Retourne :

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

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

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

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

int **get_sensitivity**()

Retourne :

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

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

anbutton→**get_userData()**

YAnButton

anbutton→**userData()****anbutton.userData()**

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

Object **get_userdata()**

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

Retourne :

l'objet stocké précédemment par l'appelant.

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

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

boolean **isOnline()**

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

Retourne :

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

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

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

```
int load( long msValidity)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

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

`YAnButton` **nextAnButton()**

Retourne :

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

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

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

```
int registerValueCallback( UpdateCallback callback)
```

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

Paramètres :

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

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

réinitialise le compteur d'impulsions et son timer

int resetCounter()

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

anbutton→**set_analogCalibration()**

YAnButton

anbutton→**setAnalogCalibration()**

anbutton.set_analogCalibration()

Enclenche ou déclenche le procédure de calibration.

```
int set_analogCalibration( int newval)
```

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

Paramètres :

newval soit `Y_ANALOGCALIBRATION_OFF`, soit `Y_ANALOGCALIBRATION_ON`

Retourne :

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

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

anbutton→set_calibrationMax()**YAnButton****anbutton→setCalibrationMax()****anbutton.set_calibrationMax()**

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

```
int set_calibrationMax( int newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

anbutton→**set_calibrationMin()**

YAnButton

anbutton→**setCalibrationMin()**

anbutton.set_calibrationMin()

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

```
int set_calibrationMin( int newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

Modifie le nom logique de l'entrée analogique.

```
int set_logicalName( String newval)
```

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

Paramètres :

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

Retourne :

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

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

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

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

```
int set_sensitivity( int newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

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

```
void set_userdata( Object data)
```

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

Paramètres :

data objet quelconque à mémoriser

3.5. Interface de la fonction CarbonDioxide

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_carbondioxide.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YCarbonDioxide = yoctolib.YCarbonDioxide;
php	require_once('yocto_carbondioxide.php');
c++	#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)	

3. Reference

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.

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

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

YCarbonDioxide **FindCarbonDioxide**(String **func**)

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

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

Retourne :

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

YCarbonDioxide.FirstCarbonDioxide()

YCarbonDioxide

yFirstCarbonDioxide()

YCarbonDioxide.FirstCarbonDioxide()

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

YCarbonDioxide **FirstCarbonDioxide()**

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

Retourne :

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

carbondioxide→calibrateFromPoints() carbondioxide.calibrateFromPoints()

YCarbonDioxide

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

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                        ArrayList<Double> refValues)
```

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

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

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

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

String describe()

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

Retourne :

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

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

YCarbonDioxide

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

String **get_advertisedValue()**

Retourne :

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

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

carbondioxide→get_currentRawValue()

YCarbonDioxide

carbondioxide→currentRawValue()

carbondioxide.get_currentRawValue()

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

double **get_currentRawValue()**

Retourne :

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

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

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

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

double **get_currentValue()**

Retourne :

une valeur numérique représentant la valeur actuelle du taux de CO2, en ppm (val), sous forme de nombre à virgule

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

carbondioxide→get_errorMessage()

YCarbonDioxide

carbondioxide→errorMessage()

carbondioxide.get_errorMessage()

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

String get_errorMessage()

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

Retourne :

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

carbondioxide→get_errorType()**YCarbonDioxide****carbondioxide→errorType()****carbondioxide.get_errorType()**

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

int get_errorType()

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

Retourne :

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

carbondioxide→get_friendlyName()

YCarbonDioxide

carbondioxide→friendlyName()

carbondioxide.get_friendlyName()

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

String get_friendlyName()

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

Retourne :

une chaîne de caractères identifiant le capteur de CO2 en utilisant les noms logiques (ex: `MyCustomName.relay1`)

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

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

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

String **get_functionDescriptor()**

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

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

carbondioxide→get_functionId()

YCarbonDioxide

carbondioxide→functionId()

carbondioxide.get_functionId()

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

String get_functionId()

Par exemple relay1.

Retourne :

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

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

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

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

String **get_hardwareId()**

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

Retourne :

une chaîne de caractères identifiant le capteur de CO2 (ex: `RELAYLO1-123456.relay1`)

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

carbondioxide→get_highestValue()

YCarbonDioxide

carbondioxide→highestValue()

carbondioxide.get_highestValue()

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

double **get_highestValue()**

Retourne :

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

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

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

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

String **get_logFrequency()**

Retourne :

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

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

carbondioxide→get_logicalName()

YCarbonDioxide

carbondioxide→logicalName()

carbondioxide.get_logicalName()

Retourne le nom logique du capteur de CO2.

String **get_logicalName()**

Retourne :

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

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

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

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

double **get_lowestValue()**

Retourne :

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

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

carbondioxide→get_module()

YCarbonDioxide

carbondioxide→module()

carbondioxide.get_module()

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

YModule **get_module()**

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

Retourne :

une instance de YModule

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

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

YDataSet **get_recordedData(** long **startTime**, long **endTime****)**

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

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

Paramètres :

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

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

Retourne :

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

carbondioxide→get_reportFrequency()

YCarbonDioxide

carbondioxide→reportFrequency()

carbondioxide.get_reportFrequency()

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

String **get_reportFrequency()**

Retourne :

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

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

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

Retourne la résolution des valeurs mesurées.

double **get_resolution()**

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

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

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

carbondioxide→**get_unit()**

YCarbonDioxide

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

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

String **get_unit()**

Retourne :

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

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

carbondioxide→get_userdata()**YCarbonDioxide****carbondioxide→userData()****carbondioxide.get_userdata()**

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

Object **get_userdata()**

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

Retourne :

l'objet stocké précédemment par l'appelant.

carbondioxide→isOnline()

YCarbonDioxide

carbondioxide.isOnline()

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

boolean **isOnline()**

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

Retourne :

true si le capteur de CO2 est joignable, false sinon

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

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

```
int load( long msValidity)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

carbondioxide→loadCalibrationPoints()

YCarbonDioxide

carbondioxide.loadCalibrationPoints()

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

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

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

`YCarbonDioxide` **nextCarbonDioxide()**

Retourne :

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

carbondioxide→registerTimedReportCallback()

YCarbonDioxide

**carbondioxide.registerTimedReportCallback(
)**

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

```
int registerTimedReportCallback( TimedReportCallback callback)
```

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

Paramètres :

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

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

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

```
int registerValueCallback( UpdateCallback callback)
```

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

Paramètres :

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

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

YCarbonDioxide

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

YCarbonDioxide

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

```
int set_logFrequency( String newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

YCarbonDioxide

Modifie le nom logique du capteur de CO2.

```
int set_logicalName( String newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

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

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

YCarbonDioxide

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

carbondioxide→set_reportFrequency()

YCarbonDioxide

carbondioxide→setReportFrequency()

carbondioxide.set_reportFrequency()

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

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

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

carbondioxide→set_userdata()

YCarbonDioxide

carbondioxide→setUserData()

carbondioxide.set_userdata()

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

void **set_userdata**(Object **data**)

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

Paramètres :

data objet quelconque à mémoriser

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');
c++	#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 affichée à l'allumage du module.

colorled→**get_userData()**

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

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

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

colorled→**isOnline()**

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

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

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

colorled→**load(msValidity)**

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

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

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

colorled→**nextColorLed()**

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

colorled→**registerValueCallback(callback)**

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

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

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

colorled→**set_hslColor(newval)**

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

colorled→**set_logicalName(newval)**

Modifie le nom logique de la led RGB.

colorled→**set_rgbColor(newval)**

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

colorled→**set_rgbColorAtPowerOn(newval)**

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

colorled→**set_userData(data)**

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

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

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

YColorLed.FindColorLed()**YColorLed****yFindColorLed()YColorLed.FindColorLed()**

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

YColorLed **FindColorLed**(String **func**)

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

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

Retourne :

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

YColorLed.FirstColorLed()

YColorLed

yFirstColorLed()`YColorLed.FirstColorLed()`

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

`YColorLed` **FirstColorLed()**

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

Retourne :

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

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

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

String describe()

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

Retourne :

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

colorled→**get_advertisedValue()**

YColorLed

colorled→**advertisedValue()**

colorled.get_advertisedValue()

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

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante de la led RGB (pas plus de 6 caractères).

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

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

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

String **get_errorMessage()**

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

Retourne :

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

colorled→**get_errorType()**

YColorLed

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

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

```
int get_errorType( )
```

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

Retourne :

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

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

YColorLed

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

String **get_friendlyName()**

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

Retourne :

une chaîne de caractères identifiant la led RGB en utilisant les noms logiques (ex: `MyCustomName.relay1`)

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

colorled→**get_functionDescriptor()**

YColorLed

colorled→**functionDescriptor()**

colorled.get_functionDescriptor()

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

String **get_functionDescriptor()**

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

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

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

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

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant la led RGB (ex: `relay1`)

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

colorled→**get_hardwareId()**

YColorLed

colorled→**hardwareId()**

colorled.get_hardwareId()

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

String **get_hardwareId()** ()

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

Retourne :

une chaîne de caractères identifiant la led RGB (ex: `RELAYLO1-123456.relay1`)

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

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

Retourne la couleur HSL courante de la led.

int **get_hslColor()**

Retourne :

un entier représentant la couleur HSL courante de la led

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

colorled→**get_logicalName()**

YColorLed

colorled→**logicalName()**

colorled.get_logicalName()

Retourne le nom logique de la led RGB.

String **get_logicalName()**

Retourne :

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

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

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

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

YModule **get_module()**

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

Retourne :

une instance de YModule

colorled→**get_rgbColor()**

YColorLed

colorled→**rgbColor()****colorled.get_rgbColor()**

Retourne la couleur RGB courante de la led.

int **get_rgbColor()** ()

Retourne :

un entier représentant la couleur RGB courante de la led

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

colorled→**get_rgbColorAtPowerOn()****YColorLed****colorled**→**rgbColorAtPowerOn()****colorled.get_rgbColorAtPowerOn()**

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

int **get_rgbColorAtPowerOn()**

Retourne :

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

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

colorled→**get_userdata()**

YColorLed

colorled→**userData()****colorled.userData()**

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

Object **get_userdata()**

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

Retourne :

l'objet stocké précédemment par l'appelant.

colorled→**hslMove()****colorled.hslMove()****YColorLed**

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

```
int hslMove( int hsl_target, int ms_duration)
```

Paramètres :

hsl_target couleur HSL désirée à la fin de la transition

ms_duration durée de la transition, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→**isOnline()****colorled.isOnline()**

YColorLed

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

boolean **isOnline**()

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

Retourne :

`true` si la led RGB est joignable, `false` sinon

colorled→**load()**`colorled.load()`**YColorLed**

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

```
int load( long msValidity)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→**nextColorLed()**

YColorLed

colorled.nextColorLed()

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

`YColorLed` **nextColorLed()**

Retourne :

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

colorled→registerValueCallback()**YColorLed****colorled.registerValueCallback()**

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

```
int registerValueCallback( UpdateCallback callback)
```

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

Paramètres :

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

colorled→**rgbMove()****colorled.rgbMove()****YColorLed**

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

```
int rgbMove( int rgb_target, int ms_duration)
```

Paramètres :

rgb_target couleur RGB désirée à la fin de la transition

ms_duration durée de la transition, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→**set_hslColor()****YColorLed****colorled**→**setHslColor()****colorled.set_hslColor()**

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

```
int set_hslColor( int newval)
```

L'encodage est réalisé de la manière suivante: 0xHHSSL.

Paramètres :

newval un entier représentant la couleur courante de la led, en utilisant une couleur HSL spécifiée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→**set_logicalName()**

YColorLed

colorled→**setLogicalName()**

colorled.set_logicalName()

Modifie le nom logique de la led RGB.

```
int set_logicalName( String newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

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

colorled→**set_rgbColor()****YColorLed****colorled**→**setRgbColor()****colorled.set_rgbColor()**

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

```
int set_rgbColor( int newval)
```

L'encodage est réalisé de la manière suivante: 0xRRGGBB.

Paramètres :

newval un entier représentant la couleur courante de la led, en utilisant une couleur RGB (Rouge Vert Bleu)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→**set_rgbColorAtPowerOn()**

YColorLed

colorled→**setRgbColorAtPowerOn()**

colorled.set_rgbColorAtPowerOn()

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

```
int set_rgbColorAtPowerOn( int newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→**set_userdata()****YColorLed****colorled**→**setUserData()****colorled.set_userdata()**

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

```
void set_userdata( Object data )
```

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

Paramètres :

data objet quelconque à mémoriser

3.7. Interface de la fonction Compass

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

yFindCompass(func)

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

yFirstCompass()

Commence l'énumération des compas accessibles par la librairie.

Méthodes des objets YCompass

compass→calibrateFromPoints(rawValues, refValues)

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

compass→describe()

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

compass→get_advertisedValue()

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

compass→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), 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, indépendamment du cap configuré.

compass→get_module()

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

compass→get_module_async(callback, context)

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

compass→get_recordedData(startTime, endTime)

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

compass→get_reportFrequency()

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

compass→get_resolution()

Retourne la résolution des valeurs mesurées.

compass→get_unit()

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

compass→get_userData()

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

compass→isOnline()

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

compass→isOnline_async(callback, context)

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

compass→load(msValidity)

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

compass→loadCalibrationPoints(rawValues, refValues)

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

compass→load_async(msValidity, callback, context)

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

compass→nextCompass()

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

compass→registerTimedReportCallback(callback)

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

compass→registerValueCallback(callback)

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

compass→set_highestValue(newval)

3. Reference

Modifie la mémoire de valeur maximale observée.

compass→**set_logFrequency(newval)**

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

compass→**set_logicalName(newval)**

Modifie le nom logique du compas.

compass→**set_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

compass→**set_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

compass→**set_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

compass→**set_userData(data)**

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

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

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

YCompass.FindCompass()**YCompass****yFindCompass()**`YCompass.FindCompass()`

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

`YCompass FindCompass(String func)`

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

func une chaîne de caractères qui référence le compas sans ambiguïté

Retourne :

un objet de classe `YCompass` qui permet ensuite de contrôler le compas.

YCompass.FirstCompass()

YCompass

yFirstCompass()`YCompass.FirstCompass()`

Commence l'énumération des compas accessibles par la librairie.

`YCompass` **FirstCompass()**

Utiliser la fonction `YCompass.nextCompass()` pour itérer sur les autres compas.

Retourne :

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

compass→calibrateFromPoints()**YCompass****compass.calibrateFromPoints()**

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

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                        ArrayList<Double> refValues)
```

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

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

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

compass→describe()`compass.describe()`**YCompass**

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

String describe()

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

Retourne :

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

compass→**get_advertisedValue()****YCompass****compass**→**advertisedValue()****compass.get_advertisedValue()**

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

String **get_advertisedValue()**

Retourne :

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

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

compass→**get_currentRawValue()**

YCompass

compass→**currentRawValue()**

compass.get_currentRawValue()

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

double **get_currentRawValue()**

Retourne :

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

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

compass→**get_currentValue()****YCompass****compass**→**currentValue()****compass.get_currentValue()**

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

double **get_currentValue()**

Retourne :

une valeur numérique représentant la valeur actuelle du cap relatif, en degrés, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

compass→get_errorMessage()

YCompass

compass→errorMessage()

compass.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du compas.

String get_errorMessage()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du compas.

compass→**get_errorType()****YCompass****compass**→**errorType()****compass.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du compas.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du compas.

compass→**get_friendlyName()**

YCompass

compass→**friendlyName()**

compass.get_friendlyName()

Retourne un identifiant global du compas au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du compas si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du compas (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le compas en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

compass→**get_functionDescriptor()****YCompass****compass**→**functionDescriptor()****compass.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

compass→**get_functionId()**

YCompass

compass→**functionId()**`compass.get_functionId()`

Retourne l'identifiant matériel du compas, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le compas (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

compass→**get_hardwareId()**
compass→**hardwareId()**
compass.get_hardwareId()

YCompass

Retourne l'identifiant matériel unique du compas au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du compas (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le compas (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

compass→**get_highestValue()**

YCompass

compass→**highestValue()**

compass.get_highestValue()

Retourne la valeur maximale observée pour le cap relatif depuis le démarrage du module.

double **get_highestValue()**

Retourne :

une valeur numérique représentant la valeur maximale observée pour le cap relatif depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

compass→**get_logFrequency()****YCompass****compass**→**logFrequency()****compass.get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

String **get_logFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

compass→**get_logicalName()**

YCompass

compass→**logicalName()**

compass.get_logicalName()

Retourne le nom logique du compas.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du compas.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

compass→**get_lowestValue()****YCompass****compass**→**lowestValue()****compass.get_lowestValue()**

Retourne la valeur minimale observée pour le cap relatif depuis le démarrage du module.

double **get_lowestValue()**

Retourne :

une valeur numérique représentant la valeur minimale observée pour le cap relatif depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

compass→**get_magneticHeading()**

YCompass

compass→**magneticHeading()**

compass.get_magneticHeading()

Retourne la direction du nord magnétique, indépendamment du cap configuré.

double **get_magneticHeading()**

Retourne :

une valeur numérique représentant la direction du nord magnétique, indépendamment du cap configuré

En cas d'erreur, déclenche une exception ou retourne Y_MAGNETICHEADING_INVALID.

compass→**get_module()****YCompass****compass**→**module()**`compass.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

compass→**get_recordedData()****YCompass****compass**→**recordedData()****compass.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

YDataSet **get_recordedData**(long **startTime**, long **endTime**)

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

compass→**get_reportFrequency()****YCompass****compass**→**reportFrequency()****compass.get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String **get_reportFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

compass→**get_resolution()**

YCompass

compass→**resolution()**`compass.get_resolution()`

Retourne la résolution des valeurs mesurées.

double **get_resolution()** ()

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_RESOLUTION_INVALID.

compass→**get_unit()****YCompass****compass**→**unit()****compass.get_unit()**

Retourne l'unité dans laquelle le cap relatif est exprimée.

String **get_unit()**

Retourne :

une chaîne de caractères représentant l'unité dans laquelle le cap relatif est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

compass→**get_userData()**

YCompass

compass→**userData()**`compass.get_userData()`

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

Object `get_userData()`

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

compass→**isOnline()****compass.isOnline()****YCompass**

Vérifie si le module hébergeant le compas est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du compas sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

true si le compas est joignable, false sinon

compass→**load()****compass.load()****YCompass**

Met en cache les valeurs courantes du compas, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→loadCalibrationPoints()**YCompass****compass.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→**nextCompass()**`compass.nextCompass()`

YCompass

Continue l'énumération des compas commencée à l'aide de `yFirstCompass()`.

`YCompass` **nextCompass()**

Retourne :

un pointeur sur un objet `YCompass` accessible en ligne, ou `null` lorsque l'énumération est terminée.

compass→registerTimedReportCallback()**YCompass****compass.registerTimedReportCallback()**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

compass→registerValueCallback()**YCompass****compass.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

compass→**set_highestValue()****YCompass****compass**→**setHighestValue()****compass.set_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→**set_logFrequency()****YCompass****compass**→**setLogFrequency()****compass.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→**set_logicalName()**
compass→**setLogicalName()**
compass.set_logicalName()

YCompass

Modifie le nom logique du compas.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du compas.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→**set_lowestValue()**
compass→**setLowestValue()**
compass.set_lowestValue()

YCompass

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→**set_reportFrequency()**
compass→**setReportFrequency()**
compass.set_reportFrequency()

YCompass

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→**set_resolution()**

YCompass

compass→**setResolution()**

compass.set_resolution()

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

compass→**set_userdata()****YCompass****compass**→**setUserData()****compass.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

3.8. Interface de la fonction Current

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_current.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YCurrent = yoctolib.YCurrent;
php	require_once('yocto_current.php');
c++	#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)

3. Reference

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

current→**set_logicalName(newval)**

Modifie le nom logique du capteur de courant.

current→**set_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

current→**set_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

current→**set_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

current→**set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

current→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YCurrent.FindCurrent()**YCurrent****yFindCurrent()**`YCurrent.FindCurrent()`

Permet de retrouver un capteur de courant d'après un identifiant donné.

```
YCurrent FindCurrent( String func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de courant soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YCurrent.isOnline()` pour tester si le capteur de courant est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de courant sans ambiguïté

Retourne :

un objet de classe `YCurrent` qui permet ensuite de contrôler le capteur de courant.

YCurrent.FirstCurrent()

YCurrent

yFirstCurrent()`YCurrent.FirstCurrent()`

Commence l'énumération des capteurs de courant accessibles par la librairie.

`YCurrent` **FirstCurrent()**

Utiliser la fonction `YCurrent.nextCurrent()` pour itérer sur les autres capteurs de courant.

Retourne :

un pointeur sur un objet `YCurrent`, correspondant au premier capteur de courant accessible en ligne, ou `null` si il n'y a pas de capteurs de courant disponibles.

current→calibrateFromPoints()**YCurrent****current.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                        ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→describe()`current.describe()`**YCurrent**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de courant au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le capteur de courant (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

current→**get_advertisedValue()****YCurrent****current**→**advertisedValue()****current.get_advertisedValue()**

Retourne la valeur courante du capteur de courant (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de courant (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

current→**get_currentRawValue()**

YCurrent

current→**currentRawValue()**

current.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en mA, sous forme de nombre à virgule.

double **get_currentRawValue()**

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en mA, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

current→**get_currentValue()**
current→**currentValue()**
current.get_currentValue()

YCurrent

Retourne la valeur actuelle du courant, en mA, sous forme de nombre à virgule.

double **get_currentValue()**

Retourne :

une valeur numérique représentant la valeur actuelle du courant, en mA, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

current→**get_errorMessage()**

YCurrent

current→**errorMessage()**

current.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de courant.

current→**get_errorType()****YCurrent****current**→**errorType()****current.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

int **get_errorType()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de courant.

current→**get_friendlyName()**

YCurrent

current→**friendlyName()**

current.get_friendlyName()

Retourne un identifiant global du capteur de courant au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du capteur de courant si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de courant (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de courant en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

current→**get_functionDescriptor()****YCurrent****current**→**functionDescriptor()****current.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

current→**get_functionId()**

YCurrent

current→**functionId()****current.get_functionId()**

Retourne l'identifiant matériel du capteur de courant, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de courant (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

current→**get_hardwareId()****YCurrent****current**→**hardwareId()****current.get_hardwareId()**

Retourne l'identifiant matériel unique du capteur de courant au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de courant (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur de courant (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

current→**get_highestValue()**

YCurrent

current→**highestValue()**

current.get_highestValue()

Retourne la valeur maximale observée pour le courant depuis le démarrage du module.

double **get_highestValue()**

Retourne :

une valeur numérique représentant la valeur maximale observée pour le courant depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

current→**get_logFrequency()****YCurrent****current**→**logFrequency()****current.get_logFrequency()**

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.

String **get_logFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

current→**get_logicalName()**

YCurrent

current→**logicalName()**

current.get_logicalName()

Retourne le nom logique du capteur de courant.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du capteur de courant.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

current→**get_lowestValue()****YCurrent****current**→**lowestValue()****current.get_lowestValue()**

Retourne la valeur minimale observée pour le courant depuis le démarrage du module.

```
double get_lowestValue()
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour le courant depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

current→**get_module()**

YCurrent

current→**module()**`current.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

current→**get_recordedData()****YCurrent****current**→**recordedData()****current.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

YDataSet **get_recordedData**(long **startTime**, long **endTime**)

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

current→get_reportFrequency()

YCurrent

current→reportFrequency()

current.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String get_reportFrequency()

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

current→**get_resolution()****YCurrent****current**→**resolution()****current.get_resolution()**

Retourne la résolution des valeurs mesurées.

double **get_resolution()** ()

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_RESOLUTION_INVALID.

current→**get_unit()**

YCurrent

current→**unit()**`current.get_unit()`

Retourne l'unité dans laquelle le courant est exprimée.

String **get_unit()**

Retourne :

une chaîne de caractères représentant l'unité dans laquelle le courant est exprimée

En cas d'erreur, déclenche une exception ou retourne Y_UNIT_INVALID.

current→**get_userData()****YCurrent****current**→**userData()****current.get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

Object **get_userData()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

current→**isOnline()****current.isOnline()**

YCurrent

Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache du capteur de courant sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur de courant est joignable, `false` sinon

current→**load()****current.load()****YCurrent**

Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→loadCalibrationPoints()**YCurrent****current.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**nextCurrent()****current.nextCurrent ()****YCurrent**

Continue l'énumération des capteurs de courant commencée à l'aide de `yFirstCurrent ()`.

`YCurrent` **nextCurrent()**

Retourne :

un pointeur sur un objet `YCurrent` accessible en ligne, ou `null` lorsque l'énumération est terminée.

current→registerTimedReportCallback()**YCurrent****current.registerTimedReportCallback()**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

current→registerValueCallback()**YCurrent****current.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

current→**set_highestValue()**
current→**setHighestValue()**
current.set_highestValue()

YCurrent

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**set_logFrequency()**
current→**setLogFrequency()**
current.set_logFrequency()

YCurrent

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**set_logicalName()**

YCurrent

current→**setLogicalName()**

current.set_logicalName()

Modifie le nom logique du capteur de courant.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de courant.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**set_lowestValue()**
current→**setLowestValue()**
current.set_lowestValue()

YCurrent

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**set_reportFrequency()**

YCurrent

current→**setReportFrequency()**

current.set_reportFrequency()

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**set_resolution()**
current→**setResolution()**
current.set_resolution()

YCurrent

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→**set_userdata()**

YCurrent

current→**setUserData()****current.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

`void set_userdata(Object data)`

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

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');
c++	#include "yocto_datalogger.h"
m	#import "yocto_datalogger.h"
pas	uses yocto_datalogger;
vb	yocto_datalogger.vb
cs	yocto_datalogger.cs
java	import com.yoctopuce.YoctoAPI.YDataLogger;
py	from yocto_datalogger import *

Fonction globales

yFindDataLogger(func)

Permet de retrouver un enregistreur de données d'après un identifiant donné.

yFirstDataLogger()

Commence l'énumération des enregistreurs de données accessibles par la librairie.

Méthodes des objets YDataLogger

datalogger→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'enregistreur de données au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

datalogger→forgetAllDataStreams()

Efface tout l'historique des mesures de l'enregistreur de données.

datalogger→get_advertisedValue()

Retourne la valeur courante de l'enregistreur de données (pas plus de 6 caractères).

datalogger→get_autoStart()

Retourne le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

datalogger→get_beaconDriven()

Retourne vrai si l'enregistreur de données est synchronisé avec la balise de localisation.

datalogger→get_currentRunIndex()

Retourne le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active.

datalogger→get_dataSets()

Retourne une liste d'objets YDataSet permettant de récupérer toutes les mesures stockées par l'enregistreur de données.

datalogger→get_dataStreams(v)

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur (ancienne méthode).

datalogger→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

datalogger→get_errorType()

3. Reference

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

`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.

YDataLogger.FindDataLogger()**YDataLogger****yFindDataLogger()****YDataLogger.FindDataLogger()**

Permet de retrouver un enregistreur de données d'après un identifiant donné.

YDataLogger **FindDataLogger**(String **func**)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'enregistreur de données soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDataLogger.isOnline()` pour tester si l'enregistreur de données est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence l'enregistreur de données sans ambiguïté

Retourne :

un objet de classe `YDataLogger` qui permet ensuite de contrôler l'enregistreur de données.

YDataLogger.FirstDataLogger()**YDataLogger****yFirstDataLogger()****YDataLogger.FirstDataLogger()**

Commence l'énumération des enregistreurs de données accessibles par la librairie.

`YDataLogger` **FirstDataLogger()**

Utiliser la fonction `YDataLogger.nextDataLogger()` pour itérer sur les autres enregistreurs de données.

Retourne :

un pointeur sur un objet `YDataLogger`, correspondant au premier enregistreur de données accessible en ligne, ou `null` si il n'y a pas de enregistreurs de données disponibles.

datalogger→**describe()**`datalogger.describe()`**YDataLogger**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'enregistreur de données au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant l'enregistreur de données (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

datalogger→forgetAllDataStreams()**YDataLogger****datalogger.forgetAllDataStreams()**

Efface tout l'historique des mesures de l'enregistreur de données.

int forgetAllDataStreams()

Cette méthode remet aussi à zéro le compteur de Runs.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→**get_advertisedValue()**

YDataLogger

datalogger→**advertisedValue()**

datalogger.get_advertisedValue()

Retourne la valeur courante de l'enregistreur de données (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante de l'enregistreur de données (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

datalogger→get_autoStart()**YDataLogger****datalogger→autoStart()****datalogger.get_autoStart()**

Retourne le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

int **get_autoStart()**

Retourne :

soit Y_AUTOSTART_OFF, soit Y_AUTOSTART_ON, selon le mode d'activation automatique de l'enregistreur de données à la mise sous tension

En cas d'erreur, déclenche une exception ou retourne Y_AUTOSTART_INVALID.

datalogger→get_beaconDriven()

YDataLogger

datalogger→beaconDriven()

datalogger.get_beaconDriven()

Retourne vrai si l'enregistreur de données est synchronisé avec la balise de localisation.

int get_beaconDriven()

Retourne :

soit Y_BEACONDRIVEN_OFF, soit Y_BEACONDRIVEN_ON, selon vrai si l'enregistreur de données est synchronisé avec la balise de localisation

En cas d'erreur, déclenche une exception ou retourne Y_BEACONDRIVEN_INVALID.

datalogger→get_currentRunIndex()**YDataLogger****datalogger→currentRunIndex()****datalogger.get_currentRunIndex()**

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.

```
int get_currentRunIndex( )
```

Retourne :

un entier représentant le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRUNINDEX_INVALID.

datalogger→**get_dataSets()**

YDataLogger

datalogger→**dataSets()**

datalogger.get_dataSets()

Retourne une liste d'objets YDataSet permettant de récupérer toutes les mesures stockées par l'enregistreur de données.

`ArrayList<YDataSet> get_dataSets() ()`

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets YDataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Retourne :

une liste d'objets YDataSet

En cas d'erreur, déclenche une exception ou retourne une liste vide.

datalogger→get_dataStreams()**YDataLogger****datalogger→dataStreams()****datalogger.get_dataStreams()**

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur (ancienne méthode).

```
int get_dataStreams( ArrayList<YDataStream> v)
```

L'appelant doit passer par référence un tableau vide pour stocker les objets YDataStream, et la méthode va les remplir avec des objets décrivant les séquences de données disponibles.

Cette méthode est préservée pour maintenir la compatibilité avec les applications existantes. Pour les nouvelles applications, il est préférable d'utiliser la méthode `get_dataSets()` ou d'appeler directement la méthode `get_recordedData()` sur l'objet représentant le capteur désiré.

Paramètres :

v un tableau de YDataStreams qui sera rempli avec les séquences trouvées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→get_errorMessage()

YDataLogger

datalogger→errorMessage()

datalogger.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

String get_errorMessage()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'enregistreur de données.

datalogger→get_errorType()**YDataLogger****datalogger→errorType()****datalogger.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'enregistreur de données.

int get_errorType()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'enregistreur de données.

datalogger→get_friendlyName()

YDataLogger

datalogger→friendlyName()

datalogger.get_friendlyName()

Retourne un identifiant global de l'enregistreur de données au format `NOM_MODULE.NOM_FONCTION`.

String get_friendlyName()

Le chaîne retournée utilise soit les noms logiques du module et de l'enregistreur de données si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'enregistreur de données (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'enregistreur de données en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

datalogger→get_functionDescriptor()**YDataLogger****datalogger→functionDescriptor()****datalogger.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

datalogger→**get_functionId()**

YDataLogger

datalogger→**functionId()**

datalogger.get_functionId()

Retourne l'identifiant matériel de l'enregistreur de données, sans référence au module.

String **get_functionId()** ()

Par exemple relay1.

Retourne :

une chaîne de caractères identifiant l'enregistreur de données (ex: relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FUNCTIONID_INVALID.

datalogger→get_hardwareId()**YDataLogger****datalogger→hardwareId()****datalogger.get_hardwareId()**

Retourne l'identifiant matériel unique de l'enregistreur de données au format `SERIAL.FUNCTIONID`.

String get_hardwareId()

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'enregistreur de données (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant l'enregistreur de données (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

datalogger→**get_logicalName()**

YDataLogger

datalogger→**logicalName()**

datalogger.get_logicalName()

Retourne le nom logique de l'enregistreur de données.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique de l'enregistreur de données.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

datalogger→get_module()**YDataLogger****datalogger→module()**`datalogger.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

datalogger→get_recording()

YDataLogger

datalogger→recording()

datalogger.get_recording()

Retourne l'état d'activation de l'enregistreur de données.

int get_recording()

Retourne :

soit Y_RECORDING_OFF, soit Y_RECORDING_ON, selon l'état d'activation de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_RECORDING_INVALID.

datalogger→**get_timeUTC()****YDataLogger****datalogger**→**timeUTC()****datalogger.get_timeUTC()**

Retourne le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue.

long **get_timeUTC()**

Retourne :

un entier représentant le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue

En cas d'erreur, déclenche une exception ou retourne Y_TIMEUTC_INVALID.

datalogger→get_userdata()

YDataLogger

datalogger→userdata()

datalogger.get_userdata()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

datalogger→**isOnline()****datalogger.isOnline()****YDataLogger**

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache de l'enregistreur de données sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'enregistreur de données est joignable, `false` sinon

datalogger→**load()**`datalogger.load()`**YDataLogger**

Met en cache les valeurs courantes de l'enregistreur de données, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→**nextDataLogger()****YDataLogger****datalogger.nextDataLogger()**

Continue l'énumération des enregistreurs de données commencée à l'aide de `yFirstDataLogger()`.

`YDataLogger` **nextDataLogger()**

Retourne :

un pointeur sur un objet `YDataLogger` accessible en ligne, ou `null` lorsque l'énumération est terminée.

datalogger→registerValueCallback()**YDataLogger****datalogger.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

datalogger→**set_autoStart()****YDataLogger****datalogger**→**setAutoStart()****datalogger.set_autoStart()**

Modifie le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

```
int set_autoStart( int newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon le mode d'activation automatique de l'enregistreur de données à la mise sous tension

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→**set_beaconDriven()****YDataLogger****datalogger**→**setBeaconDriven()****datalogger.set_beaconDriven()**

Modifie le mode de synchronisation de l'enregistreur de données .

```
int set_beaconDriven( int newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval soit `Y_BEACONDRIVEN_OFF`, soit `Y_BEACONDRIVEN_ON`, selon le mode de synchronisation de l'enregistreur de données

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→set_logicalName()**YDataLogger****datalogger→setLogicalName()****datalogger.set_logicalName()**

Modifie le nom logique de l'enregistreur de données.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'enregistreur de données.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→**set_recording()**

YDataLogger

datalogger→**setRecording()**

datalogger.set_recording()

Modifie l'état d'activation de l'enregistreur de données.

```
int set_recording( int newval)
```

Paramètres :

newval soit Y_RECORDING_OFF, soit Y_RECORDING_ON, selon l'état d'activation de l'enregistreur de données

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→set_timeUTC()**YDataLogger****datalogger→setTimeUTC()****datalogger.set_timeUTC()**

Modifie la référence de temps UTC, afin de l'attacher aux données enregistrées.

```
int set_timeUTC( long newval)
```

Paramètres :

newval un entier représentant la référence de temps UTC, afin de l'attacher aux données enregistrées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

datalogger→**set_userdata()**

YDataLogger

datalogger→**setUserData()**

datalogger.set_userdata()

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

void **set_userdata**(Object **data**)

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

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');
c++	#include "yocto_datalogger.h"
m	#import "yocto_datalogger.h"
pas	uses yocto_datalogger;
vb	yocto_datalogger.vb
cs	yocto_datalogger.cs
java	import com.yoctopuce.YoctoAPI.YDataLogger;
py	from yocto_datalogger import *

Méthodes des objets YDataRun
datarun→get_averageValue(measureName, pos) Retourne la valeur moyenne des mesures observées au moment choisi.
datarun→get_duration() Retourne la durée (en secondes) du Run.
datarun→get_maxValue(measureName, pos) Retourne la valeur maximale des mesures observées au moment choisi.
datarun→get_measureNames() Retourne les noms des valeurs mesurées par l'enregistreur de données.
datarun→get_minValue(measureName, pos) Retourne la valeur minimale des mesures observées au moment choisi.
datarun→get_startTimeUTC() Retourne l'heure absolue du début du Run, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).
datarun→get_valueCount() Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.
datarun→get_valueInterval() Retourne l'intervalle de temps représenté par chaque valeur de ce run.
datarun→set_valueInterval(valueInterval) Change l'intervalle de temps représenté par chaque valeur de ce run.

datarun→**get_averageValue()****datarun**→**averageValue()****datarun.get_averageValue()**

Retourne la valeur moyenne des mesures observées au moment choisi.

```
double get_averageValue( String measureName, int pos)
```

datarun→**get_averageValue()****datarun**→**averageValue()****datarun.get_averageValue()**

Retourne la valeur moyenne des mesures observées au moment choisi.

js	function get_averageValue (measureName , pos)
nodejs	function get_averageValue (measureName , pos)
php	function get_averageValue (\$measureName , \$pos)
java	double get_averageValue (String measureName , int pos)
py	def get_averageValue (measureName , pos)

Paramètres :

measureName le nom de la mesure désirée (un des noms retournés par `get_measureNames`)
pos l'index de la position désirée, entre 0 et la valeur de `get_valueCount`

Retourne :

une nombre flottant (la valeur moyenne).

En cas d'erreur, déclenche une exception ou retourne `Y_AVERAGEVALUE_INVALID`.

datarun→**get_duration()****YDataRun****datarun**→**duration()****datarun.get_duration()**

Retourne la durée (en secondes) du Run.

```
long get_duration( )
```

datarun→**get_duration()****datarun**→**duration()****datarun.get_duration()**

Retourne la durée (en secondes) du Run.

```
js function get_duration( )
```

```
nodejs function get_duration( )
```

```
php function get_duration( )
```

```
java long get_duration( )
```

```
py def get_duration( )
```

Lorsque cette méthode est appelée dur le Run courant et que l'enregistreur de données est actif, l'appel à cette méthode force un rechargement de la dernière séquence du module pour s'assurer que la réponse prend en compte les dernières données enregistrées.

Retourne :

un entier positif correspondant au nombre de secondes écoulées entre le début du Run (quand le module a été mis sous tension) et la dernière mesure enregistrée.

datarun→get_maxValue()**YDataRun****datarun→maxValue()****datarun.get_maxValue()**

Retourne la valeur maximale des mesures observées au moment choisi.

```
double get_maxValue( String measureName, int pos)
```

datarun→get_maxValue()**datarun→maxValue()****datarun.get_maxValue()**

Retourne la valeur maximale des mesures observées au moment choisi.

```
js function get_maxValue( measureName, pos)
```

```
nodejs function get_maxValue( measureName, pos)
```

```
php function get_maxValue( $measureName, $pos)
```

```
java double get_maxValue( String measureName, int pos)
```

```
py def get_maxValue( measureName, pos)
```

Paramètres :

measureName le nom de la mesure désirée (un des noms retournés par get_measureNames)

pos l'index de la position désirée, entre 0 et la valeur de get_valueCount

Retourne :

une nombre flottant (la valeur maximale).

En cas d'erreur, déclenche une exception ou retourne Y_MAXVALUE_INVALID.

datarun→get_measureNames()**datarun→measureNames()****datarun.get_measureNames()**

Retourne les noms des valeurs mesurées par l'enregistreur de données.

`ArrayList<String> get_measureNames()`

datarun→get_measureNames()**datarun→measureNames()****datarun.get_measureNames()**

Retourne les noms des valeurs mesurées par l'enregistreur de données.

<code>js</code>	<code>function get_measureNames()</code>
<code>nodejs</code>	<code>function get_measureNames()</code>
<code>php</code>	<code>function get_measureNames()</code>
<code>java</code>	<code>ArrayList<String> get_measureNames()</code>
<code>py</code>	<code>def get_measureNames()</code>

Dans la plupart des cas, le nom des colonnes correspond à l'identifiant matériel du capteur qui a produit la mesure.

Retourne :

une liste de chaîne de caractères (les noms des mesures)

En cas d'erreur, déclenche une exception ou retourne une liste vide.

datarun→**get_minValue()****YDataRun****datarun**→**minValue()****datarun.get_minValue()**

Retourne la valeur minimale des mesures observées au moment choisi.

```
double get_minValue( String measureName, int pos)
```

datarun→**get_minValue()****datarun**→**minValue()****datarun.get_minValue()**

Retourne la valeur minimale des mesures observées au moment choisi.

js	<pre>function get_minValue(measureName, pos)</pre>
nodejs	<pre>function get_minValue(measureName, pos)</pre>
php	<pre>function get_minValue(\$measureName, \$pos)</pre>
java	<pre>double get_minValue(String measureName, int pos)</pre>
py	<pre>def get_minValue(measureName, pos)</pre>

Paramètres :

measureName le nom de la mesure désirée (un des noms retournés par `get_measureNames`)

pos l'index de la position désirée, entre 0 et la valeur de `get_valueCount`

Retourne :

une nombre flottant (la valeur minimale).

En cas d'erreur, déclenche une exception ou retourne `Y_MINVALUE_INVALID`.

datarun→get_startTimeUTC()**YDataRun****datarun→startTimeUTC()**

Retourne l'heure absolue du début du Run, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

Si l'heure UTC n'a jamais été configurée dans l'enregistreur de données durant le run, et si il ne s'agit pas du run courant, cette méthode retourne 0.

Retourne :

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et le début du Run.

datarun→**get_valueCount()****YDataRun****datarun**→**valueCount()****datarun.get_valueCount()**

Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.

```
int get_valueCount( )
```

datarun→**get_valueCount()****datarun**→**valueCount()****datarun.get_valueCount()**

Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.

```
js function get_valueCount( )
```

```
nodejs function get_valueCount( )
```

```
php function get_valueCount( )
```

```
java int get_valueCount( )
```

```
py def get_valueCount( )
```

Lorsque cette méthode est appelée sur le Run courant et que l'enregistreur de données est actif, l'appel à cette méthode force un rechargement de la dernière séquence du module pour s'assurer que la réponse prend en compte les dernières données enregistrées.

Retourne :

un entier positif correspondant à la durée du Run divisée par l'intervalle entre les valeurs.

datarun→get_valueInterval()
datarun→valueInterval()
datarun.get_valueInterval()

YDataRun

Retourne l'intervalle de temps représenté par chaque valeur de ce run.

int **get_valueInterval()**

datarun→get_valueInterval()
datarun→valueInterval()**datarun.get_valueInterval()**

Retourne l'intervalle de temps représenté par chaque valeur de ce run.

js	function get_valueInterval()
nodejs	function get_valueInterval()
php	function get_valueInterval()
java	int get_valueInterval()
py	def get_valueInterval()

La valeur par défaut correspond à la plus grande granularité des mesures archivées dans la flash de l'enregistreur de données pour ce Run, mais l'intervalle à utiliser peut être configuré librement si désiré.

Retourne :

un entier positif correspondant au nombre de secondes couvertes par chaque valeur représentée dans le Run.

datarun→**set_valueInterval()****datarun**→**setValueInterval()****datarun.set_valueInterval()**

Change l'intervalle de temps représenté par chaque valeur de ce run.

```
void set_valueInterval( int valueInterval)
```

datarun→**set_valueInterval()****datarun**→**setValueInterval()****datarun.set_valueInterval()**

Change l'intervalle de temps représenté par chaque valeur de ce run.

```
js function set_valueInterval( valueInterval)
nodejs function set_valueInterval( valueInterval)
php function set_valueInterval( $valueInterval)
java void set_valueInterval( int valueInterval)
py def set_valueInterval( valueInterval)
```

La valeur par défaut correspond à la plus grande granularité des mesures archivées dans la flash de l'enregistreur de données pour ce Run, mais l'intervalle à utiliser peut être configuré librement si désiré.

Paramètres :

valueInterval un nombre entier de secondes.

Retourne :

nothing

3.11. 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	<code><script type='text/javascript' src='yocto_api.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib'); var YAPI = yoctolib.YAPI; var YModule = yoctolib.YModule;</code>
php	<code>require_once('yocto_api.php');</code>
c++	<code>#include "yocto_api.h"</code>
m	<code>#import "yocto_api.h"</code>
pas	<code>uses yocto_api;</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>

Méthodes des objets YDataSet

dataset→get_endTimeUTC()

Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

dataset→get_functionId()

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

dataset→get_hardwareId()

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format `SERIAL.FUNCTIONID`.

dataset→get_measures()

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

dataset→get_preview()

Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.

dataset→get_progress()

Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.

dataset→get_startTimeUTC()

Retourne l'heure absolue du début des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

dataset→get_summary()

Retourne un objet YMeasure résumant tout le YDataSet.

dataset→get_unit()

3. Reference

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

dataset→loadMore()

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, et met à jour l'indicateur d'avancement.

dataset→loadMore_async(callback, context)

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

dataset→**get_endTimeUTC()****YDataSet****dataset**→**endTimeUTC()****dataset.get_endTimeUTC()**

Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

```
long get_endTimeUTC( )
```

Lorsque l'objet YDataSet est créé, l'heure de fin est celle qui a été passée en paramètre à la fonction `get_dataSet`. Dès le premier appel à la méthode `loadMore()`, l'heure de fin est mise à jour à la dernière mesure effectivement disponible dans l'enregistreur de données pour la plage spécifiée.

Retourne :

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et la dernière mesure.

dataset→**get_functionId()**

YDataSet

dataset→**functionId()**`dataset.get_functionId()`

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

String **get_functionId()**

Par exemple `temperature1`.

Retourne :

une chaîne de caractères identifiant la fonction (ex: `temperature1`)

dataset→**get_hardwareId()****YDataSet****dataset**→**hardwareId()****dataset.get_hardwareId()**

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par exemple `THRMCPL1-123456.temperature1`).

Retourne :

une chaîne de caractères identifiant la fonction (ex: `THRMCPL1-123456.temperature1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

dataset→**get_measures()****YDataSet****dataset**→**measures()****dataset.get_measures()**

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

```
ArrayList<YMeasure> get_measures( )
```

Chaque élément contient: - le moment où la mesure a débuté - le moment où la mesure s'est terminée - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Avant d'appeler cette méthode, vous devez appeler `loadMore()` pour charger des données depuis l'enregistreur sur le module. L'appel doit être répété plusieurs fois pour charger toutes les données, mais vous pouvez commencer à utiliser les données disponibles avant qu'elles n'aient été toutes chargées

Les mesures les plus anciennes sont toujours chargées les premières, et les plus récentes en dernier. De ce fait, les timestamps dans la table des mesures sont normalement par ordre chronologique. La seule exception est dans le cas où il y a eu un ajustement de l'horloge UTC de l'enregistreur de données pendant l'enregistrement.

Retourne :

un tableau d'enregistrements, chaque enregistrement représentant une mesure effectuée à un moment précis.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

dataset→**get_preview()****YDataSet****dataset**→**preview()****dataset.get_preview()**

Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.

```
ArrayList<YMeasure> get_preview()
```

Chaque élément contient: - le début d'un intervalle de temps - la fin d'un intervalle de temps - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Le résumé des mesures est disponible dès que `loadMore()` a été appelé pour la première fois.

Retourne :

un tableau d'enregistrements, chaque enregistrement représentant les mesures observée durant un certain intervalle de temps.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

dataset→**get_progress()**

YDataSet

dataset→**progress()**`dataset.get_progress()`

Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.

`int get_progress()`

A l'instanciation de l'objet par la fonction `get_dataSet()`, l'avancement est nul. Au fur et à mesure des appels à `loadMore()`, l'avancement progresse pour atteindre la valeur 100 lorsque toutes les mesures ont été chargées.

Retourne :

un nombre entier entre 0 et 100 représentant l'avancement du chargement des données demandées.

dataset→**get_startTimeUTC()****YDataSet****dataset**→**startTimeUTC()****dataset.startTimeUTC()**

Retourne l'heure absolue du début des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

```
long get_startTimeUTC( )
```

Lorsque l'objet YDataSet est créé, l'heure de départ est celle qui a été passée en paramètre à la fonction `get_dataSet`. Dès le premier appel à la méthode `loadMore()`, l'heure de départ est mise à jour à la première mesure effectivement disponible dans l'enregistreur de données pour la plage spécifiée.

Retourne :

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et la première mesure enregistrée.

dataset→**get_summary()**

YDataSet

dataset→**summary()**`dataset.get_summary()`

Retourne un objet YMeasure résumant tout le YDataSet.

YMeasure **get_summary()**

Il inclut les information suivantes: - le moment de la première mesure - le moment de la dernière mesure - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Ce résumé des mesures est disponible dès que `loadMore()` a été appelé pour la première fois.

Retourne :

un objet YMeasure

dataset→**get_unit()****YDataSet****dataset**→**unit()**`dataset.get_unit()`

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

String **get_unit()**

Retourne :

une chaîne de caractères représentant une unité physique.

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

dataset→**loadMore()****dataset.loadMore()****YDataSet**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, et met à jour l'indicateur d'avancement.

int loadMore()**Retourne :**

un nombre entier entre 0 et 100 représentant l'avancement du chargement des données demandées, ou un code d'erreur négatif en cas de problème.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.12. 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	<code><script type='text/javascript' src='yocto_api.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YAPI = yoctolib.YAPI;</code> <code>var YModule = yoctolib.YModule;</code>
php	<code>require_once('yocto_api.php');</code>
c++	<code>#include "yocto_api.h"</code>
m	<code>#import "yocto_api.h"</code>
pas	<code>uses yocto_api;</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>

Méthodes des objets `YDataStream`

`datastream→get_averageValue()`

Retourne la moyenne des valeurs observées durant cette séquence.

`datastream→get_columnCount()`

Retourne le nombre de colonnes de données contenus dans la séquence.

`datastream→get_columnNames()`

Retourne le nom (la sémantique) des colonnes de données contenus dans la séquence.

`datastream→get_data(row, col)`

Retourne une mesure unique de la séquence, spécifiée par l'index de l'enregistrement (ligne) et de la mesure (colonne).

`datastream→get_dataRows()`

Retourne toutes les données mesurées 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()`

3. Reference

Retourne le temps de départ relatif de la séquence (en secondes).

datastream→**get_startTimeUTC()**

Retourne l'heure absolue du début de la séquence de données, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

datastream→get_averageValue()**YDataStream****datastream→averageValue()****datastream.get_averageValue()**

Retourne la moyenne des valeurs observées durant cette séquence.

```
double get_averageValue()
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y_DATA_INVALID.

Retourne :

un nombre décimal correspondant à la moyenne des valeurs, ou Y_DATA_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y_DATA_INVALID.

datastream→**get_columnCount()**

YDataStream

datastream→**columnCount()**

datastream.get_columnCount()

Retourne le nombre de colonnes de données contenus dans la séquence.

```
int get_columnCount()
```

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclenche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

Retourne :

un entier positif correspondant au nombre de colonnes.

En cas d'erreur, déclenche une exception ou retourne zéro.

datastream→get_columnNames()**YDataStream****datastream→columnNames()****datastream.get_columnNames()**

Retourne le nom (la sémantique) des colonnes de données contenus dans la séquence.

`ArrayList<String> get_columnNames()`

Dans la plupart des cas, le nom des colonnes correspond à l'identifiant matériel du capteur qui a produit la mesure. Pour les séquences enregistrées à faible fréquence, l'enregistreur de donnée stocke la valeur min, moyenne et max observée durant chaque intervalle de temps dans des colonnes avec les suffixes `_min`, `_avg` et `_max` respectivement.

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclenche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

Retourne :

une liste de chaîne de caractères.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

datastream→**get_data()**

YDataStream

datastream→**data()**`datastream.get_data()`

Retourne une mesure unique de la séquence, spécifiée par l'index de l'enregistrement (ligne) et de la mesure (colonne).

```
double get_data( int row, int col)
```

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Cette méthode déclenche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

Paramètres :

row index de l'enregistrement (ligne)

col index de la mesure (colonne)

Retourne :

un nombre décimal

En cas d'erreur, déclenche une exception ou retourne `Y_DATA_INVALID`.

datastream→get_dataRows()**YDataStream****datastream→dataRows()****datastream.get_dataRows()**

Retourne toutes les données mesurées contenues dans la séquence, sous forme d'une liste de vecteurs (table bidimensionnelle).

```
ArrayList<ArrayList<Double>> get_dataRows( )
```

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Cette méthode déclenche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

Retourne :

une liste d'enregistrements, chaque enregistrement étant lui-même une liste de nombres décimaux.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

datastream→**get_dataSamplesIntervalMs()**

YDataStream

datastream→**dataSamplesIntervalMs()**

datastream.get_dataSamplesIntervalMs()

Retourne le nombre de millisecondes entre chaque mesure de la séquence.

```
int get_dataSamplesIntervalMs()
```

Par défaut, l'enregistreur mémorise une mesure par seconde, mais la fréquence d'enregistrement peut être changée pour chaque fonction.

Retourne :

un entier positif correspondant au nombre de millisecondes entre deux mesures consécutives.

datastream→**get_duration()****YDataStream****datastream**→**duration()****datastream.get_duration()**

Retourne la durée approximative de cette séquence, en secondes.

int **get_duration()**

Retourne :

le nombre de secondes couvertes par cette séquence.

En cas d'erreur, déclenche une exception ou retourne Y_DURATION_INVALID.

datastream→get_maxValue()

YDataStream

datastream→maxValue()

datastream.get_maxValue()

Retourne la plus grande valeur observée durant cette séquence.

```
double get_maxValue( )
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y_DATA_INVALID.

Retourne :

un nombre décimal correspondant à la plus grande valeur, ou Y_DATA_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y_DATA_INVALID.

datastream→get_minValue()
datastream→minValue()
datastream.get_minValue()

YDataStream

Retourne la plus petite valeur observée durant cette séquence.

```
double get_minValue()
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode retournera toujours Y_DATA_INVALID.

Retourne :

un nombre décimal correspondant à la plus petite valeur, ou Y_DATA_INVALID si la séquence n'est pas encore terminée.

En cas d'erreur, déclenche une exception ou retourne Y_DATA_INVALID.

datastream→get_rowCount()

YDataStream

datastream→rowCount()

datastream.getRowCount()

Retourne le nombre d'enregistrement contenus dans la séquence.

```
int get_rowCount( )
```

Si le module utilise un firmware antérieur à la version 13000, cette méthode déclenche le chargement de toutes les données de la séquence si nécessaire, ce qui peut prendre un petit instant.

Retourne :

un entier positif correspondant au nombre d'enregistrements.

En cas d'erreur, déclenche une exception ou retourne zéro.

datastream→get_runIndex()**YDataStream****datastream→runIndex()****datastream.get_runIndex()**

Retourne le numéro de Run de la séquence de données.

int **get_runIndex()**

Un Run peut être composé de plusieurs séquences, couvrant différents intervalles de temps.

Retourne :

un entier positif correspondant au numéro du Run

datastream→get_startTime()

YDataStream

datastream→startTime()

datastream.get_startTime()

Retourne le temps de départ relatif de la séquence (en secondes).

```
int get_startTime( )
```

Pour les firmwares récents, la valeur est relative à l'heure courante (valeur négative). Pour les modules utilisant un firmware plus ancien que la version 13000, la valeur est le nombre de secondes depuis la mise sous tension du module (valeur positive). Si vous désirez obtenir l'heure absolue du début de la séquence, utilisez `get_startTimeUTC()`.

Retourne :

un entier positif correspondant au nombre de secondes écoulées entre le début du Run et le début de la séquence enregistrée.

datastream→get_startTimeUTC()**YDataStream****datastream→startTimeUTC()****datastream.get_startTimeUTC()**

Retourne l'heure absolue du début de la séquence de données, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

long get_startTimeUTC()

Si l'heure UTC n'était pas configurée dans l'enregistreur de données au début de la séquence, cette méthode retourne 0.

Retourne :

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et le début de la séquence enregistrée.

3.13. Interface de la fonction DigitalIO

La librairie de programmation Yoctopuce permet simplement de changer l'état de chaque bit du port d'entrée sortie. Il est possible de changer tous les bits du port à la fois, ou de les changer indépendamment. La librairie permet aussi de créer des courtes impulsions de durée déterminée. Le comportement électrique de chaque entrée/sortie peut être modifié (open drain et polarité inverse).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_digitalio.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YDigitalIO = yoctolib.YDigitalIO;
php	require_once('yocto_digitalio.php');
c++	#include "yocto_digitalio.h"
m	#import "yocto_digitalio.h"
pas	uses yocto_digitalio;
vb	yocto_digitalio.vb
cs	yocto_digitalio.cs
java	import com.yoctopuce.YoctoAPI.YDigitalIO;
py	from yocto_digitalio import *

Fonction globales

yFindDigitalIO(func)

Permet de retrouver un port d'E/S digital d'après un identifiant donné.

yFirstDigitalIO()

Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.

Méthodes des objets YDigitalIO

digitalio→delayedPulse(bitno, ms_delay, ms_duration)

Préprogramme une impulsion de durée spécifiée sur un bit choisi.

digitalio→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du port d'E/S digital au format
TYPE (NAME) = SERIAL . FUNCTIONID.

digitalio→get_advertisedValue()

Retourne la valeur courante du port d'E/S digital (pas plus de 6 caractères).

digitalio→get_bitDirection(bitno)

Retourne la direction d'un seul bit du port d'E/S.

digitalio→get_bitOpenDrain(bitno)

Retourne la direction d'un seul bit du port d'E/S.

digitalio→get_bitPolarity(bitno)

Retourne la polarité d'un seul bit du port d'E/S.

digitalio→get_bitState(bitno)

Retourne l'état d'un seul bit du port d'E/S.

digitalio→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

digitalio→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

digitalio→get_friendlyName()

Retourne un identifiant global du port d'E/S digital au format NOM_MODULE . NOM_FONCTION.

digitalio→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

digitalio→get_functionId()

Retourne l'identifiant matériel du port d'E/S digital, sans référence au module.

digitalio→get_hardwareId()

Retourne l'identifiant matériel unique du port d'E/S digital au format `SERIAL.FUNCTIONID`.

digitalio→get_logicalName()

Retourne le nom logique du port d'E/S digital.

digitalio→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

digitalio→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

digitalio→get_outputVoltage()

Retourne la source de tension utilisée pour piloter les bits en sortie.

digitalio→get_portDirection()

Retourne la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

digitalio→get_portOpenDrain()

Retourne le type d'interface électrique de chaque bit du port (bitmap).

digitalio→get_portPolarity()

Retourne la polarité des bits du port (bitmap).

digitalio→get_portSize()

Retourne le nombre de bits implémentés dans le port d'E/S.

digitalio→get_portState()

Retourne l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

digitalio→get_userData()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

digitalio→isOnline()

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

digitalio→isOnline_async(callback, context)

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

digitalio→load(msValidity)

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

digitalio→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

digitalio→nextDigitalIO()

Continue l'énumération des ports d'E/S digitaux commencée à l'aide de `yFirstDigitalIO()`.

digitalio→pulse(bitno, ms_duration)

Déclenche une impulsion de durée spécifiée sur un bit choisi.

digitalio→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

digitalio→set_bitDirection(bitno, bitdirection)

Change la direction d'un seul bit du port d'E/S.

digitalio→set_bitOpenDrain(bitno, opendrain)

Change le type d'interface électrique d'un seul bit du port d'E/S.

digitalio→set_bitPolarity(bitno, bitpolarity)

Change la polarité d'un seul bit du port d'E/S.

3. Reference

digitalio→**set_bitState**(**bitno**, **bitstate**)

Change l'état d'un seul bit du port d'E/S.

digitalio→**set_logicalName**(**newval**)

Modifie le nom logique du port d'E/S digital.

digitalio→**set_outputVoltage**(**newval**)

Modifie la source de tension utilisée pour piloter les bits en sortie.

digitalio→**set_portDirection**(**newval**)

Modifie la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

digitalio→**set_portOpenDrain**(**newval**)

Modifie le type d'interface électrique de chaque bit du port (bitmap).

digitalio→**set_portPolarity**(**newval**)

Modifie la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée.

digitalio→**set_portState**(**newval**)

Modifie l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

digitalio→**set_userData**(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

digitalio→**toggle_bitState**(**bitno**)

Inverse l'état d'un seul bit du port d'E/S.

digitalio→**wait_async**(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YDigitalIO.FindDigitalIO()**YDigitalIO****yFindDigitalIO()****YDigitalIO.FindDigitalIO()**

Permet de retrouver un port d'E/S digital d'après un identifiant donné.

YDigitalIO FindDigitalIO(String func)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le port d'E/S digital soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDigitalIO.isOnline()` pour tester si le port d'E/S digital est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le port d'E/S digital sans ambiguïté

Retourne :

un objet de classe `YDigitalIO` qui permet ensuite de contrôler le port d'E/S digital.

YDigitalIO.FirstDigitalIO()

YDigitalIO

yFirstDigitalIO()**YDigitalIO.FirstDigitalIO()**

Commence l'énumération des ports d'E/S digitaux accessibles par la librairie.

YDigitalIO FirstDigitalIO()

Utiliser la fonction `YDigitalIO.nextDigitalIO()` pour itérer sur les autres ports d'E/S digitaux.

Retourne :

un pointeur sur un objet `YDigitalIO`, correspondant au premier port d'E/S digital accessible en ligne, ou `null` si il n'y a pas de ports d'E/S digitaux disponibles.

digitalio→delayedPulse()**YDigitalIO****digitalio.delayedPulse()**

Préprogramme une impulsion de durée spécifiée sur un bit choisi.

```
int delayedPulse( int bitno, int ms_delay, int ms_duration)
```

Le bit va passer à 1 puis automatiquement revenir à 0 après le temps donné.

Paramètres :

bitno	index du bit dans le port; le bit de poids faible est à l'index 0
ms_delay	délai d'attente avant l'impulsion, en millisecondes
ms_duration	durée de l'impulsion désirée, en millisecondes. Notez que la résolution temporelle du module n'est pas garantie à la milliseconde.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→describe()**digitalio.describe()****YDigitalIO**

Retourne un court texte décrivant de manière non-ambigüe l'instance du port d'E/S digital au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le port d'E/S digital (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

digitalio→**get_advertisedValue()****YDigitalIO****digitalio**→**advertisedValue()****digitalio.get_advertisedValue()**

Retourne la valeur courante du port d'E/S digital (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du port d'E/S digital (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

digitalio→**get_bitDirection()**

YDigitalIO

digitalio→**bitDirection()**

digitalio.get_bitDirection()

Retourne la direction d'un seul bit du port d'E/S.

```
int get_bitDirection( int bitno)
```

(0 signifie que le bit est une entrée, 1 une sortie)

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**get_bitOpenDrain()****YDigitalIO****digitalio**→**bitOpenDrain()****digitalio.get_bitOpenDrain()**

Retourne la direction d'un seul bit du port d'E/S.

```
int get_bitOpenDrain( int bitno)
```

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

Retourne :

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert)..

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**get_bitPolarity()**

YDigitalIO

digitalio→**bitPolarity()**

digitalio.get_bitPolarity()

Retourne la polarité d'un seul bit du port d'E/S.

```
int get_bitPolarity( int bitno)
```

0 signifie que l'entrée sortie est en mode normal, 1 qu'elle est en mode inverse

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**get_bitState()****YDigitalIO****digitalio**→**bitState()****digitalio.get_bitState()**

Retourne l'état d'un seul bit du port d'E/S.

```
int get_bitState( int bitno )
```

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

Retourne :

l'état du bit (0 ou 1).

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→get_errorMessage()

YDigitalIO

digitalio→errorMessage()

digitalio.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

String get_errorMessage()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du port d'E/S digital.

digitalio→**get_errorType()****YDigitalIO****digitalio**→**errorType()****digitalio.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port d'E/S digital.

int **get_errorType()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du port d'E/S digital.

digitalio→**get_friendlyName()****YDigitalIO****digitalio**→**friendlyName()****digitalio.get_friendlyName()**

Retourne un identifiant global du port d'E/S digital au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du port d'E/S digital si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du port d'E/S digital (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le port d'E/S digital en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

digitalio→**get_functionDescriptor()****YDigitalIO****digitalio**→**functionDescriptor()****digitalio.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

digitalio→**get_functionId()**

YDigitalIO

digitalio→**functionId()**

digitalio.get_functionId()

Retourne l'identifiant matériel du port d'E/S digital, sans référence au module.

String **get_functionId()** ()

Par exemple relay1.

Retourne :

une chaîne de caractères identifiant le port d'E/S digital (ex: relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FUNCTIONID_INVALID.

digitalio→**get_hardwareId()****YDigitalIO****digitalio**→**hardwareId()****digitalio.get_hardwareId()**

Retourne l'identifiant matériel unique du port d'E/S digital au format `SERIAL.FUNCTIONID`.

`String get_hardwareId()`

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du port d'E/S digital (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le port d'E/S digital (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

digitalio→**get_logicalName()**

YDigitalIO

digitalio→**logicalName()**

digitalio.get_logicalName()

Retourne le nom logique du port d'E/S digital.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du port d'E/S digital.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

digitalio→**get_module()****YDigitalIO****digitalio**→**module()****digitalio.get_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

YModule **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :

une instance de YModule

digitalio→**get_outputVoltage()****YDigitalIO****digitalio**→**outputVoltage()****digitalio.get_outputVoltage()**

Retourne la source de tension utilisée pour piloter les bits en sortie.

```
int get_outputVoltage( )
```

Retourne :

une valeur parmi Y_OUTPUTVOLTAGE_USB_5V, Y_OUTPUTVOLTAGE_USB_3V et Y_OUTPUTVOLTAGE_EXT_V représentant la source de tension utilisée pour piloter les bits en sortie

En cas d'erreur, déclenche une exception ou retourne Y_OUTPUTVOLTAGE_INVALID.

digitalio→**get_portDirection()****YDigitalIO****digitalio**→**portDirection()****digitalio.get_portDirection()**

Retourne la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

int **get_portDirection()****Retourne :**

un entier représentant la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie

En cas d'erreur, déclenche une exception ou retourne Y_PORTDIRECTION_INVALID.

digitalio→**get_portOpenDrain()**

YDigitalIO

digitalio→**portOpenDrain()**

digitalio.get_portOpenDrain()

Retourne le type d'interface électrique de chaque bit du port (bitmap).

int **get_portOpenDrain()**

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert).

Retourne :

un entier représentant le type d'interface électrique de chaque bit du port (bitmap)

En cas d'erreur, déclenche une exception ou retourne Y_PORTOPENDRAIN_INVALID.

digitalio→**get_portPolarity()****YDigitalIO****digitalio**→**portPolarity()****digitalio.get_portPolarity()**

Retourne la polarité des bits du port (bitmap).

int **get_portPolarity()** ()

Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée.

Retourne :

un entier représentant la polarité des bits du port (bitmap)

En cas d'erreur, déclenche une exception ou retourne Y_PORTPOLARITY_INVALID.

digitalio→**get_portSize()**

YDigitalIO

digitalio→**portSize()****digitalio.get_portSize()**

Retourne le nombre de bits implémentés dans le port d'E/S.

```
int get_portSize( )
```

Retourne :

un entier représentant le nombre de bits implémentés dans le port d'E/S

En cas d'erreur, déclenche une exception ou retourne Y_PORTSIZE_INVALID.

digitalio→**get_portState()****YDigitalIO****digitalio**→**portState()****digitalio.get_portState()**

Retourne l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

```
int get_portState( )
```

Retourne :

un entier représentant l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite

En cas d'erreur, déclenche une exception ou retourne `Y_PORTSTATE_INVALID`.

digitalio→**get_userdata()**

YDigitalIO

digitalio→**userData()****digitalio.get_userdata()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

digitalio→**isOnline()****digitalio.isOnline()****YDigitalIO**

Vérifie si le module hébergeant le port d'E/S digital est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du port d'E/S digital sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le port d'E/S digital est joignable, `false` sinon

digitalio→**load()****digitalio.load()****YDigitalIO**

Met en cache les valeurs courantes du port d'E/S digital, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**nextDigitalIO()****YDigitalIO****digitalio.nextDigitalIO()**

Continue l'énumération des ports d'E/S digitaux commencée à l'aide de `yFirstDigitalIO()`.

`YDigitalIO` **nextDigitalIO()**

Retourne :

un pointeur sur un objet `YDigitalIO` accessible en ligne, ou `null` lorsque l'énumération est terminée.

digitalio→**pulse()****digitalio.pulse()****YDigitalIO**

Déclenche une impulsion de durée spécifiée sur un bit choisi.

```
int pulse( int bitno, int ms_duration)
```

Le bit va passer à 1 puis automatiquement revenir à 0 après le temps donné.

Paramètres :

bitno index du bit dans le port; le bit de poid faible est à l'index 0

ms_duration durée de l'impulsion désirée, en millisecondes. Notez que la résolution temporelle du module n'est pas garantie à la milliseconde.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→registerValueCallback()**YDigitalIO****digitalio.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

digitalio→**set_bitDirection()****YDigitalIO****digitalio**→**setBitDirection()****digitalio.set_bitDirection()**

Change la direction d'un seul bit du port d'E/S.

```
int set_bitDirection( int bitno, int bitdirection)
```

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

bitdirection nouvelle valeur de la direction, 0=entrée, 1=sortie. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_bitOpenDrain()****YDigitalIO****digitalio**→**setBitOpenDrain()****digitalio.set_bitOpenDrain()**

Change le type d'interface électrique d'un seul bit du port d'E/S.

```
int set_bitOpenDrain( int bitno, int opendrain)
```

Paramètres :

bitno index du bit dans le port; le bit de poid faible est à l'index 0

opendrain 0 pour faire une entrée ou une sortie digitale standard, 1 pour une entrée ou sortie en mode collecteur ouvert (drain ouvert). N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_bitPolarity()****YDigitalIO****digitalio**→**setBitPolarity()****digitalio.set_bitPolarity()**

Change la polarité d'un seul bit du port d'E/S.

```
int set_bitPolarity( int bitno, int bitpolarity)
```

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

bitpolarity nouvelle valeur de la polarité. 0=mode normal, 1=mode inverse. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_bitState()****YDigitalIO****digitalio**→**setBitState()****digitalio.set_bitState()**

Change l'état d'un seul bit du port d'E/S.

```
int set_bitState( int bitno, int bitstate)
```

Paramètres :

bitno index du bit dans le port; le bit de poid faible est à l'index 0

bitstate nouvel état du bit (1 ou 0)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_logicalName()****YDigitalIO****digitalio**→**setLogicalName()****digitalio.set_logicalName()**

Modifie le nom logique du port d'E/S digital.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du port d'E/S digital.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_outputVoltage()****YDigitalIO****digitalio**→**setOutputVoltage()****digitalio.set_outputVoltage()**

Modifie la source de tension utilisée pour piloter les bits en sortie.

```
int set_outputVoltage( int newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après un redémarrage du module.

Paramètres :

newval une valeur parmi `Y_OUTPUTVOLTAGE_USB_5V`, `Y_OUTPUTVOLTAGE_USB_3V` et `Y_OUTPUTVOLTAGE_EXT_V` représentant la source de tension utilisée pour piloter les bits en sortie

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_portDirection()**

YDigitalIO

digitalio→**setPortDirection()**

digitalio.set_portDirection()

Modifie la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie.

```
int set_portDirection( int newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la direction des bits du port (bitmap): 0 représente un bit en entrée, 1 représente un bit en sortie

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_portOpenDrain()****YDigitalIO****digitalio**→**setPortOpenDrain()****digitalio.set_portOpenDrain()**

Modifie le type d'interface électrique de chaque bit du port (bitmap).

```
int set_portOpenDrain( int newval)
```

0 représente une entrée ou une sortie digitale standard, 1 représente une entrée ou sortie en mode collecteur ouvert (drain ouvert). N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant le type d'interface électrique de chaque bit du port (bitmap)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_portPolarity()****YDigitalIO****digitalio**→**setPortPolarity()****digitalio.set_portPolarity()**

Modifie la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée.

```
int set_portPolarity( int newval)
```

Paramètres :

newval un entier représentant la polarité des bits du port (bitmap): Pour chaque bit à 0 l'entrée sortie correspondante fonctionne manière normale, pour chaque bit à 1 elle fonctionne ne manière inversée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_portState()****YDigitalIO****digitalio**→**setPortState()****digitalio.set_portState()**

Modifie l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite.

```
int set_portState( int newval)
```

Seuls les bits configurés en sortie dans `portDirection` sont affectés.

Paramètres :

newval un entier représentant l'état du port d'E/S digital: le bit 0 représente l'input 0 et ainsi de suite

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

digitalio→**set_userdata()**

YDigitalIO

digitalio→**setUserData()**

digitalio.set_userdata()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

void **set_userdata**(Object **data**)

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

digitalio→**toggle_bitState()****YDigitalIO****digitalio.toggle_bitState()**

Inverse l'état d'un seul bit du port d'E/S.

```
int toggle_bitState( int bitno)
```

Paramètres :

bitno index du bit dans le port; le bit de poids faible est à l'index 0

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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');
c++	#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 écran d'après un identifiant donné.

yFirstDisplay()

Commence l'énumération des écran accessibles par la librairie.

Méthodes des objets YDisplay

display→copyLayerContent(srcLayerId, dstLayerId)

Copie le contenu d'un couche d'affichage vers une autre couche.

display→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'écran au format
TYPE (NAME) = SERIAL . FUNCTIONID.

display→fade(brightness, duration)

Change la luminosité de l'écran en douceur, pour produire un effet de fade-in ou fade-out.

display→get_advertisedValue()

Retourne la valeur courante de l'écran (pas plus de 6 caractères).

display→get_brightness()

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

display→get_displayHeight()

Retourne la hauteur de l'écran, en pixels.

display→get_displayLayer(layerId)

Retourne un objet YDisplayLayer utilisable pour dessiner sur la couche d'affichage correspondante.

display→get_displayType()

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

display→get_displayWidth()

Retourne la largeur de l'écran, en pixels.

display→get_enabled()

Retourne vrai si le l'écran est alimenté, faux sinon.

display→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

display→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

display→get_friendlyName()

Retourne un identifiant global de l'écran au format `NOM_MODULE . NOM_FONCTION`.

display→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

display→get_functionId()

Retourne l'identifiant matériel de l'écran, sans référence au module.

display→get_hardwareId()

Retourne l'identifiant matériel unique de l'écran au format `SERIAL . FUNCTIONID`.

display→get_layerCount()

Retourne le nombre des couches affichables disponibles.

display→get_layerHeight()

Retourne la hauteur des couches affichables, en pixels.

display→get_layerWidth()

Retourne la largeur des couches affichables, en pixels.

display→get_logicalName()

Retourne le nom logique de l'écran.

display→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

display→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

display→get_orientation()

Retourne l'orientation sélectionnée pour l'écran.

display→get_startupSeq()

Retourne le nom de la séquence à jouer à la mise sous tension de l'écran.

display→get_userData()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

display→isOnline()

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

display→isOnline_async(callback, context)

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

display→load(msValidity)

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

display→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

display→newSequence()

Enclanche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

display→nextDisplay()

Continue l'énumération des écran commencée à l'aide de `yFirstDisplay()`.

display→pauseSequence(delay_ms)

Attend pour la durée spécifiée (en millisecondes) avant de jouer les commandes suivantes de la séquence active.

display→playSequence(sequenceName)

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes `newSequence()` et `saveSequence()`.

display→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

display→resetAll()

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

display→saveSequence(sequenceName)

Termine l'enregistrement d'une séquence et la sauvegarde sur la mémoire interne de l'écran, sous le nom choisi.

display→set_brightness(newval)

Modifie la luminosité de l'écran.

display→set_enabled(newval)

Modifie l'état d'activité de l'écran.

display→set_logicalName(newval)

Modifie le nom logique de l'écran.

display→set_orientation(newval)

Modifie l'orientation de l'écran.

display→set_startupSeq(newval)

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

display→set_userdata(data)

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

display→stopSequence(sequenceName)

Arrête immédiatement la séquence d'affichage actuellement jouée sur l'écran.

display→swapLayerContent(layerIdA, layerIdB)

Permute le contenu de deux couches d'affichage.

display→upload(pathname, content)

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

display→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YDisplay.FindDisplay()**YDisplay****yFindDisplay()**`YDisplay.FindDisplay()`

Permet de retrouver un ecran d'après un identifiant donné.

`YDisplay FindDisplay(String func)`

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'ecran soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDisplay.isOnline()` pour tester si l'ecran est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence l'ecran sans ambiguïté

Retourne :

un objet de classe `YDisplay` qui permet ensuite de contrôler l'ecran.

YDisplay.FirstDisplay()

YDisplay

yFirstDisplay()YDisplay.FirstDisplay()

Commence l'énumération des écran accessibles par la librairie.

YDisplay **FirstDisplay()**

Utiliser la fonction `YDisplay.nextDisplay()` pour itérer sur les autres écran.

Retourne :

un pointeur sur un objet `YDisplay`, correspondant au premier écran accessible en ligne, ou `null` si il n'y a pas de écran disponibles.

display→copyLayerContent()**YDisplay****display.copyLayerContent ()**

Copie le contenu d'un couche d'affichage vers une autre couche.

```
int copyLayerContent( int srcLayerId, int dstLayerId)
```

La couleur et la transparence de tous les pixels de la couche de destination sont changés pour correspondre à la couche source. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

Paramètres :

srcLayerId l'identifiant de la couche d'origine (un chiffre parmi 0..layerCount-1)

dstLayerId l'identifiant de la couche de destination (un chiffre parmi 0..layerCount-1)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→describe()`display.describe()`**YDisplay**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'ecran au format
`TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant l'ecran (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

display→**fade()****display.fade()****YDisplay**

Change la luminosité de l'écran en douceur, pour produire un effet de fade-in ou fade-out.

```
int fade( int brightness, int duration)
```

Paramètres :

brightness nouvelle valeur de luminosité de l'écran

duration durée en millisecondes de la transition.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**get_advertisedValue()**

YDisplay

display→**advertisedValue()**

display.get_advertisedValue()

Retourne la valeur courante de l'ecran (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante de l'ecran (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

display→**get_brightness()****YDisplay****display**→**brightness()****display.get_brightness()**

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

```
int get_brightness( )
```

Retourne :

un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

En cas d'erreur, déclenche une exception ou retourne `Y_BRIGHTNESS_INVALID`.

display→get_displayHeight()

YDisplay

display→displayHeight()

display.get_displayHeight()

Retourne la hauteur de l'écran, en pixels.

```
int get_displayHeight( )
```

Retourne :

un entier représentant la hauteur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y_DISPLAYHEIGHT_INVALID.

display→**get_displayLayer()****YDisplay****display**→**displayLayer()****display.get_displayLayer()**

Retourne un objet YDisplayLayer utilisable pour dessiner sur la couche d'affichage correspondante.

synchronized YDisplayLayer **get_displayLayer**(int **layerId**)

Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

Paramètres :

layerId l'identifiant de la couche d'affichage désirée (un chiffre parmi 0..layerCount-1)

Retourne :

un objet YDisplayLayer

En cas d'erreur, déclenche une exception ou retourne `null`.

display→**get_displayType()**

YDisplay

display→**displayType()**

display.get_displayType()

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

int **get_displayType()**

Retourne :

une valeur parmi Y_DISPLAYTYPE_MONO, Y_DISPLAYTYPE_GRAY et Y_DISPLAYTYPE_RGB
représentant le type de l'écran: monochrome, niveaux de gris ou couleur

En cas d'erreur, déclenche une exception ou retourne Y_DISPLAYTYPE_INVALID.

display→**get_displayWidth()**
display→**displayWidth()**
display.get_displayWidth()

YDisplay

Retourne la largeur de l'écran, en pixels.

int **get_displayWidth()**

Retourne :

un entier représentant la largeur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y_DISPLAYWIDTH_INVALID.

display→get_enabled()

YDisplay

display→enabled()`display.get_enabled()`

Retourne vrai si le l'ecran est alimenté, faux sinon.

```
int get_enabled( )
```

Retourne :

soit Y_ENABLED_FALSE, soit Y_ENABLED_TRUE, selon vrai si le l'ecran est alimenté, faux sinon

En cas d'erreur, déclenche une exception ou retourne Y_ENABLED_INVALID.

display→**get_errorMessage()****YDisplay****display**→**errorMessage()****display.errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'écran.

display→**get_errorType()**

YDisplay

display→**errorType()****display.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'écran.

display→**get_friendlyName()**
display→**friendlyName()**
display.get_friendlyName()

YDisplay

Retourne un identifiant global de l'écran au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et de l'écran si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'écran (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'écran en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

display→**get_functionDescriptor()**

YDisplay

display→**functionDescriptor()**

display.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

display→**get_functionId()****YDisplay****display**→**functionId()****display.get_functionId()**

Retourne l'identifiant matériel de l'ecran, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant l'ecran (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

display→**get_hardwareId()**

YDisplay

display→**hardwareId()****display.get_hardwareId()**

Retourne l'identifiant matériel unique de l'ecran au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'ecran (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant l'ecran (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

display→**get_layerCount()****YDisplay****display**→**layerCount()****display.get_layerCount()**

Retourne le nombre des couches affichables disponibles.

int **get_layerCount()**

Retourne :

un entier représentant le nombre des couches affichables disponibles

En cas d'erreur, déclenche une exception ou retourne Y_LAYERCOUNT_INVALID.

display→get_layerHeight()

YDisplay

display→layerHeight()

display.get_layerHeight()

Retourne la hauteur des couches affichables, en pixels.

int get_layerHeight()

Retourne :

un entier représentant la hauteur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne Y_LAYERHEIGHT_INVALID.

display→**get_layerWidth()****YDisplay****display**→**layerWidth()****display.get_layerWidth()**

Retourne la largeur des couches affichables, en pixels.

int **get_layerWidth()**

Retourne :

un entier représentant la largeur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne Y_LAYERWIDTH_INVALID.

display→get_logicalName()

YDisplay

display→logicalName()

display.get_logicalName()

Retourne le nom logique de l'ecran.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique de l'ecran.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

display→**get_module()****YDisplay****display**→**module()****display.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

display→**get_orientation()**

YDisplay

display→**orientation()****display.get_orientation()**

Retourne l'orientation sélectionnée pour l'écran.

int **get_orientation()** ()

Retourne :

une valeur parmi Y_ORIENTATION_LEFT, Y_ORIENTATION_UP, Y_ORIENTATION_RIGHT et Y_ORIENTATION_DOWN représentant l'orientation sélectionnée pour l'écran

En cas d'erreur, déclenche une exception ou retourne Y_ORIENTATION_INVALID.

display→**get_startupSeq()****YDisplay****display**→**startupSeq()****display.get_startupSeq()**

Retourne le nom de la séquence à jouer à la mise sous tension de l'écran.

String **get_startupSeq()**

Retourne :

une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

En cas d'erreur, déclenche une exception ou retourne Y_STARTUPSEQ_INVALID.

display→**get_userData()**

YDisplay

display→**userData()****display.get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

Object **get_userData()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

display→**isOnline()****display.isOnline()****YDisplay**

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache de l'écran sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'écran est joignable, `false` sinon

display→load()`display.load()`**YDisplay**

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**newSequence()****display.newSequence()****YDisplay**

Enclanche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

int newSequence()

Le nom de la séquence sera donné au moment de l'appel à `saveSequence()`, une fois la séquence terminée.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**nextDisplay()****display.nextDisplay()**

YDisplay

Continue l'énumération des écran commencée à l'aide de `yFirstDisplay()`.

YDisplay **nextDisplay()**

Retourne :

un pointeur sur un objet `YDisplay` accessible en ligne, ou `null` lorsque l'énumération est terminée.

display→pauseSequence()**YDisplay****display.pauseSequence()**

Attend pour la durée spécifiée (en millisecondes) avant de jouer les commandes suivantes de la séquence active.

```
int pauseSequence( int delay_ms)
```

Cette méthode peut être utilisée lors de l'enregistrement d'une séquence d'affichage, pour insérer une attente mesurée lors de l'exécution (mais sans effet immédiat). Cette méthode peut aussi être appelée dynamiquement pendant l'exécution d'une séquence enregistrée, pour suspendre temporairement ou reprendre l'exécution. Pour annuler une attente, appelez simplement la méthode avec une attente de zéro.

Paramètres :

delay_ms la durée de l'attente, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→playSequence()`display.playSequence()`**YDisplay**

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes `newSequence()` et `saveSequence()`.

```
int playSequence( String sequenceName)
```

Paramètres :

sequenceName le nom de la nouvelle séquence créée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→registerValueCallback()**YDisplay****display.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

display→**resetAll()****display.resetAll()**

YDisplay

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

int resetAll()

Utiliser cette fonction dans une sequence va tuer stopper l'affichage de la sequence: ne pas utiliser cette fonction pour réinitialiser l'écran au début d'une séquence.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**saveSequence()****display.saveSequence ()****YDisplay**

Termine l'enregistrement d'une séquence et la sauvegarde sur la mémoire interne de l'écran, sous le nom choisi.

```
int saveSequence( String sequenceName)
```

La séquence peut être rejouée ultérieurement à l'aide de la méthode `playSequence ()`.

Paramètres :

sequenceName le nom de la nouvelle séquence créée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**set_brightness()**

YDisplay

display→**setBrightness()**

display.set_brightness()

Modifie la luminosité de l'écran.

```
int set_brightness( int newval)
```

Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la luminosité de l'écran

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**set_enabled()****YDisplay****display**→**setEnabled()****display.set_enabled()**

Modifie l'état d'activité de l'écran.

```
int set_enabled( int newval)
```

Paramètres :

newval soit Y_ENABLED_FALSE, soit Y_ENABLED_TRUE, selon l'état d'activité de l'écran

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**set_logicalName()****YDisplay****display**→**setLogicalName()****display.set_logicalName()**

Modifie le nom logique de l'ecran.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'ecran.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**set_orientation()**
display→**setOrientation()**
display.set_orientation()

YDisplay

Modifie l'orientation de l'écran.

```
int set_orientation( int newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur parmi `Y_ORIENTATION_LEFT`, `Y_ORIENTATION_UP`, `Y_ORIENTATION_RIGHT` et `Y_ORIENTATION_DOWN` représentant l'orientation de l'écran

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**set_startupSeq()**

YDisplay

display→**setStartupSeq()**

display.set_startupSeq()

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

```
int set_startupSeq( String newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→**set_userdata()****YDisplay****display**→**setUserData()****display.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

display→**stopSequence()****display.stopSequence()****YDisplay**

Arrête immédiatement la séquence d'affichage actuellement jouée sur l'écran.

```
int stopSequence( )
```

L'affichage est laissé tel quel.

Paramètres :

sequenceName le nom de la nouvelle séquence créée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→swapLayerContent()**YDisplay****display.swapLayerContent ()**

Permute le contenu de deux couches d'affichage.

```
int swapLayerContent( int layerIdA, int layerIdB)
```

La couleur et la transparence de tous les pixels des deux couches sont permutées. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. En particulier, la visibilité des deux couches reste inchangée. Cela permet d'implémenter très efficacement un affichage par double-buffering, en utilisant une couche cachée et une couche visible. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

Paramètres :

layerIdA l'identifiant de la première couche (un chiffre parmi 0..layerCount-1)

layerIdB l'identifiant de la deuxième couche (un chiffre parmi 0..layerCount-1)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

display→upload()`display.upload()`**YDisplay**

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

```
int upload( String pathname)
```

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

Paramètres :

pathname nom complet du fichier, y compris le chemin d'accès.

content contenu du fichier à télécharger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.15. Interface des objets DisplayLayer

Un DisplayLayer est une couche de contenu affichable (images, texte, etc.). Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_display.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YDisplay = yoctolib.YDisplay;
php	require_once('yocto_display.php');
c++	#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, bgcolor)

Dessine un bitmap à la position spécifiée de la couche.

displaylayer→drawCircle(x, y, r)

Dessine un cercle vide à une position spécifiée.

displaylayer→drawDisc(x, y, r)

Dessine un disque plein à une position spécifiée.

displaylayer→drawImage(x, y, imagename)

Dessine une image GIF à la position spécifiée de la couche.

displaylayer→drawPixel(x, y)

Dessine un pixel unique à une position spécifiée.

displaylayer→drawRect(x1, y1, x2, y2)

Dessine un rectangle vide à une position spécifiée.

displaylayer→drawText(x, y, anchor, text)

Affiche un texte à la position spécifiée de la couche.

displaylayer→get_display()

Retourne l'YDisplay parent.

displaylayer→get_displayHeight()

Retourne la hauteur de l'écran, en pixels.

displaylayer→get_displayWidth()

Retourne la largeur de l'écran, en pixels.

displaylayer→get_layerHeight()

Retourne la hauteur des couches affichables, en pixels.

displaylayer→get_layerWidth()

Retourne la largeur des couches affichables, en pixels.

displaylayer→hide()

Cache la couche de dessin.

displaylayer→lineTo(x, y)

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

displaylayer→moveTo(x, y)

Déplace le point de dessin courant de cette couche à la position spécifiée.

displaylayer→reset()

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

displaylayer→selectColorPen(color)

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

displaylayer→selectEraser()

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de copie d'images bitmaps.

displaylayer→selectFont(fontname)

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

displaylayer→selectGrayPen(graylevel)

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

displaylayer→setAntialiasingMode(mode)

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

displaylayer→setConsoleBackground(bgcol)

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

displaylayer→setConsoleMargins(x1, y1, x2, y2)

Configure les marges d'affichage pour la fonction `consoleOut`.

displaylayer→setConsoleWordWrap(wordwrap)

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

displaylayer→setLayerPosition(x, y, scrollTime)

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

displaylayer→unhide()

Affiche la couche.

displaylayer→**clear()****displaylayer.clear()****YDisplayLayer**

Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.

int **clear()** **()**

Cette méthode ne change pas les réglages de la couche. Si vous désirez remettre la couche dans son état initial, utilisez plutôt la méthode `reset()`.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→clearConsole()

YDisplayLayer

displaylayer.clearConsole()

Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.

int **clearConsole**()

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→consoleOut()**YDisplayLayer****displaylayer.consoleOut()**

Affiche un message dans la zone de console, et déplace le curseur de la console à la fin du texte.

```
int consoleOut( String text)
```

Le curseur revient automatiquement en début de ligne suivante lorsqu'un saut de ligne est rencontré, ou lorsque la marge droite est atteinte. Lorsque le texte à afficher s'apprête à dépasser la marge inférieure, le contenu de la zone de console est automatiquement décalé vers le haut afin de laisser la place à la nouvelle ligne de texte.

Paramètres :

text le message à afficher

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→drawBar()`displaylayer.drawBar()`**YDisplayLayer**

Dessine un rectangle plein à une position spécifiée.

```
int drawBar( int x1, int y1, int x2, int y2)
```

Paramètres :

- x1** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle
- y1** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle
- x2** la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle
- y2** la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→drawBitmap()**YDisplayLayer****displaylayer.drawBitmap()**

Dessine un bitmap à la position spécifiée de la couche.

```
int drawBitmap( int x, int y, int w, int bgcol)
```

Le bitmap est passé sous forme d'un objet binaire, où chaque bit correspond à un pixel, de gauche à droite et de haut en bas. Le bit de poids fort de chaque octet correspond au pixel de gauche, et le bit de poids faible au pixel le plus à droite. Les bits à 1 sont dessinés avec la couleur active de la couche. Les bits à 0 avec la couleur de fond spécifiée, sauf si la valeur -1 a été choisie, auquel cas ils ne sont pas dessinés (ils sont considérés comme transparents). Chaque ligne commence sur un nouvel octet. La hauteur du bitmap est donnée implicitement par la taille de l'objet binaire.

Paramètres :

- x** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du bitmap
- y** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du bitmap
- w** la largeur du bitmap, en pixels
- bitmap** l'objet binaire contenant le bitmap
- bgcol** le niveau de gris à utiliser pour les bits à zéro (0 = noir, 255 = blanc), ou -1 pour laisser les pixels inchangés

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→drawCircle()**YDisplayLayer****displaylayer.drawCircle()**

Dessine un cercle vide à une position spécifiée.

```
int drawCircle( int x, int y, int r)
```

Paramètres :

- x** la distance en pixels depuis la gauche de la couche jusqu'au centre du cercle
- y** la distance en pixels depuis le haut de la couche jusqu'au centre du cercle
- r** le rayon du cercle, en pixels

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→drawDisc()**YDisplayLayer****displaylayer.drawDisc()**

Dessine un disque plein à une position spécifiée.

```
int drawDisc( int x, int y, int r)
```

Paramètres :

- x** la distance en pixels depuis la gauche de la couche jusqu'au centre du disque
- y** la distance en pixels depuis le haut de la couche jusqu'au centre du disque
- r** le rayon du disque, en pixels

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→drawImage()**YDisplayLayer****displaylayer.drawImage()**

Dessine une image GIF à la position spécifiée de la couche.

```
int drawImage( int x, int y, String imagename)
```

L'image GIF doit avoir été préalablement préchargée dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une image bitmap, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier d'image manquant ou d'un format de fichier invalide.

Paramètres :

- x** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche de l'image
- y** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur de l'image
- imagename** le nom du fichier GIF à afficher

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→drawPixel()**YDisplayLayer****displaylayer.drawPixel()**

Dessine un pixel unique à une position spécifiée.

```
int drawPixel( int x, int y)
```

Paramètres :

- x** la distance en pixels depuis la gauche de la couche
- y** la distance en pixels depuis le haut de la couche

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→drawRect()**YDisplayLayer****displaylayer.drawRect()**

Dessine un rectangle vide à une position spécifiée.

```
int drawRect( int x1, int y1, int x2, int y2)
```

Paramètres :

- x1** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle
- y1** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle
- x2** la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle
- y2** la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→drawText()**YDisplayLayer****displaylayer.drawText()**

Affiche un texte à la position spécifiée de la couche.

```
int drawText( int x, int y, ALIGN anchor, String text)
```

Le point du texte qui sera aligné sur la position spécifiée est appelé point d'ancrage, et peut être choisi parmi plusieurs options.

Paramètres :

- x** la distance en pixels depuis la gauche de la couche jusqu'au point d'ancrage du texte
- y** la distance en pixels depuis le haut de la couche jusqu'au point d'ancrage du texte
- anchor** le point d'ancrage du texte, choisi parmi l'énumération Y_ALIGN: Y_ALIGN_TOP_LEFT, Y_ALIGN_CENTER_LEFT, Y_ALIGN_BASELINE_LEFT, Y_ALIGN_BOTTOM_LEFT, Y_ALIGN_TOP_CENTER, Y_ALIGN_CENTER, Y_ALIGN_BASELINE_CENTER, Y_ALIGN_BOTTOM_CENTER, Y_ALIGN_TOP_DECIMAL, Y_ALIGN_CENTER_DECIMAL, Y_ALIGN_BASELINE_DECIMAL, Y_ALIGN_BOTTOM_DECIMAL, Y_ALIGN_TOP_RIGHT, Y_ALIGN_CENTER_RIGHT, Y_ALIGN_BASELINE_RIGHT, Y_ALIGN_BOTTOM_RIGHT.
- text** le texte à afficher

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→get_display()

YDisplayLayer

displaylayer→display()

displaylayer.get_display()

Retourne l'YDisplay parent.

YDisplay **get_display()**

Retourne l'objet YDisplay parent du YDisplayLayer courant.

Retourne :

un objet YDisplay

displaylayer→**get_displayHeight()****YDisplayLayer****displaylayer**→**displayHeight()****displaylayer.get_displayHeight()**

Retourne la hauteur de l'écran, en pixels.

int **get_displayHeight()**

Retourne :

un entier représentant la hauteur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y_DISPLAYHEIGHT_INVALID.

displaylayer→get_displayWidth()
displaylayer→displayWidth()
displaylayer.get_displayWidth()

YDisplayLayer

Retourne la largeur de l'écran, en pixels.

int get_displayWidth()

Retourne :

un entier représentant la largeur de l'écran, en pixels

En cas d'erreur, déclenche une exception ou retourne Y_DISPLAYWIDTH_INVALID.

displaylayer→**get_layerHeight()****YDisplayLayer****displaylayer**→**layerHeight()****displaylayer.get_layerHeight()**

Retourne la hauteur des couches affichables, en pixels.

int **get_layerHeight()**

Retourne :

un entier représentant la hauteur des couches affichables, en pixels. En cas d'erreur, déclenche une exception ou retourne Y_LAYERHEIGHT_INVALID.

displaylayer→get_layerWidth()

YDisplayLayer

displaylayer→layerWidth()

displaylayer.get_layerWidth()

Retourne la largeur des couches affichables, en pixels.

int get_layerWidth()

Retourne :

un entier représentant la largeur des couches affichables, en pixels

En cas d'erreur, déclenche une exception ou retourne Y_LAYERWIDTH_INVALID.

displaylayer→**hide()****displaylayer.hide()****YDisplayLayer**

Cache la couche de dessin.

```
int hide( )
```

L'état de la couche est préservé, mais la couche ne sera plus affichée à l'écran jusqu'au prochain appel à `unhide()`. Le fait de cacher la couche améliore les performances de toutes les primitives d'affichage, car il évite de consacrer inutilement des cycles de calcul à afficher les états intermédiaires (technique de double-buffering).

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→lineTo()`displaylayer.lineTo()`**YDisplayLayer**

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

```
int lineTo( int x, int y)
```

Le pixel final spécifié est inclus dans la ligne dessinée. Le point de dessin courant est déplacé à au point final de la ligne.

Paramètres :

- x** la distance en pixels depuis la gauche de la couche jusqu'au point final
- y** la distance en pixels depuis le haut de la couche jusqu'au point final

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**moveTo()****displaylayer.moveTo()****YDisplayLayer**

Déplace le point de dessin courant de cette couche à la position spécifiée.

```
int moveTo( int x, int y)
```

Paramètres :

- x** la distance en pixels depuis la gauche de la couche de dessin
- y** la distance en pixels depuis le haut de la couche de dessin

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→reset()`displaylayer.reset()`**YDisplayLayer**

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

```
int reset( )
```

Réinitialise la position du point de dessin courant au coin supérieur gauche, et la couleur de dessin à la valeur la plus lumineuse. Si vous désirez simplement effacer le contenu de la couche, utilisez plutôt la méthode `clear()`.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**selectColorPen()****YDisplayLayer****displaylayer.selectColorPen()**

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

```
int selectColorPen( int color)
```

La couleur est fournie sous forme de couleur RGB. Pour les écrans monochromes ou en niveaux de gris, la couleur est automatiquement ramenée dans les valeurs permises.

Paramètres :

color la couleur RGB désirée (sous forme d'entier 24 bits)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→selectEraser()**YDisplayLayer****displaylayer.selectEraser()**

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de copie d'images bitmaps.

int selectEraser()

Tous les points dessinés à la gomme redeviennent transparents (comme ils l'étaient lorsque la couche était vide), rendant ainsi visibles les couches inférieures.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→selectFont()**YDisplayLayer****displaylayer.selectFont()**

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

```
int selectFont( String fontname)
```

La police est spécifiée par le nom de son fichier. Vous pouvez utiliser l'une des polices prédéfinies dans le module, ou une autre police que vous avez préalablement préchargé dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une police de caractères, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier de police manquant ou d'un format de fichier invalide.

Paramètres :

fontname le nom du fichier définissant la police de caractères

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**selectGrayPen()****YDisplayLayer****displaylayer.selectGrayPen()**

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

```
int selectGrayPen( int graylevel)
```

Le niveau de gris est fourni sous forme d'un chiffre allant de 0 (noir) à 255 (blanc, ou la couleur la plus claire de l'écran, quelle qu'elle soit). Pour les écrans monochromes (sans niveaux de gris), toute valeur inférieure à 128 conduit à un point noir, et toute valeur supérieure ou égale à 128 devient un point lumineux.

Paramètres :

graylevel le niveau de gris désiré, de 0 à 255

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→setAntialiasingMode()**YDisplayLayer****displaylayer.setAntialiasingMode()**

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

int setAntialiasingMode(boolean mode)

L'anti-aliasing est atténue la pixelisation des images lorsqu'on regarde l'écran depuis une distance suffisante, mais peut aussi donner parfois une impression de flou lorsque l'écran est regardé de très près. Au final, c'est un choix esthétique qui vous revient. L'anti-aliasing est activé par défaut pour les écrans en niveaux de gris et les écrans couleurs, mais vous pouvez le désactiver si vous préférez. Ce réglage n'a pas d'effet sur les écrans monochromes.

Paramètres :

mode true pour activer l'antialiasing, false pour le désactiver.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→setConsoleBackground()

YDisplayLayer

displaylayer.setConsoleBackground()

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

```
int setConsoleBackground( int bgcol)
```

Paramètres :

bgcol le niveau de gris à utiliser pour le fond lors de défilement (0 = noir, 255 = blanc), ou -1 pour un fond transparent

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→setConsoleMargins()**YDisplayLayer****displaylayer.setConsoleMargins()**

Configure les marges d'affichage pour la fonction `consoleOut`.

```
int setConsoleMargins( int x1, int y1, int x2, int y2)
```

Paramètres :

- x1** la distance en pixels depuis la gauche de la couche jusqu'à la marge gauche
- y1** la distance en pixels depuis le haut de la couche jusqu'à la marge supérieure
- x2** la distance en pixels depuis la gauche de la couche jusqu'à la marge droite
- y2** la distance en pixels depuis le haut de la couche jusqu'à la marge inférieure

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**setConsoleWordWrap()****YDisplayLayer****displaylayer.setConsoleWordWrap()**

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

```
int setConsoleWordWrap( boolean wordwrap )
```

Paramètres :

wordwrap `true` pour retourner à la ligne entre les mots seulement, `false` pour retourner à l'extrême droite de chaque ligne.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→setLayerPosition()**YDisplayLayer****displaylayer.setLayerPosition()**

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

```
int setLayerPosition( int x, int y, int scrollTime)
```

Lorsqu'une durée de défilement est configurée, la position d'affichage de la couche est automatiquement mise à jour durant les millisecondes suivantes pour animer le déplacement.

Paramètres :

- x** la distance en pixels depuis la gauche de l'écran jusqu'à l'origine de la couche.
- y** la distance en pixels depuis le haut de l'écran jusqu'à l'origine de la couche.
- scrollTime** durée en millisecondes du déplacement, ou 0 si le déplacement doit être immédiat.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

displaylayer→**unhide()****displaylayer.unhide()**

YDisplayLayer

Affiche la couche.

int unhide()

Affiche a nouveau la couche après la command hide.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.16. Interface de contrôle de l'alimentation

La librairie de programmation Yoctopuce permet de contrôler la source d'alimentation qui doit être utilisée pour les fonctions du module consommant beaucoup de courant. Le module est par ailleurs capable de couper automatiquement l'alimentation externe lorsqu'il détecte que la tension a trop chuté (batterie épuisée).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_dualpower.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YDualPower = yoctolib.YDualPower;
php	require_once('yocto_dualpower.php');
c++	#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 <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
dualpower→get_module_async(callback, context)	Retourne l'objet <code>YModule</code> 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 <code>userData</code> , précédemment stocké à l'aide de la méthode <code>set_userData</code> .
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 <code>yFirstDualPower()</code> .
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 <code>userData</code> de la fonction, afin de le retrouver plus tard à l'aide de la méthode <code>get_userData</code> .
dualpower→wait_async(callback, context)	Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YDualPower.FindDualPower()**YDualPower****yFindDualPower()**`YDualPower.FindDualPower()`

Permet de retrouver un contrôle d'alimentation d'après un identifiant donné.

`YDualPower` **FindDualPower**(`String func`)

L'identifiant peut être spécifié sous plusieurs formes:

- `NomLogiqueFonction`
- `NoSerieModule.IdentifiantFonction`
- `NoSerieModule.NomLogiqueFonction`
- `NomLogiqueModule.IdentifiantMatériel`
- `NomLogiqueModule.NomLogiqueFonction`

Cette fonction n'exige pas que le contrôle d'alimentation soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDualPower.isOnline()` pour tester si le contrôle d'alimentation est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le contrôle d'alimentation sans ambiguïté

Retourne :

un objet de classe `YDualPower` qui permet ensuite de contrôler le contrôle d'alimentation.

YDualPower.FirstDualPower()

YDualPower

yFirstDualPower()`YDualPower.FirstDualPower()`

Commence l'énumération des contrôles d'alimentation accessibles par la librairie.

`YDualPower` **FirstDualPower()**

Utiliser la fonction `YDualPower.nextDualPower()` pour itérer sur les autres contrôles d'alimentation.

Retourne :

un pointeur sur un objet `YDualPower`, correspondant au premier contrôle d'alimentation accessible en ligne, ou `null` si il n'y a pas de contrôles d'alimentation disponibles.

dualpower→describe()`dualpower.describe()`**YDualPower**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'alimentation au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String `describe()`

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le contrôle d'alimentation (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

dualpower→**get_advertisedValue()**

YDualPower

dualpower→**advertisedValue()**

dualpower.get_advertisedValue()

Retourne la valeur courante du contrôle d'alimentation (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du contrôle d'alimentation (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

dualpower→get_errorMessage()**YDualPower****dualpower→errorMessage()****dualpower.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'alimentation.

dualpower→get_errorType()

YDualPower

dualpower→errorType()

dualpower.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'alimentation.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'alimentation.

dualpower→get_extVoltage()**YDualPower****dualpower→extVoltage()****dualpower.get_extVoltage()**

Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.

int **get_extVoltage()**

Retourne :

un entier représentant la tension mesurée sur l'alimentation de puissance externe, en millivolts

En cas d'erreur, déclenche une exception ou retourne Y_EXTVOLTAGE_INVALID.

dualpower→**get_friendlyName()****YDualPower****dualpower**→**friendlyName()****dualpower.get_friendlyName()**

Retourne un identifiant global du contrôle d'alimentation au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du contrôle d'alimentation si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du contrôle d'alimentation (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le contrôle d'alimentation en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

dualpower→**get_functionDescriptor()****YDualPower****dualpower**→**functionDescriptor()****dualpower.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

dualpower→**get_functionId()**

YDualPower

dualpower→**functionId()**

dualpower.get_functionId()

Retourne l'identifiant matériel du contrôle d'alimentation, sans référence au module.

String **get_functionId()** ()

Par exemple relay1.

Retourne :

une chaîne de caractères identifiant le contrôle d'alimentation (ex: relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FUNCTIONID_INVALID.

dualpower→**get_hardwareId()****YDualPower****dualpower**→**hardwareId()****dualpower.get_hardwareId()**

Retourne l'identifiant matériel unique du contrôle d'alimentation au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du contrôle d'alimentation (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le contrôle d'alimentation (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

dualpower→get_logicalName()

YDualPower

dualpower→logicalName()

dualpower.get_logicalName()

Retourne le nom logique du contrôle d'alimentation.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du contrôle d'alimentation.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

dualpower→**get_module()****YDualPower****dualpower**→**module()**`dualpower.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

dualpower→get_powerControl()

YDualPower

dualpower→powerControl()

dualpower.get_powerControl()

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

int get_powerControl()

Retourne :

une valeur parmi Y_POWERCONTROL_AUTO, Y_POWERCONTROL_FROM_USB, Y_POWERCONTROL_FROM_EXT et Y_POWERCONTROL_OFF représentant le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant

En cas d'erreur, déclenche une exception ou retourne Y_POWERCONTROL_INVALID.

dualpower→get_powerState()**YDualPower****dualpower→powerState()****dualpower.get_powerState()**

Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

```
int get_powerState( )
```

Retourne :

une valeur parmi Y_POWERSTATE_OFF, Y_POWERSTATE_FROM_USB et Y_POWERSTATE_FROM_EXT représentant la source d'alimentation active pour les fonctions du module consommant beaucoup de courant

En cas d'erreur, déclenche une exception ou retourne Y_POWERSTATE_INVALID.

dualpower→**get_userdata()**

YDualPower

dualpower→**userData()**`dualpower.get_userdata()`

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

Object `get_userdata()`

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

dualpower→**isOnline()****dualpower.isOnline()****YDualPower**

Vérifie si le module hébergeant le contrôle d'alimentation est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du contrôle d'alimentation sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le contrôle d'alimentation est joignable, `false` sinon

dualpower→load()`dualpower.load()`**YDualPower**

Met en cache les valeurs courantes du contrôle d'alimentation, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

dualpower→**nextDualPower()****YDualPower****dualpower.nextDualPower()**

Continue l'énumération des contrôles d'alimentation commencée à l'aide de `yFirstDualPower()`.

YDualPower **nextDualPower()**

Retourne :

un pointeur sur un objet `YDualPower` accessible en ligne, ou `null` lorsque l'énumération est terminée.

dualpower→registerValueCallback()**YDualPower****dualpower.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

dualpower→**set_logicalName()****YDualPower****dualpower**→**setLogicalName()****dualpower.set_logicalName()**

Modifie le nom logique du contrôle d'alimentation.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du contrôle d'alimentation.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

dualpower→set_powerControl()

YDualPower

dualpower→setPowerControl()

dualpower.set_powerControl()

Modifie le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

```
int set_powerControl( int newval)
```

Paramètres :

newval une valeur parmi Y_POWERCONTROL_AUTO, Y_POWERCONTROL_FROM_USB, Y_POWERCONTROL_FROM_EXT et Y_POWERCONTROL_OFF représentant le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

dualpower→**set_userdata()****YDualPower****dualpower**→**setUserData()****dualpower.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

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	<code><script type='text/javascript' src='yocto_files.js'></script></code>
node.js	<code>var yoctolib = require('yoctolib');</code> <code>var YFiles = yoctolib.YFiles;</code>
php	<code>require_once('yocto_files.php');</code>
c++	<code>#include "yocto_files.h"</code>
m	<code>#import "yocto_files.h"</code>
pas	<code>uses yocto_files;</code>
vb	<code>yocto_files.vb</code>
cs	<code>yocto_files.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YFiles;</code>
py	<code>from yocto_files import *</code>

Fonction globales

yFindFiles(func)

Permet de retrouver un système de fichier d'après un identifiant donné.

yFirstFiles()

Commence l'énumération des système de fichier accessibles par la librairie.

Méthodes des objets YFiles

files→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

files→download(pathname)

Télécharge le fichier choisi du filesystem et retourne son contenu.

files→download_async(pathname, callback, context)

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

files→format_fs()

Rétabli le système de fichier dans on état original, défragmenté.

files→get_advertisedValue()

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

files→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

files→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

files→get_filesCount()

Retourne le nombre de fichiers présents dans le système de fichier.

files→get_freeSpace()

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

files→get_friendlyName()

Retourne un identifiant global du système de fichier au format `NOM_MODULE . NOM_FONCTION`.

files→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

files→get_functionId()

Retourne l'identifiant matériel du système de fichier, sans référence au module.

files→get_hardwareId()

Retourne l'identifiant matériel unique du système de fichier au format SERIAL . FUNCTIONID.

files→get_list(pattern)

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

files→get_logicalName()

Retourne le nom logique du système de fichier.

files→get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

files→get_module_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

files→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

files→isOnline()

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

files→isOnline_async(callback, context)

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

files→load(msValidity)

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

files→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

files→nextFiles()

Continue l'énumération des système de fichier commencée à l'aide de yFirstFiles().

files→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

files→remove(pathname)

Efface un fichier, spécifié par son path complet, du système de fichier.

files→set_logicalName(newval)

Modifie le nom logique du système de fichier.

files→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userData.

files→upload(pathname, content)

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

files→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YFiles.FindFiles()**YFiles****yFindFiles()****YFiles.FindFiles()**

Permet de retrouver un système de fichier d'après un identifiant donné.

YFiles **FindFiles**(String **func**)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le système de fichier soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YFiles.isOnline()` pour tester si le système de fichier est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le système de fichier sans ambiguïté

Retourne :

un objet de classe `YFiles` qui permet ensuite de contrôler le système de fichier.

YFiles.FirstFiles()**YFiles****yFirstFiles()****YFiles.FirstFiles()**

Commence l'énumération des système de fichier accessibles par la librairie.

YFiles **FirstFiles()**

Utiliser la fonction `YFiles.nextFiles()` pour itérer sur les autres système de fichier.

Retourne :

un pointeur sur un objet `YFiles`, correspondant au premier système de fichier accessible en ligne, ou `null` si il n'y a pas de système de fichier disponibles.

files→describe()`files.describe()`**YFiles**

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le système de fichier (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

files→**format_fs()****files.format_fs()****YFiles**

Rétabli le système de fichier dans on état original, défragmenté.

int **format_fs**()

entièrement vide. Tous les fichiers précédemment chargés sont irrémédiablement effacés.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→**get_advertisedValue()**

YFiles

files→**advertisedValue()**

files.get_advertisedValue()

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du système de fichier (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

files→**get_errorMessage()****YFiles****files**→**errorMessage()****files.errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du système de fichier.

files→**get_errorType()**

YFiles

files→**errorType()****files.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

int **get_errorType()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du système de fichier.

files→**get_filesCount()****YFiles****files**→**filesCount()****files.get_filesCount()**

Retourne le nombre de fichiers présents dans le système de fichier.

int **get_filesCount()**

Retourne :

un entier représentant le nombre de fichiers présents dans le système de fichier

En cas d'erreur, déclenche une exception ou retourne `Y_FILESCOUNT_INVALID`.

files→**get_freeSpace()**

YFiles

files→**freeSpace()****files.get_freeSpace()**

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

int **get_freeSpace()** ()

Retourne :

un entier représentant l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets

En cas d'erreur, déclenche une exception ou retourne Y_FREESPACE_INVALID.

files→**get_friendlyName()****YFiles****files**→**friendlyName()****files.get_friendlyName()**

Retourne un identifiant global du système de fichier au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du système de fichier si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du système de fichier (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le système de fichier en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

files→get_functionDescriptor()

YFiles

files→functionDescriptor()

files.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

files→**get_functionId()****YFiles****files**→**functionId()****files.get_functionId()**

Retourne l'identifiant matériel du système de fichier, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le système de fichier (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

files→**get_hardwareId()**

YFiles

files→**hardwareId()****files.get_hardwareId()**

Retourne l'identifiant matériel unique du système de fichier au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du système de fichier (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le système de fichier (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

files→**get_list()****YFiles****files**→**list()****files.get_list()**

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

```
ArrayList<YFileRecord> get_list( String pattern)
```

Paramètres :

pattern un filtre optionel sur les noms de fichiers retournés, pouvant contenir des astérisques et des points d'interrogations comme jokers. Si le pattern fourni est vide, tous les fichiers sont retournés.

Retourne :

une liste d'objets YFileRecord, contenant le nom complet (y compris le chemin d'accès), la taille en octets et le CRC 32-bit du contenu du fichier.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

files→**get_logicalName()**

YFiles

files→**logicalName()****files.get_logicalName()**

Retourne le nom logique du système de fichier.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du système de fichier.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

files→**get_module()****YFiles****files**→**module()**`files.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

files→**get_userdata()**

YFiles

files→**userData()****files.get_userdata()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.

Object **get_userdata()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

files→**isOnline()****files.isOnline()****YFiles**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache du système de fichier sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le système de fichier est joignable, `false` sinon

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→**nextFiles()****files.nextFiles()****YFiles**

Continue l'énumération des système de fichier commencée à l'aide de `yFirstFiles()`.

YFiles **nextFiles()**

Retourne :

un pointeur sur un objet `YFiles` accessible en ligne, ou `null` lorsque l'énumération est terminée.

files→registerValueCallback()**YFiles****files.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

files→remove()`files.remove()`**YFiles**

Efface un fichier, spécifié par son path complet, du système de fichier.

```
int remove( String pathname)
```

A cause de la fragmentation, l'effacement d'un fichier ne libère pas toujours la totalité de l'espace qu'il occupe. Par contre, la ré-écriture d'un fichier du même nom récupérera dans tout les cas l'espace qui n'aurait éventuellement pas été libéré. Pour s'assurer de libérer la totalité de l'espace du système de fichier, utilisez la fonction `format_fs`.

Paramètres :

pathname nom complet du fichier, y compris le chemin d'accès.

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→**set_logicalName()****YFiles****files**→**setLogicalName()****files.set_logicalName()**

Modifie le nom logique du système de fichier.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du système de fichier.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→**set_userdata()****YFiles****files**→**setUserData()****files.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

files→**upload()****files.upload()****YFiles**

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

```
int upload( String pathname)
```

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

Paramètres :

pathname nom complet du fichier, y compris le chemin d'accès.

content contenu du fichier à télécharger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.18. Interface de la fonction GenericSensor

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_genericsensor.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YGenericSensor = yoctolib.YGenericSensor;
php	require_once('yocto_genericsensor.php');
c++	#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 mesure depuis le démarrage du module.

genericsensor→get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

genericsensor→get_logicalName()

Retourne le nom logique du capteur générique.

genericsensor→get_lowestValue()

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

genericsensor→get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

genericsensor→get_module_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

genericsensor→get_recordedData(startTime, endTime)

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

genericsensor→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

genericsensor→get_resolution()

Retourne la résolution des valeurs mesurées.

genericsensor→get_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 mesure est exprimée.

genericsensor→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

genericsensor→get_valueRange()

Retourne la plage de valeurs physiques mesurés par le capteur.

genericsensor→isOnline()

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

genericsensor→isOnline_async(callback, context)

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

genericsensor→load(msValidity)

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

genericsensor→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode calibrateFromPoints.

genericsensor→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

genericsensor→nextGenericSensor()

Continue l'énumération des capteurs génériques commencée à l'aide de `yFirstGenericSensor()`.

genericsensor→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

genericsensor→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

genericsensor→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

genericsensor→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

genericsensor→set_logicalName(newval)

Modifie le nom logique du capteur générique.

genericsensor→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

genericsensor→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

genericsensor→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

genericsensor→set_signalBias(newval)

Modifie le biais du signal électrique pour la correction du point zéro.

genericsensor→set_signalRange(newval)

Modifie la plage de signal électrique utilisée par le capteur.

genericsensor→set_unit(newval)

Change l'unité dans laquelle la valeur mesurée est exprimée.

genericsensor→set_userData(data)

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

genericsensor→set_valueRange(newval)

Modifie la plage de valeurs physiques mesurés par le capteur.

genericsensor→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

genericsensor→zeroAdjust()

Ajuste le biais du signal de sorte à ce que la valeur actuelle du signal soit interprétée comme zéro (tare).

YGenericSensor.FindGenericSensor()**YGenericSensor****yFindGenericSensor()****YGenericSensor.FindGenericSensor()**

Permet de retrouver un capteur générique d'après un identifiant donné.

YGenericSensor **FindGenericSensor**(String **func**)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur générique soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGenericSensor.isOnline()` pour tester si le capteur générique est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur générique sans ambiguïté

Retourne :

un objet de classe `YGenericSensor` qui permet ensuite de contrôler le capteur générique.

YGenericSensor.FirstGenericSensor()**YGenericSensor****yFirstGenericSensor()****YGenericSensor.FirstGenericSensor()**

Commence l'énumération des capteurs génériques accessibles par la librairie.

YGenericSensor **FirstGenericSensor()**

Utiliser la fonction `YGenericSensor.nextGenericSensor()` pour itérer sur les autres capteurs génériques.

Retourne :

un pointeur sur un objet `YGenericSensor`, correspondant au premier capteur générique accessible en ligne, ou `null` si il n'y a pas de capteurs génériques disponibles.

genericsensor→calibrateFromPoints()**YGenericSensor****genericsensor.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→describe()**YGenericSensor****genericsensor.describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur générique au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

String **describe()**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le capteur générique (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

genericsensor→get_advertisedValue()

YGenericSensor

genericsensor→advertisedValue()

genericsensor.get_advertisedValue()

Retourne la valeur courante du capteur générique (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du capteur générique (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

genericsensor→get_currentRawValue()**YGenericSensor****genericsensor→currentRawValue()****genericsensor.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration).

double **get_currentRawValue()**

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration)

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

genericsensor→**get_currentValue()**
genericsensor→**currentValue()**
genericsensor.get_currentValue()

YGenericSensor

Retourne la valeur mesurée actuelle.

`double get_currentValue()`

Retourne :

une valeur numérique représentant la valeur mesurée actuelle

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

genericsensor→get_errorMessage()**YGenericSensor****genericsensor→errorMessage()****genericsensor.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur générique.

genericsensor→get_errorType()

YGenericSensor

genericsensor→errorType()

genericsensor.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur générique.

```
int get_errorType()
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur générique.

genericsensor→**get_friendlyName()****YGenericSensor****genericsensor**→**friendlyName()****genericsensor.get_friendlyName()**

Retourne un identifiant global du capteur générique au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du capteur générique si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur générique (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur générique en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

genericsensor→**get_functionDescriptor()**

YGenericSensor

genericsensor→**functionDescriptor()**

genericsensor.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

genericsensor→**get_functionId()****YGenericSensor****genericsensor**→**functionId()****genericsensor.get_functionId()**

Retourne l'identifiant matériel du capteur générique, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur générique (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

genericsensor→**get_hardwareId()**
genericsensor→**hardwareId()**
genericsensor.get_hardwareId()

YGenericSensor

Retourne l'identifiant matériel unique du capteur générique au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur générique (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur générique (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

genericsensor→get_highestValue()**YGenericSensor****genericsensor→highestValue()****genericsensor.get_highestValue()**

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

double **get_highestValue()**

Retourne :

une valeur numérique représentant la valeur maximale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

genericsensor→get_logFrequency()

YGenericSensor

genericsensor→logFrequency()

genericsensor.get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

String get_logFrequency()

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

genericsensor→get_logicalName()**YGenericSensor****genericsensor→logicalName()****genericsensor.get_logicalName()**

Retourne le nom logique du capteur générique.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du capteur générique.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

genericsensor→**get_lowestValue()**

YGenericSensor

genericsensor→**lowestValue()**

genericsensor.get_lowestValue()

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

double **get_lowestValue()**

Retourne :

une valeur numérique représentant la valeur minimale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_LOWESTVALUE_INVALID.

genericsensor→get_module()
genericsensor→module()
genericsensor.get_module()

YGenericSensor

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

YModule **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :

une instance de YModule

genericsensor→**get_recordedData()**

YGenericSensor

genericsensor→**recordedData()**

genericsensor.get_recordedData()

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

YDataSet **get_recordedData**(long **startTime**, long **endTime**)

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

genericsensor→get_reportFrequency()**YGenericSensor****genericsensor→reportFrequency()****genericsensor.get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String **get_reportFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

genericsensor→**get_resolution()**

YGenericSensor

genericsensor→**resolution()**

genericsensor.get_resolution()

Retourne la résolution des valeurs mesurées.

`double get_resolution()`

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_RESOLUTION_INVALID.

genericsensor→get_signalBias()**YGenericSensor****genericsensor→signalBias()****genericsensor.get_signalBias()**

Retourne le biais du signal électrique pour la correction du point zéro.

double **get_signalBias()**

Un biais positif correspond à la correction d'un signal trop positif, tandis qu'un biais négatif correspond à la correction d'un signal trop négatif.

Retourne :

une valeur numérique représentant le biais du signal électrique pour la correction du point zéro

En cas d'erreur, déclenche une exception ou retourne Y_SIGNALBIAS_INVALID.

genericsensor→get_signalRange()

YGenericSensor

genericsensor→signalRange()

genericsensor.get_signalRange()

Retourne la plage de signal électrique utilisée par le capteur.

String **get_signalRange()**

Retourne :

une chaîne de caractères représentant la plage de signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne Y_SIGNALRANGE_INVALID.

genericsensor→get_signalUnit()**YGenericSensor****genericsensor→signalUnit()****genericsensor.get_signalUnit()**

Retourne l'unité du signal électrique utilisée par le capteur.

String **get_signalUnit()** ()

Retourne :

une chaîne de caractères représentant l'unité du signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne Y_SIGNALUNIT_INVALID.

genericsensor→get_signalValue()

YGenericSensor

genericsensor→signalValue()

genericsensor.get_signalValue()

Retourne la valeur mesurée du signal électrique utilisée par le capteur.

double get_signalValue()

Retourne :

une valeur numérique représentant la valeur mesurée du signal électrique utilisée par le capteur

En cas d'erreur, déclenche une exception ou retourne Y_SIGNALVALUE_INVALID.

genericsensor→**get_unit()****YGenericSensor****genericsensor**→**unit()**`genericsensor.get_unit()`

Retourne l'unité dans laquelle la mesure est exprimée.

String **get_unit()**

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la mesure est exprimée

En cas d'erreur, déclenche une exception ou retourne Y_UNIT_INVALID.

genericsensor→get_userdata()

YGenericSensor

genericsensor→userData()

genericsensor.getUserData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.

Object **get_userdata()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

genericsensor→get_valueRange()**YGenericSensor****genericsensor→valueRange()****genericsensor.get_valueRange()**

Retourne la plage de valeurs physiques mesurés par le capteur.

String **get_valueRange()**

Retourne :

une chaîne de caractères représentant la plage de valeurs physiques mesurés par le capteur

En cas d'erreur, déclenche une exception ou retourne Y_VALUERANGE_INVALID.

genericsensor→isOnline()

YGenericSensor

genericsensor.isOnline()

Vérifie si le module hébergeant le capteur générique est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache du capteur générique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur générique est joignable, `false` sinon

genericsensor→**load()**`genericsensor.load()`**YGenericSensor**

Met en cache les valeurs courantes du capteur générique, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→loadCalibrationPoints()**YGenericSensor****genericsensor.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**nextGenericSensor()****YGenericSensor****genericsensor.nextGenericSensor()**

Continue l'énumération des capteurs génériques commencée à l'aide de `yFirstGenericSensor()`.

YGenericSensor **nextGenericSensor()****Retourne :**

un pointeur sur un objet `YGenericSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

genericsensor→**registerTimedReportCallback()****YGenericSensor****genericsensor.registerTimedReportCallback(
)**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

genericsensor→registerValueCallback()**YGenericSensor****genericsensor.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

genericsensor→**set_highestValue()**

YGenericSensor

genericsensor→**setHighestValue()**

genericsensor.set_highestValue()

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**set_logFrequency()****YGenericSensor****genericsensor**→**setLogFrequency()****genericsensor.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→set_logicalName()

YGenericSensor

genericsensor→setLogicalName()

genericsensor.set_logicalName()

Modifie le nom logique du capteur générique.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur générique.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→set_lowestValue()**YGenericSensor****genericsensor→setLowestValue()****genericsensor.set_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→set_reportFrequency()

YGenericSensor

genericsensor→setReportFrequency()

genericsensor.set_reportFrequency()

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**set_resolution()****YGenericSensor****genericsensor**→**setResolution()****genericsensor.set_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→set_signalBias()

YGenericSensor

genericsensor→setSignalBias()

genericsensor.set_signalBias()

Modifie le biais du signal électrique pour la correction du point zéro.

```
int set_signalBias( double newval)
```

Si votre signal électrique est positif lorsqu'il devrait être nul, configurez un biais positif de la même valeur afin de corriger l'erreur.

Paramètres :

newval une valeur numérique représentant le biais du signal électrique pour la correction du point zéro

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→set_signalRange()**YGenericSensor****genericsensor→setSignalRange()****genericsensor.set_signalRange()**

Modifie la plage de signal électrique utilisée par le capteur.

```
int set_signalRange( String newval)
```

Paramètres :

newval une chaîne de caractères représentant la plage de signal électrique utilisée par le capteur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**set_unit()**

YGenericSensor

genericsensor→**setUnit()**

genericsensor.set_unit()

Change l'unité dans laquelle la valeur mesurée est exprimée.

```
int set_unit( String newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**set_userData()****YGenericSensor****genericsensor**→**setUserData()****genericsensor.set_userData()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

```
void set_userData( Object data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

genericsensor→**set_valueRange()**
genericsensor→**setValueRange()**
genericsensor.set_valueRange()

YGenericSensor

Modifie la plage de valeurs physiques mesurés par le capteur.

```
int set_valueRange( String newval)
```

Le changement de plage peut avoir pour effet de bord un changement automatique de la résolution affichée.

Paramètres :

newval une chaîne de caractères représentant la plage de valeurs physiques mesurés par le capteur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

genericsensor→**zeroAdjust()****YGenericSensor****genericsensor.zeroAdjust()**

Ajuste le biais du signal de sorte à ce que la valeur actuelle du signal soit interprétée comme zéro (tare).

int **zeroAdjust()**

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

3.19. Interface de la fonction Gyro

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_gyro.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YGyro = yoctolib.YGyro;
php	require_once('yocto_gyro.php');
c++	#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 estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

gyro→get_quaternionX()

Retourne la composante x du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

gyro→get_quaternionY()

Retourne la composante y du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

gyro→get_quaternionZ()

Retourne la composante z du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

gyro→get_recordedData(startTime, endTime)

Retourne un objet `DataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du `DataLogger`, pour l'intervalle de temps spécifié.

gyro→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

gyro→get_resolution()

Retourne la résolution des valeurs mesurées.

gyro→get_roll()

Retourne une estimation de l'inclinaison (angle de roulis), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

gyro→get_unit()

Retourne l'unité dans laquelle la vitesse angulaire est exprimée.

gyro→get_userData()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

gyro→get_xValue()

Retourne la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule.

gyro→get_yValue()

Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.

gyro→get_zValue()

Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.

gyro→isOnline()

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

gyro→isOnline_async(callback, context)

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

gyro→load(msValidity)

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

gyro→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

gyro→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

gyro→nextGyro()

Continue l'énumération des gyroscopes commencée à l'aide de `yFirstGyro()`.

gyro→registerAnglesCallback(callback)

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

gyro→registerQuaternionCallback(callback)

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

gyro→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

gyro→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

gyro→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

gyro→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

gyro→set_logicalName(newval)

Modifie le nom logique du gyroscope.

gyro→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

gyro→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

gyro→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

gyro→set_userData(data)

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

gyro→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YGyro.FindGyro()**YGyro****yFindGyro()****YGyro.FindGyro()**

Permet de retrouver un gyroscope d'après un identifiant donné.

YGyro **FindGyro**(String **func**)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le gyroscope soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YGyro.isOnline()` pour tester si le gyroscope est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le gyroscope sans ambiguïté

Retourne :

un objet de classe `YGyro` qui permet ensuite de contrôler le gyroscope.

YGyro.FirstGyro()

YGyro

yFirstGyro()**YGyro.FirstGyro()**

Commence l'énumération des gyroscopes accessibles par la librairie.

YGyro **FirstGyro()**

Utiliser la fonction `YGyro.nextGyro()` pour itérer sur les autres gyroscopes.

Retourne :

un pointeur sur un objet `YGyro`, correspondant au premier gyroscope accessible en ligne, ou `null` si il n'y a pas de gyroscopes disponibles.

gyro→calibrateFromPoints()**YGyro****gyro.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→describe()`gyro.describe()`**YGyro**

Retourne un court texte décrivant de manière non-ambigüe l'instance du gyroscope au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le gyroscope (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

gyro→**get_advertisedValue()**
gyro→**advertisedValue()**
gyro.get_advertisedValue()

YGyro

Retourne la valeur courante du gyroscope (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du gyroscope (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

gyro→get_currentRawValue()

YGyro

gyro→currentRawValue()

gyro.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés par seconde, sous forme de nombre à virgule.

double **get_currentRawValue()**

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés par seconde, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

gyro→**get_currentValue()****YGyro****gyro**→**currentValue()****gyro.get_currentValue()**

Retourne la valeur actuelle de la vitesse angulaire, en degrés par seconde, sous forme de nombre à virgule.

`double get_currentValue()`**Retourne :**

une valeur numérique représentant la valeur actuelle de la vitesse angulaire, en degrés par seconde, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

gyro→**get_errorMessage()**

YGyro

gyro→**errorMessage()****gyro.errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du gyroscope.

gyro→get_errorType()**YGyro****gyro→errorType()**`gyro.errorType()`

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du gyroscope.

`int get_errorType()`

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du gyroscope.

gyro→get_friendlyName()

YGyro

gyro→friendlyName()`gyro.get_friendlyName()`

Retourne un identifiant global du gyroscope au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du gyroscope si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du gyroscope (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le gyroscope en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

gyro→**get_functionDescriptor()**
gyro→**functionDescriptor()**
gyro.get_functionDescriptor()

YGyro

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

gyro→**get_functionId()**

YGyro

gyro→**functionId()****gyro.get_functionId()**

Retourne l'identifiant matériel du gyroscope, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le gyroscope (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

gyro→get_hardwareId()**YGyro****gyro→hardwareId()**`gyro.get_hardwareId()`

Retourne l'identifiant matériel unique du gyroscope au format `SERIAL.FUNCTIONID`.

`String` **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du gyroscope (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le gyroscope (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

gyro→**get_heading()**

YGyro

gyro→**heading()**`gyro.get_heading()`

Retourne une estimation du cap (angle de lacet), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

double **get_heading()**

L'axe de lacet peut être attribué à n'importe laquelle des direction physiques X, Y ou Z du module à l'aide des méthodes de la classe `YRefFrame`.

Retourne :

un nombre à virgule correspondant au cap, exprimé en degrés (entre 0 et 360).

gyro→**get_highestValue()****YGyro****gyro**→**highestValue()****gyro.get_highestValue()**

Retourne la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module.

double **get_highestValue()**

Retourne :

une valeur numérique représentant la valeur maximale observée pour la vitesse angulaire depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

gyro→**get_logFrequency()**

YGyro

gyro→**logFrequency()****gyro.get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

String **get_logFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

gyro→**get_logicalName()****YGyro****gyro**→**logicalName()****gyro.get_logicalName()**

Retourne le nom logique du gyroscope.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du gyroscope.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

gyro→**get_lowestValue()**

YGyro

gyro→**lowestValue()****gyro.get_lowestValue()**

Retourne la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module.

double **get_lowestValue()** ()

Retourne :

une valeur numérique représentant la valeur minimale observée pour la vitesse angulaire depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_LOWESTVALUE_INVALID.

gyro→**get_module()****YGyro****gyro**→**module()**`gyro.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

gyro→**get_pitch()**

YGyro

gyro→**pitch()****gyro.get_pitch()**

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.

double **get_pitch()**

L'axe de tangage peut être attribué à n'importe laquelle des direction physiques X, Y ou Z du module à l'aide des méthodes de la classe `YRefFrame`.

Retourne :

un nombre à virgule correspondant à l'assiette, exprimée en degrés (entre -90 et +90).

gyro→**get_quaternionW()****YGyro****gyro**→**quaternionW()****gyro.get_quaternionW()**

Retourne la composante w (composante réelle) du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

double **get_quaternionW()****Retourne :**

un nombre à virgule correspondant à la composante w du quaternion.

gyro→**get_quaternionX()**

YGyro

gyro→**quaternionX()**`gyro.get_quaternionX()`

Retourne la composante x du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

`double get_quaternionX()`

La composante x est essentiellement corrélée aux rotations sur l'axe de roulis.

Retourne :

un nombre à virgule correspondant à la composante x du quaternion.

gyro→**get_quaternionY()****YGyro****gyro**→**quaternionY()****gyro.get_quaternionY()**

Retourne la composante y du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

```
double get_quaternionY( )
```

La composante y est essentiellement corrélée aux rotations sur l'axe de tangage.

Retourne :

un nombre à virgule correspondant à la composante y du quaternion.

gyro→**get_quaternionZ()**

YGyro

gyro→**quaternionZ()****gyro.get_quaternionZ()**

Retourne la composante *z* du quaternion décrivant l'orientation estimée du module, basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

double **get_quaternionZ()**

La composante *z* est essentiellement corrélée aux rotations sur l'axe de lacet.

Retourne :

un nombre à virgule correspondant à la composante *z* du quaternion.

gyro→get_recordedData()**YGyro****gyro→recordedData()**`gyro.get_recordedData()`

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

`YDataSet` **get_recordedData**(long **startTime**, long **endTime**)

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intercalé de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

gyro→get_reportFrequency()
gyro→reportFrequency()
gyro.get_reportFrequency()

YGyro

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String get_reportFrequency()

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

gyro→**get_resolution()****YGyro****gyro**→**resolution()****gyro.get_resolution()**

Retourne la résolution des valeurs mesurées.

double **get_resolution()** ()

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_RESOLUTION_INVALID.

gyro→**get_roll()**

YGyro

gyro→**roll()**`gyro.get_roll()`

Retourne une estimation de l'inclinaison (angle de roulis), basée sur l'intégration de mesures gyroscopiques combinée à des mesures statiques d'accélération et de champ magnétique.

double **get_roll()** ()

L'axe de roulis peut être attribué à n'importe laquelle des direction physiques X, Y ou Z du module à l'aide des méthodes de la classe `YRefFrame`.

Retourne :

un nombre à virgule correspondant à l'inclinaison, exprimée en degrés (entre -180 et +180).

gyro→**get_unit()****YGyro****gyro**→**unit()****gyro.get_unit()**

Retourne l'unité dans laquelle la vitesse angulaire est exprimée.

String **get_unit()**

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la vitesse angulaire est exprimée

En cas d'erreur, déclenche une exception ou retourne Y_UNIT_INVALID.

gyro→**get_userdata()**

YGyro

gyro→**userData()****gyro.userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

gyro→**get_xValue()****YGyro****gyro**→**xValue()****gyro.get_xValue()**

Retourne la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule.

double **get_xValue()**

Retourne :

une valeur numérique représentant la vitesse angulaire autour de l'axe X du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_XVALUE_INVALID.

gyro→**get_yValue()**

YGyro

gyro→**yValue()****gyro.get_yValue()**

Retourne la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule.

double **get_yValue()**

Retourne :

une valeur numérique représentant la vitesse angulaire autour de l'axe Y du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_YVALUE_INVALID.

gyro→**get_zValue()****YGyro****gyro**→**zValue()****gyro.get_zValue()**

Retourne la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule.

double **get_zValue()**

Retourne :

une valeur numérique représentant la vitesse angulaire autour de l'axe Z du module, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_ZVALUE_INVALID.

gyro→**isOnline()****gyro.isOnline()**

YGyro

Vérifie si le module hébergeant le gyroscope est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache du gyroscope sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

true si le gyroscope est joignable, false sinon

gyro→load()`gyro.load()`**YGyro**

Met en cache les valeurs courantes du gyroscope, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→loadCalibrationPoints()**YGyro****gyro.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→**nextGyro()****gyro.nextGyro()****YGyro**

Continue l'énumération des gyroscopes commencée à l'aide de `yFirstGyro()`.

`YGyro` **nextGyro()**

Retourne :

un pointeur sur un objet `YGyro` accessible en ligne, ou `null` lorsque l'énumération est terminée.

gyro→registerAnglesCallback()**YGyro****gyro.registerAnglesCallback()**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

```
int registerAnglesCallback( YAnglesCallback callback)
```

La fréquence d'appel est typiquement de 95Hz durant un mouvement. Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand le callback peut se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que le callback ne soit pas appelé trop tard. Pour désactiver le callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter quatre arguments: l'objet YGyro du module qui a tourné, et les valeurs des trois angles roll, pitch et heading en degrés (nombres à virgules).

gyro→registerQuaternionCallback()**YGyro****gyro.registerQuaternionCallback()**

Enregistre une fonction de callback qui sera appelée à chaque changement de l'estimation de l'orientation du module.

```
int registerQuaternionCallback( YQuatCallback callback)
```

La fréquence d'appel est typiquement de 95Hz durant un mouvement. Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand le callback peut se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que le callback ne soit pas appelés trop tard. Pour désactiver le callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter cinq arguments: l'objet YGyro du module qui a tourné, et les valeurs des quatre composantes w, x, y et z du quaternion (nombres à virgules).

gyro→registerTimedReportCallback()**YGyro****gyro.registerTimedReportCallback()**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

gyro→registerValueCallback()**YGyro****gyro.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

gyro→**set_highestValue()**

YGyro

gyro→**setHighestValue()**

gyro.set_highestValue()

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→set_logFrequency()
gyro→setLogFrequency()
gyro.set_logFrequency()

YGyro

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→**set_logicalName()**

YGyro

gyro→**setLogicalName()****gyro.set_logicalName()**

Modifie le nom logique du gyroscope.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du gyroscope.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→**set_lowestValue()****YGyro****gyro**→**setLowestValue()****gyro.set_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→set_reportFrequency()**YGyro****gyro→setReportFrequency()****gyro.set_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→**set_resolution()****YGyro****gyro**→**setResolution()****gyro.set_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

gyro→**set_userdata()**

YGyro

gyro→**setUserData()****gyro.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

`void set_userdata(Object data)`

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

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');
c++	#include "yocto_hubport.h"
m	#import "yocto_hubport.h"
pas	uses yocto_hubport;
vb	yocto_hubport.vb
cs	yocto_hubport.cs
java	import com.yoctopuce.YoctoAPI.YHubPort;
py	from yocto_hubport import *

Fonction globales

yFindHubPort(func)

Permet de retrouver un port de Yocto-hub d'après un identifiant donné.

yFirstHubPort()

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

Méthodes des objets YHubPort

hubport→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du port de Yocto-hub au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

hubport→get_advertisedValue()

Retourne la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

hubport→get_baudRate()

Retourne la vitesse de transfert utilisée par le port de Yocto-hub, en kbps.

hubport→get_enabled()

Retourne vrai si le port du Yocto-hub est alimenté, faux sinon.

hubport→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

hubport→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

hubport→get_friendlyName()

Retourne un identifiant global du port de Yocto-hub au format `NOM_MODULE . NOM_FONCTION`.

hubport→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

hubport→get_functionId()

Retourne l'identifiant matériel du port de Yocto-hub, sans référence au module.

hubport→get_hardwareId()

Retourne l'identifiant matériel unique du port de Yocto-hub au format `SERIAL . FUNCTIONID`.

hubport→get_logicalName()

	Retourne le nom logique du port de Yocto-hub.
hubport→get_module()	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
hubport→get_module_async(callback, context)	Retourne l'objet <code>YModule</code> 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 <code>userData</code> , précédemment stocké à l'aide de la méthode <code>set_userData</code> .
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 <code>yFirstHubPort()</code> .
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 <code>userData</code> de la fonction, afin de le retrouver plus tard à l'aide de la méthode <code>get_userData</code> .
hubport→wait_async(callback, context)	Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YHubPort.FindHubPort()**YHubPort****yFindHubPort()**`YHubPort.FindHubPort()`

Permet de retrouver un port de Yocto-hub d'après un identifiant donné.

`YHubPort` **FindHubPort**(`String func`)

L'identifiant peut être spécifié sous plusieurs formes:

- `NomLogiqueFonction`
- `NoSerieModule.IdentifiantFonction`
- `NoSerieModule.NomLogiqueFonction`
- `NomLogiqueModule.IdentifiantMatériel`
- `NomLogiqueModule.NomLogiqueFonction`

Cette fonction n'exige pas que le port de Yocto-hub soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YHubPort.isOnline()` pour tester si le port de Yocto-hub est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le port de Yocto-hub sans ambiguïté

Retourne :

un objet de classe `YHubPort` qui permet ensuite de contrôler le port de Yocto-hub.

YHubPort.FirstHubPort()

YHubPort

yFirstHubPort()`YHubPort.FirstHubPort()`

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

`YHubPort` **FirstHubPort()**

Utiliser la fonction `YHubPort.nextHubPort()` pour itérer sur les autres port de Yocto-hub.

Retourne :

un pointeur sur un objet `YHubPort`, correspondant au premier port de Yocto-hub accessible en ligne, ou `null` si il n'y a pas de port de Yocto-hub disponibles.

hubport→**describe()****hubport.describe()****YHubPort**

Retourne un court texte décrivant de manière non-ambigüe l'instance du port de Yocto-hub au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String **describe()**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le port de Yocto-hub (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

hubport→**get_advertisedValue()**

YHubPort

hubport→**advertisedValue()**

hubport.get_advertisedValue()

Retourne la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du port de Yocto-hub (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

hubport→**get_baudRate()****YHubPort****hubport**→**baudRate()****hubport.get_baudRate()**

Retourne la vitesse de transfert utilisée par le port de Yocto-hub, en kbps.

int **get_baudRate()**

La valeur par défaut est 1000 kbps, une valeur inférieure révèle des problèmes de communication.

Retourne :

un entier représentant la vitesse de transfert utilisée par le port de Yocto-hub, en kbps

En cas d'erreur, déclenche une exception ou retourne Y_BAUDRATE_INVALID.

hubport→**get_enabled()**

YHubPort

hubport→**enabled()**`hubport.get_enabled()`

Retourne vrai si le port du Yocto-hub est alimenté, faux sinon.

```
int get_enabled( )
```

Retourne :

soit `Y_ENABLED_FALSE`, soit `Y_ENABLED_TRUE`, selon vrai si le port du Yocto-hub est alimenté, faux sinon

En cas d'erreur, déclenche une exception ou retourne `Y_ENABLED_INVALID`.

hubport→**get_errorMessage()****YHubPort****hubport**→**errorMessage()****hubport.errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

String **errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du port de Yocto-hub.

hubport→**get_errorType()**

YHubPort

hubport→**errorType()****hubport.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port de Yocto-hub.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du port de Yocto-hub.

hubport→**get_friendlyName()****YHubPort****hubport**→**friendlyName()****hubport.get_friendlyName()**

Retourne un identifiant global du port de Yocto-hub au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du port de Yocto-hub si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du port de Yocto-hub (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le port de Yocto-hub en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

hubport→**get_functionDescriptor()**

YHubPort

hubport→**functionDescriptor()**

hubport.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

hubport→**get_functionId()****YHubPort****hubport**→**functionId()**`hubport.get_functionId()`

Retourne l'identifiant matériel du port de Yocto-hub, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le port de Yocto-hub (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

hubport→**get_hardwareId()**

YHubPort

hubport→**hardwareId()**`hubport.get_hardwareId()`

Retourne l'identifiant matériel unique du port de Yocto-hub au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du port de Yocto-hub (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le port de Yocto-hub (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

hubport→**get_logicalName()**
hubport→**logicalName()**
hubport.get_logicalName()

YHubPort

Retourne le nom logique du port de Yocto-hub.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du port de Yocto-hub.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

hubport→**get_module()**

YHubPort

hubport→**module()**`hubport.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

hubport→**get_portState()****YHubPort****hubport**→**portState()**`hubport.get_portState()`

Retourne l'état actuel du port de Yocto-hub.

`int` **get_portState()** ()

Retourne :

une valeur parmi `Y_PORTSTATE_OFF`, `Y_PORTSTATE_OVRLD`, `Y_PORTSTATE_ON`, `Y_PORTSTATE_RUN` et `Y_PORTSTATE_PROG` représentant l'état actuel du port de Yocto-hub

En cas d'erreur, déclenche une exception ou retourne `Y_PORTSTATE_INVALID`.

hubport→**get_userData()**

YHubPort

hubport→**userData()****hubport.userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

hubport→**isOnline()**`hubport.isOnline()`**YHubPort**

Vérifie si le module hébergeant le port de Yocto-hub est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du port de Yocto-hub sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le port de Yocto-hub est joignable, `false` sinon

hubport→**load()****hubport.load()****YHubPort**

Met en cache les valeurs courantes du port de Yocto-hub, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

hubport→**nextHubPort()**`hubport.nextHubPort()`**YHubPort**

Continue l'énumération des port de Yocto-hub commencée à l'aide de `yFirstHubPort()`.

`YHubPort` **nextHubPort()**

Retourne :

un pointeur sur un objet `YHubPort` accessible en ligne, ou `null` lorsque l'énumération est terminée.

hubport→registerValueCallback()

YHubPort

hubport.registerValueCallback()

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

hubport→**set_enabled()****YHubPort****hubport**→**setEnabled()**`hubport.setEnabled()`

Modifie le mode d'activation du port du Yocto-hub.

```
int set_enabled( int newval)
```

Si le port est actif, il sera alimenté. Sinon, l'alimentation du module est coupée.

Paramètres :

newval soit Y_ENABLED_FALSE, soit Y_ENABLED_TRUE, selon le mode d'activation du port du Yocto-hub

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

hubport→**set_logicalName()**

YHubPort

hubport→**setLogicalName()**

hubport.set_logicalName()

Modifie le nom logique du port de Yocto-hub.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du port de Yocto-hub.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

hubport→**set_userdata()****YHubPort****hubport**→**setUserData()****hubport.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

3.21. Interface de la fonction Humidity

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_humidity.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YHumidity = yoctolib.YHumidity;
php	require_once('yocto_humidity.php');
c++	#include "yocto_humidity.h"
m	#import "yocto_humidity.h"
pas	uses yocto_humidity;
vb	yocto_humidity.vb
cs	yocto_humidity.cs
java	import com.yoctopuce.YoctoAPI.YHumidity;
py	from yocto_humidity import *

Fonction globales

yFindHumidity(func)

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

yFirstHumidity()

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

Méthodes des objets YHumidity

humidity→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

humidity→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur d'humidité au format TYPE (NAME) = SERIAL . FUNCTIONID.

humidity→get_advertisedValue()

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

humidity→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), 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)

3. Reference

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

humidity→**set_logicalName(newval)**

Modifie le nom logique du capteur d'humidité.

humidity→**set_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

humidity→**set_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

humidity→**set_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

humidity→**set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

humidity→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YHumidity.FindHumidity()**YHumidity****yFindHumidity()YHumidity.FindHumidity()**

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

YHumidity FindHumidity(String func)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur d'humidité soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YHumidity.isOnline()` pour tester si le capteur d'humidité est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur d'humidité sans ambiguïté

Retourne :

un objet de classe `YHumidity` qui permet ensuite de contrôler le capteur d'humidité.

YHumidity.FirstHumidity()

YHumidity

yFirstHumidity()`YHumidity.FirstHumidity()`

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

`YHumidity` **FirstHumidity()**

Utiliser la fonction `YHumidity.nextHumidity()` pour itérer sur les autres capteurs d'humidité.

Retourne :

un pointeur sur un objet `YHumidity`, correspondant au premier capteur d'humidité accessible en ligne, ou `null` si il n'y a pas de capteurs d'humidité disponibles.

humidity→calibrateFromPoints()**YHumidity****humidity.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                        ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→describe()`humidity.describe()`**YHumidity**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur d'humidité au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le capteur d'humidité (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

humidity→get_advertisedValue()**YHumidity****humidity→advertisedValue()****humidity.get_advertisedValue()**

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du capteur d'humidité (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

humidity→get_currentRawValue()

YHumidity

humidity→currentRawValue()

humidity.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en %RH, sous forme de nombre à virgule.

`double get_currentRawValue()`

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en %RH, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

humidity→get_currentValue()**YHumidity****humidity→currentValue()****humidity.get_currentValue()**

Retourne la valeur actuelle de l'humidité, en %RH, sous forme de nombre à virgule.

```
double get_currentValue()
```

Retourne :

une valeur numérique représentant la valeur actuelle de l'humidité, en %RH, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

humidity→get_errorMessage()

YHumidity

humidity→errorMessage()

humidity.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

String get_errorMessage()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur d'humidité.

humidity→get_errorType()**YHumidity****humidity→errorType()humidity.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur d'humidité.

int **get_errorType()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur d'humidité.

humidity→get_friendlyName()

YHumidity

humidity→friendlyName()

humidity.get_friendlyName()

Retourne un identifiant global du capteur d'humidité au format `NOM_MODULE.NOM_FONCTION`.

String get_friendlyName()

Le chaîne retournée utilise soit les noms logiques du module et du capteur d'humidité si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur d'humidité (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur d'humidité en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

humidity→get_functionDescriptor()**YHumidity****humidity→functionDescriptor()****humidity.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

humidity→**get_functionId()**

YHumidity

humidity→**functionId()**

humidity.get_functionId()

Retourne l'identifiant matériel du capteur d'humidité, sans référence au module.

String **get_functionId()** ()

Par exemple relay1.

Retourne :

une chaîne de caractères identifiant le capteur d'humidité (ex: relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FUNCTIONID_INVALID.

humidity→get_hardwareId()**YHumidity****humidity→hardwareId()****humidity.get_hardwareId()**

Retourne l'identifiant matériel unique du capteur d'humidité au format `SERIAL.FUNCTIONID`.

String `get_hardwareId()`

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur d'humidité (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur d'humidité (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

humidity→get_highestValue()

YHumidity

humidity→highestValue()

humidity.get_highestValue()

Retourne la valeur maximale observée pour l'humidité depuis le démarrage du module.

`double get_highestValue()`

Retourne :

une valeur numérique représentant la valeur maximale observée pour l'humidité depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

humidity→get_logFrequency()**YHumidity****humidity→logFrequency()****humidity.get_logFrequency()**

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.

String **get_logFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

humidity→get_logicalName()

YHumidity

humidity→logicalName()

humidity.get_logicalName()

Retourne le nom logique du capteur d'humidité.

String get_logicalName()

Retourne :

une chaîne de caractères représentant le nom logique du capteur d'humidité.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

humidity→get_lowestValue()**YHumidity****humidity→lowestValue()****humidity.get_lowestValue()**

Retourne la valeur minimale observée pour l'humidité depuis le démarrage du module.

```
double get_lowestValue()
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour l'humidité depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_LOWESTVALUE_INVALID.

humidity→**get_module()**

YHumidity

humidity→**module()**`humidity.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

humidity→get_recordedData()**YHumidity****humidity→recordedData()****humidity.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

YDataSet **get_recordedData**(long **startTime**, long **endTime**)

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

humidity→get_reportFrequency()

YHumidity

humidity→reportFrequency()

humidity.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String get_reportFrequency()

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

humidity→get_resolution()**YHumidity****humidity→resolution()****humidity.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
double get_resolution( )
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_RESOLUTION_INVALID.

humidity→**get_unit()**

YHumidity

humidity→**unit()**`humidity.get_unit()`

Retourne l'unité dans laquelle l'humidité est exprimée.

String **get_unit()** ()

Retourne :

une chaîne de caractères représentant l'unité dans laquelle l'humidité est exprimée

En cas d'erreur, déclenche une exception ou retourne Y_UNIT_INVALID.

humidity→get_userdata()**YHumidity****humidity→userData()****humidity.get_userdata()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.

Object **get_userdata()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

humidity→isOnline()humidity.isOnline()

YHumidity

Vérifie si le module hébergeant le capteur d'humidité est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du capteur d'humidité sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

true si le capteur d'humidité est joignable, false sinon

humidity→load()`humidity.load()`**YHumidity**

Met en cache les valeurs courantes du capteur d'humidité, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→loadCalibrationPoints()**YHumidity****humidity.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→**nextHumidity()****YHumidity****humidity.nextHumidity()**

Continue l'énumération des capteurs d'humidité commencée à l'aide de `yFirstHumidity()`.

`YHumidity` **nextHumidity()**

Retourne :

un pointeur sur un objet `YHumidity` accessible en ligne, ou `null` lorsque l'énumération est terminée.

humidity→registerTimedReportCallback()**YHumidity****humidity.registerTimedReportCallback()**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

humidity→registerValueCallback()**YHumidity****humidity.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

humidity→set_highestValue()

YHumidity

humidity→setHighestValue()

humidity.set_highestValue()

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→set_logFrequency()**YHumidity****humidity→setLogFrequency()****humidity.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→set_logicalName()

YHumidity

humidity→setLogicalName()

humidity.set_logicalName()

Modifie le nom logique du capteur d'humidité.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur d'humidité.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→set_lowestValue()**YHumidity****humidity→setLowestValue()****humidity.set_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→set_reportFrequency()
humidity→setReportFrequency()
humidity.set_reportFrequency()

YHumidity

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→set_resolution()**YHumidity****humidity→setResolution()****humidity.set_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

humidity→set_userdata()

YHumidity

humidity→setUserData()

humidity.set_userdata()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

void **set_userdata**(Object **data**)

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

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');
c++	#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.

YLed.FindLed()**YLed****yFindLed()****YLed.FindLed()**

Permet de retrouver une led d'après un identifiant donné.

YLed FindLed(String func)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la led soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YLed.isOnline()` pour tester si la led est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence la led sans ambiguïté

Retourne :

un objet de classe `YLed` qui permet ensuite de contrôler la led.

YLed.FirstLed()

YLed

yFirstLed()**YLed.FirstLed()**

Commence l'énumération des leds accessibles par la librairie.

YLed FirstLed()

Utiliser la fonction `YLed.nextLed()` pour itérer sur les autres leds.

Retourne :

un pointeur sur un objet `YLed`, correspondant à la première led accessible en ligne, ou `null` si il n'y a pas de leds disponibles.

led→describe()`led.describe()`**YLed**

Retourne un court texte décrivant de manière non-ambigüe l'instance de la led au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String `describe()`

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant la led (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

led→**get_advertisedValue()**

YLed

led→**advertisedValue()**

led.get_advertisedValue()

Retourne la valeur courante de la led (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante de la led (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

led→**get_blinking()****YLed****led**→**blinking()****led.get_blinking()**

Retourne le mode de signalisation de la led.

int **get_blinking()** ()

Retourne :

une valeur parmi Y_BLINKING_STILL, Y_BLINKING_RELAX, Y_BLINKING_AWARE, Y_BLINKING_RUN, Y_BLINKING_CALL et Y_BLINKING_PANIC représentant le mode de signalisation de la led

En cas d'erreur, déclenche une exception ou retourne Y_BLINKING_INVALID.

led→**get_errorMessage()**

YLed

led→**errorMessage()****led.errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la led.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la led.

led→**get_errorType()****YLed****led**→**errorType()****led.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la led.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la led.

led→**get_friendlyName()**

YLed

led→**friendlyName()****led.get_friendlyName()**

Retourne un identifiant global de la led au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et de la led si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de la led (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant la led en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

led→**get_functionDescriptor()****YLed****led**→**functionDescriptor()****led.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

led→**get_functionId()**

YLed

led→**functionId()****led.get_functionId()**

Retourne l'identifiant matériel de la led, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant la led (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

led→**get_hardwareId()****YLed****led**→**hardwareId()****led.get_hardwareId()**

Retourne l'identifiant matériel unique de la led au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la led (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant la led (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

led→**get_logicalName()**

YLed

led→**logicalName()**`led.get_logicalName()`

Retourne le nom logique de la led.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique de la led.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

led→**get_luminosity()****YLed****led**→**luminosity()****led.get_luminosity()**

Retourne l'intensité de la led en pour cent.

int **get_luminosity()** ()

Retourne :

un entier représentant l'intensité de la led en pour cent

En cas d'erreur, déclenche une exception ou retourne Y_LUMINOSITY_INVALID.

led→**get_module()**

YLed

led→**module()**`led.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

led→**get_power()****YLed****led**→**power()**`led.get_power()`

Retourne l'état courant de la led.

`int get_power()`

Retourne :

soit `Y_POWER_OFF`, soit `Y_POWER_ON`, selon l'état courant de la led

En cas d'erreur, déclenche une exception ou retourne `Y_POWER_INVALID`.

led→**get_userdata()**

YLed

led→**userData()****led.get_userdata()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.

Object **get_userdata()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

led→**isOnline()****led.isOnline()****YLed**

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache de la led sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si la led est joignable, `false` sinon

led→**load()****led.load()****YLed**

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

led→**nextLed()****led.nextLed()****YLed**

Continue l'énumération des leds commencée à l'aide de `yFirstLed()`.

YLed **nextLed()**

Retourne :

un pointeur sur un objet YLed accessible en ligne, ou `null` lorsque l'énumération est terminée.

led→registerValueCallback()**YLed****led.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

led→**set_blinking()****YLed****led**→**setBlinking()****led.set_blinking()**

Modifie le mode de signalisation de la led.

```
int set_blinking( int newval)
```

Paramètres :

newval une valeur parmi Y_BLINKING_STILL, Y_BLINKING_RELAX, Y_BLINKING_AWARE, Y_BLINKING_RUN, Y_BLINKING_CALL et Y_BLINKING_PANIC représentant le mode de signalisation de la led

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

led→**set_logicalName()****YLed****led**→**setLogicalName()****led.set_logicalName()**

Modifie le nom logique de la led.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de la led.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

led→**set_luminosity()****YLed****led**→**setLuminosity()****led.set_luminosity()**

Modifie l'intensité lumineuse de la led (en pour cent).

```
int set_luminosity( int newval)
```

Paramètres :

newval un entier représentant l'intensité lumineuse de la led (en pour cent)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

led→**set_power()**

YLed

led→**setPower()**`led.set_power()`

Modifie l'état courant de la led.

```
int set_power( int newval)
```

Paramètres :

newval soit Y_POWER_OFF, soit Y_POWER_ON, selon l'état courant de la led

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

led→**set_userdata()****YLed****led**→**setUserData()****led.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

3.23. Interface de la fonction LightSensor

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_lightsensor.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YLightSensor = yoctolib.YLightSensor;
php	require_once('yocto_lightsensor.php');
c++	#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.

3. Reference

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 de mesure de lumière effectuée par le capteur.

lightsensor→**set_reportFrequency**(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

lightsensor→**set_resolution**(newval)

Modifie la résolution des valeurs physique mesurées.

lightsensor→**set_userdata**(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

lightsensor→**wait_async**(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YLightSensor.FindLightSensor()**YLightSensor****yFindLightSensor()****YLightSensor.FindLightSensor()**

Permet de retrouver un capteur de lumière d'après un identifiant donné.

```
YLightSensor FindLightSensor( String func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de lumière soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YLightSensor.isOnline()` pour tester si le capteur de lumière est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de lumière sans ambiguïté

Retourne :

un objet de classe `YLightSensor` qui permet ensuite de contrôler le capteur de lumière.

YLightSensor.FirstLightSensor()

YLightSensor

yFirstLightSensor()

YLightSensor.FirstLightSensor()

Commence l'énumération des capteurs de lumière accessibles par la librairie.

YLightSensor **FirstLightSensor()**

Utiliser la fonction `YLightSensor.nextLightSensor()` pour itérer sur les autres capteurs de lumière.

Retourne :

un pointeur sur un objet `YLightSensor`, correspondant au premier capteur de lumière accessible en ligne, ou `null` si il n'y a pas de capteurs de lumière disponibles.

lightsensor→**calibrate()****lightsensor.calibrate()****YLightSensor**

Modifie le paramètre de calibration spécifique du senseur de sorte à ce que la valeur actuelle corresponde à une consigne donnée (correction linéaire).

```
int calibrate( double calibratedVal)
```

Paramètres :

calibratedVal la consigne de valeur désirée.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→calibrateFromPoints()**YLightSensor****lightsensor.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→describe()lightsensor.describe()**YLightSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de lumière au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String **describe()**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le capteur de lumière (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

lightsensor→get_advertisedValue()

YLightSensor

lightsensor→advertisedValue()

lightsensor.get_advertisedValue()

Retourne la valeur courante du capteur de lumière (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de lumière (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

lightsensor→get_currentRawValue()**YLightSensor****lightsensor→currentRawValue()****lightsensor.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en lux, sous forme de nombre à virgule.

double **get_currentRawValue()**

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en lux, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

lightsensor→get_currentValue()

YLightSensor

lightsensor→currentValue()

lightsensor.get_currentValue()

Retourne la valeur actuelle de la lumière ambiante, en lux, sous forme de nombre à virgule.

double **get_currentValue()**

Retourne :

une valeur numérique représentant la valeur actuelle de la lumière ambiante, en lux, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

lightsensor→get_errorMessage()**YLightSensor****lightsensor→errorMessage()****lightsensor.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de lumière.

lightsensor→get_errorType()

YLightSensor

lightsensor→errorType()

lightsensor.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de lumière.

```
int get_errorType()
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de lumière.

lightsensor→get_friendlyName()**YLightSensor****lightsensor→friendlyName()****lightsensor.get_friendlyName()**

Retourne un identifiant global du capteur de lumière au format `NOM_MODULE.NOM_FONCTION`.

String `get_friendlyName()`

Le chaîne retournée utilise soit les noms logiques du module et du capteur de lumière si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de lumière (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de lumière en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

lightsensor→get_functionDescriptor()

YLightSensor

lightsensor→functionDescriptor()

lightsensor.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

lightsensor→**get_functionId()****YLightSensor****lightsensor**→**functionId()****lightsensor.get_functionId()**

Retourne l'identifiant matériel du capteur de lumière, sans référence au module.

String **get_functionId()** ()

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de lumière (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

lightsensor→get_hardwareId()

YLightSensor

lightsensor→hardwareId()

lightsensor.get_hardwareId()

Retourne l'identifiant matériel unique du capteur de lumière au format `SERIAL.FUNCTIONID`.

String get_hardwareId()

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de lumière (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur de lumière (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

lightsensor→get_highestValue()**YLightSensor****lightsensor→highestValue()****lightsensor.get_highestValue()**

Retourne la valeur maximale observée pour la lumière ambiante depuis le démarrage du module.

double **get_highestValue()**

Retourne :

une valeur numérique représentant la valeur maximale observée pour la lumière ambiante depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

lightsensor→get_logFrequency()

YLightSensor

lightsensor→logFrequency()

lightsensor.get_logFrequency()

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.

String **get_logFrequency()** ()

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

lightsensor→**get_logicalName()****YLightSensor****lightsensor**→**logicalName()****lightsensor.get_logicalName()**

Retourne le nom logique du capteur de lumière.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du capteur de lumière.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

lightsensor→get_lowestValue()

YLightSensor

lightsensor→lowestValue()

lightsensor.get_lowestValue()

Retourne la valeur minimale observée pour la lumière ambiante depuis le démarrage du module.

double **get_lowestValue()**

Retourne :

une valeur numérique représentant la valeur minimale observée pour la lumière ambiante depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_LOWESTVALUE_INVALID.

lightsensor→**get_measureType()****YLightSensor****lightsensor**→**measureType()****lightsensor.get_measureType()**

Retourne le type de mesure de lumière utilisé par le module.

int **get_measureType()**

Retourne :

une valeur parmi Y_MEASURETYPE_HUMAN_EYE, Y_MEASURETYPE_WIDE_SPECTRUM, Y_MEASURETYPE_INFRARED, Y_MEASURETYPE_HIGH_RATE et Y_MEASURETYPE_HIGH_ENERGY représentant le type de mesure de lumière utilisé par le module

En cas d'erreur, déclenche une exception ou retourne Y_MEASURETYPE_INVALID.

lightsensor→get_module()

YLightSensor

lightsensor→module()`lightsensor.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` [get_module\(\)](#)

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

lightsensor→**get_recordedData()****YLightSensor****lightsensor**→**recordedData()****lightsensor.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

YDataSet **get_recordedData**(long **startTime**, long **endTime**)

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

lightsensor→get_reportFrequency()

YLightSensor

lightsensor→reportFrequency()

lightsensor.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String get_reportFrequency()

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

lightsensor→**get_resolution()****YLightSensor****lightsensor**→**resolution()****lightsensor.get_resolution()**

Retourne la résolution des valeurs mesurées.

double **get_resolution()**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_RESOLUTION_INVALID.

lightsensor→**get_unit()**

YLightSensor

lightsensor→**unit()**`lightsensor.get_unit()`

Retourne l'unité dans laquelle la lumière ambiante est exprimée.

String **get_unit()**

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la lumière ambiante est exprimée

En cas d'erreur, déclenche une exception ou retourne Y_UNIT_INVALID.

lightsensor→get_userdata()
lightsensor→userData()
lightsensor.getUserData()

YLightSensor

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.

Object **get_userdata()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

lightsensor→**isOnline()**`lightsensor.isOnline()`

YLightSensor

Vérifie si le module hébergeant le capteur de lumière est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache du capteur de lumière sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur de lumière est joignable, `false` sinon

lightsensor→load()`lightsensor.load()`**YLightSensor**

Met en cache les valeurs courantes du capteur de lumière, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→loadCalibrationPoints()**YLightSensor****lightsensor.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→**nextLightSensor()****YLightSensor****lightsensor.nextLightSensor()**

Continue l'énumération des capteurs de lumière commencée à l'aide de `yFirstLightSensor()`.

`YLightSensor` **nextLightSensor()**

Retourne :

un pointeur sur un objet `YLightSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

lightsensor→registerTimedReportCallback()**YLightSensor****lightsensor.registerTimedReportCallback()**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

lightsensor→registerValueCallback()**YLightSensor****lightsensor.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

lightsensor→set_highestValue()

YLightSensor

lightsensor→setHighestValue()

lightsensor.set_highestValue()

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_logFrequency()**YLightSensor****lightsensor→setLogFrequency()****lightsensor.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

int **set_logFrequency**(String **newval**)

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_logicalName()**YLightSensor****lightsensor→setLogicalName()****lightsensor.set_logicalName()**

Modifie le nom logique du capteur de lumière.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de lumière.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→**set_lowestValue()****YLightSensor****lightsensor**→**setLowestValue()****lightsensor.set_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_measureType()**YLightSensor****lightsensor→setMeasureType()****lightsensor.set_measureType()**

Change le type de mesure de lumière effectuée par le capteur.

```
int set_measureType( int newval)
```

La mesure peut soit approximer la réponse de l'oeil humain, soit donner une valeur ciblant un spectre particulier, en fonction des possibilités offertes par le récepteur de lumière. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur parmi `Y_MEASURETYPE_HUMAN_EYE`,
`Y_MEASURETYPE_WIDE_SPECTRUM`, `Y_MEASURETYPE_INFRARED`,
`Y_MEASURETYPE_HIGH_RATE` et `Y_MEASURETYPE_HIGH_ENERGY`

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_reportFrequency()**YLightSensor****lightsensor→setReportFrequency()****lightsensor.set_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_resolution()

YLightSensor

lightsensor→setResolution()

lightsensor.set_resolution()

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

lightsensor→set_userdata()**YLightSensor****lightsensor→setUserData()****lightsensor.set_userdata()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

```
void set_userdata( Object data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

3.24. Interface de la fonction Magnetometer

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_magnetometer.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YMagnetometer = yoctolib.YMagnetometer;
php	require_once('yocto_magnetometer.php');
c++	#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)

3. Reference

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

magnetometer→**registerValueCallback**(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

magnetometer→**set_highestValue**(newval)

Modifie la mémoire de valeur maximale observée.

magnetometer→**set_logFrequency**(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

magnetometer→**set_logicalName**(newval)

Modifie le nom logique du magnétomètre.

magnetometer→**set_lowestValue**(newval)

Modifie la mémoire de valeur minimale observée.

magnetometer→**set_reportFrequency**(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

magnetometer→**set_resolution**(newval)

Modifie la résolution des valeurs physique mesurées.

magnetometer→**set_userdata**(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

magnetometer→**wait_async**(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YMagnetometer.FindMagnetometer()**YMagnetometer****yFindMagnetometer()****YMagnetometer.FindMagnetometer()**

Permet de retrouver un magnétomètre d'après un identifiant donné.

`YMagnetometer` **FindMagnetometer**(`String` **func**)

L'identifiant peut être spécifié sous plusieurs formes:

- `NomLogiqueFonction`
- `NoSerieModule.IdentifiantFonction`
- `NoSerieModule.NomLogiqueFonction`
- `NomLogiqueModule.IdentifiantMatériel`
- `NomLogiqueModule.NomLogiqueFonction`

Cette fonction n'exige pas que le magnétomètre soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YMagnetometer.isOnline()` pour tester si le magnétomètre est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le magnétomètre sans ambiguïté

Retourne :

un objet de classe `YMagnetometer` qui permet ensuite de contrôler le magnétomètre.

YMagnetometer.FirstMagnetometer()

YMagnetometer

yFirstMagnetometer()

YMagnetometer.FirstMagnetometer()

Commence l'énumération des magnétomètres accessibles par la librairie.

YMagnetometer **FirstMagnetometer()**

Utiliser la fonction `YMagnetometer.nextMagnetometer()` pour itérer sur les autres magnétomètres.

Retourne :

un pointeur sur un objet `YMagnetometer`, correspondant au premier magnétomètre accessible en ligne, ou `null` si il n'y a pas de magnétomètres disponibles.

magnetometer→calibrateFromPoints()**YMagnetometer****magnetometer.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                        ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→describe()**YMagnetometer****magnetometer.describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du magnétomètre au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le magnétomètre (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

magnetometer→**get_advertisedValue()****YMagnetometer****magnetometer**→**advertisedValue()****magnetometer.get_advertisedValue()**

Retourne la valeur courante du magnétomètre (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du magnétomètre (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

magnetometer→get_currentRawValue()

YMagnetometer

magnetometer→currentRawValue()

magnetometer.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en mT, sous forme de nombre à virgule.

double **get_currentRawValue()**

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en mT, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

magnetometer→get_currentValue()**YMagnetometer****magnetometer→currentValue()****magnetometer.get_currentValue()**

Retourne la valeur actuelle du champ magnétique, en mT, sous forme de nombre à virgule.

double **get_currentValue()**

Retourne :

une valeur numérique représentant la valeur actuelle du champ magnétique, en mT, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

magnetometer→get_errorMessage()

YMagnetometer

magnetometer→errorMessage()

magnetometer.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

String get_errorMessage()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du magnétomètre.

magnetometer→get_errorType()**YMagnetometer****magnetometer→errorType()****magnetometer.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du magnétomètre.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du magnétomètre.

magnetometer→**get_friendlyName()**

YMagnetometer

magnetometer→**friendlyName()**

magnetometer.get_friendlyName()

Retourne un identifiant global du magnétomètre au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du magnétomètre si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du magnétomètre (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le magnétomètre en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

magnetometer→**get_functionDescriptor()****YMagnetometer****magnetometer**→**functionDescriptor()****magnetometer.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

magnetometer→**get_functionId()**

YMagnetometer

magnetometer→**functionId()**

magnetometer.get_functionId()

Retourne l'identifiant matériel du magnétomètre, sans référence au module.

String **get_functionId()** ()

Par exemple relay1.

Retourne :

une chaîne de caractères identifiant le magnétomètre (ex: relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FUNCTIONID_INVALID.

magnetometer→**get_hardwareId()****YMagnetometer****magnetometer**→**hardwareId()****magnetometer.get_hardwareId()**

Retourne l'identifiant matériel unique du magnétomètre au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du magnétomètre (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le magnétomètre (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

magnetometer→get_highestValue()

YMagnetometer

magnetometer→highestValue()

magnetometer.get_highestValue()

Retourne la valeur maximale observée pour le champ magnétique depuis le démarrage du module.

double get_highestValue()

Retourne :

une valeur numérique représentant la valeur maximale observée pour le champ magnétique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

magnetometer→get_logFrequency()**YMagnetometer****magnetometer→logFrequency()****magnetometer.get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

String **get_logFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

magnetometer→get_logicalName()
magnetometer→logicalName()
magnetometer.get_logicalName()

YMagnetometer

Retourne le nom logique du magnétomètre.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du magnétomètre.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

magnetometer→get_lowestValue()**YMagnetometer****magnetometer→lowestValue()****magnetometer.get_lowestValue()**

Retourne la valeur minimale observée pour le champ magnétique depuis le démarrage du module.

double **get_lowestValue()**

Retourne :

une valeur numérique représentant la valeur minimale observée pour le champ magnétique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_LOWESTVALUE_INVALID.

magnetometer→**get_module()**

YMagnetometer

magnetometer→**module()**

magnetometer.get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` [get_module\(\)](#)

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

magnetometer→get_recordedData()**YMagnetometer****magnetometer→recordedData()****magnetometer.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

YDataSet **get_recordedData**(long **startTime**, long **endTime**)

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

magnetometer→get_reportFrequency()

YMagnetometer

magnetometer→reportFrequency()

magnetometer.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String **get_reportFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

magnetometer→**get_resolution()**
magnetometer→**resolution()**
magnetometer.get_resolution()

YMagnetometer

Retourne la résolution des valeurs mesurées.

double **get_resolution()**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_RESOLUTION_INVALID.

magnetometer→**get_unit()**

YMagnetometer

magnetometer→**unit()**`magnetometer.get_unit()`

Retourne l'unité dans laquelle le champ magnétique est exprimée.

String **get_unit()**

Retourne :

une chaîne de caractères représentant l'unité dans laquelle le champ magnétique est exprimée

En cas d'erreur, déclenche une exception ou retourne Y_UNIT_INVALID.

magnetometer→get_userdata()
magnetometer→userData()
magnetometer.get_userdata()

YMagnetometer

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.

Object **get_userdata()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

magnetometer→**get_xValue()**
magnetometer→**xValue()**
magnetometer.get_xValue()

YMagnetometer

Retourne la composante X du champ magnétique, sous forme de nombre à virgule.

double **get_xValue()**

Retourne :

une valeur numérique représentant la composante X du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_XVALUE_INVALID.

magnetometer→get_yValue()**YMagnetometer****magnetometer→yValue()****magnetometer.get_yValue()**

Retourne la composante Y du champ magnétique, sous forme de nombre à virgule.

double **get_yValue()**

Retourne :

une valeur numérique représentant la composante Y du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_YVALUE_INVALID.

magnetometer→get_zValue()

YMagnetometer

magnetometer→zValue()

magnetometer.get_zValue()

Retourne la composante Z du champ magnétique, sous forme de nombre à virgule.

`double get_zValue()`

Retourne :

une valeur numérique représentant la composante Z du champ magnétique, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_ZVALUE_INVALID.

magnetometer→**isOnline()****YMagnetometer****magnetometer.isOnline()**

Vérifie si le module hébergeant le magnétomètre est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache du magnétomètre sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le magnétomètre est joignable, `false` sinon

magnetometer→**load()****magnetometer.load()****YMagnetometer**

Met en cache les valeurs courantes du magnétomètre, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→loadCalibrationPoints()**YMagnetometer****magnetometer.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**nextMagnetometer()**

YMagnetometer

magnetometer.nextMagnetometer()

Continue l'énumération des magnétomètres commencée à l'aide de `yFirstMagnetometer()`.

`YMagnetometer` **nextMagnetometer()**

Retourne :

un pointeur sur un objet `YMagnetometer` accessible en ligne, ou `null` lorsque l'énumération est terminée.

magnetometer→registerTimedReportCallback()**YMagnetometer****magnetometer.registerTimedReportCallback()**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

magnetometer→registerValueCallback()**YMagnetometer****magnetometer.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

magnetometer→set_highestValue()**YMagnetometer****magnetometer→setHighestValue()****magnetometer.set_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**set_logFrequency()**

YMagnetometer

magnetometer→**setLogFrequency()**

magnetometer.set_logFrequency()

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**set_logicalName()****YMagnetometer****magnetometer**→**setLogicalName()****magnetometer.set_logicalName()**

Modifie le nom logique du magnétomètre.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du magnétomètre.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**set_lowestValue()**
magnetometer→**setLowestValue()**
magnetometer.set_lowestValue()

YMagnetometer

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→set_reportFrequency()**YMagnetometer****magnetometer→setReportFrequency()****magnetometer.set_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**set_resolution()**

YMagnetometer

magnetometer→**setResolution()**

magnetometer.set_resolution()

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

magnetometer→**set_userdata()****YMagnetometer****magnetometer**→**setUserData()****magnetometer.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

3.25. Valeur mesurée

Les objets YMeasure sont utilisés dans l'interface de programmation Yoctopuce pour représenter une valeur observée un moment donnée. Ces objets sont utilisés en particulier en conjonction avec la classe YDataSet.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_api.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YAPI = yoctolib.YAPI; var YModule = yoctolib.YModule;
php	require_once('yocto_api.php');
cpp	#include "yocto_api.h"
m	#import "yocto_api.h"
pas	uses yocto_api;
vb	yocto_api.vb
cs	yocto_api.cs
java	import com.yoctopuce.YoctoAPI.YModule;
py	from yocto_api import *

Méthodes des objets YMeasure

measure→get_averageValue()

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par la mesure.

measure→get_endTimeUTC()

Retourne l'heure absolue de la fin de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

measure→get_maxValue()

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par la mesure.

measure→get_minValue()

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par la mesure.

measure→get_startTimeUTC()

Retourne l'heure absolue du début de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

measure→**get_averageValue()****YMeasure****measure**→**averageValue()****measure.get_averageValue()**

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par la mesure.

double **get_averageValue()**

Retourne :

un nombre décimal correspondant à la valeur moyenne observée.

measure→**get_endTimeUTC()**

YMeasure

measure→**endTimeUTC()**

measure.get_endTimeUTC()

Retourne l'heure absolue de la fin de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

double **get_endTimeUTC()** ()

Lors que l'enregistrement de données se fait à une fréquence supérieure à une mesure par seconde, le timestamp peuvent inclure une fraction décimale.

Retourne :

un nombre réel positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 UTC et la fin de la mesure.

measure→**get_maxValue()****YMeasure****measure**→**maxValue()****measure.get_maxValue()**

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par la mesure.

double **get_maxValue()**

Retourne :

un nombre décimal correspondant à la plus grande valeur observée.

measure→**get_minValue()**

YMeasure

measure→**minValue()**`measure.get_minValue()`

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par la mesure.

double **get_minValue()**

Retourne :

un nombre décimal correspondant à la plus petite valeur observée.

measure→get_startTimeUTC()**YMeasure****measure→startTimeUTC()****measure.get_startTimeUTC()**

Retourne l'heure absolue du début de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

```
double get_startTimeUTC( )
```

Lors que l'enregistrement de données se fait à une fréquence supérieure à une mesure par seconde, le timestamp peuvent inclure une fraction décimale.

Retourne :

un nombre réel positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 UTC et le début de la mesure.

3.26. 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(funcn) Permet de retrouver un module d'après son numéro de série ou son nom logique.
yFirstModule() Commence l'énumération des modules accessibles par la librairie.
Méthodes des objets YModule
module→checkFirmware(path, onlynew) Test si le fichier byn est valid pour le module.
module→describe() Retourne un court texte décrivant le module.
module→download(pathname) Télécharge le fichier choisi du module et retourne son contenu.
module→functionCount() Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.
module→functionId(functionIndex) Retourne l'identifiant matériel de la <i>nième</i> fonction du module.
module→functionName(functionIndex) Retourne le nom logique de la <i>nième</i> fonction du module.
module→functionValue(functionIndex) Retourne la valeur publiée par la <i>nième</i> 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 chaîne de caractère contenant les derniers logs du module.
module → get_logicalName()	Retourne le nom logique du module.
module → get_luminosity()	Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).
module → get_persistentSettings()	Retourne l'état courant des réglages persistents du module.
module → get_productId()	Retourne l'identifiant USB du module, préprogrammé en usine.
module → get_productName()	Retourne le nom commercial du module, préprogrammé en usine.
module → get_productRelease()	Retourne le numéro de version matériel du module, préprogrammé en usine.
module → get_rebootCountdown()	Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.
module → get_serialNumber()	Retourne le numéro de série du module, préprogrammé en usine.
module → get_upTime()	Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module
module → get_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 <code>set_userData</code> .
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 <code>yFirstModule()</code> .
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.

YModule.FindModule()**YModule****yFindModule()**`YModule.FindModule()`

Permet de retrouver un module d'après son numéro de série ou son nom logique.

`YModule FindModule(String func)`

Cette fonction n'exige pas que le module soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YModule.isOnline()` pour tester si le module est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères contenant soit le numéro de série, soit le nom logique du module désiré

Retourne :

un objet de classe `YModule` qui permet ensuite de contrôler le module ou d'obtenir de plus amples informations sur le module.

YModule.FirstModule()

YModule

yFirstModule()`YModule.FirstModule()`

Commence l'énumération des modules accessibles par la librairie.

`YModule` **FirstModule()**

Utiliser la fonction `YModule.nextModule()` pour itérer sur les autres modules.

Retourne :

un pointeur sur un objet `YModule`, correspondant au premier module accessible en ligne, ou `null` si aucun module n'a été trouvé.

module→**checkFirmware()**
module.checkFirmware()

YModule

Test si le fichié byn est valid pour le module.

String **checkFirmware**(**String** **path**, **boolean** **onlynew**)

Cette methode est utile pour tester si il est nécessaire de mettre à jour le module avec un nouveau firmware. Il est possible de passer un répertoire qui contiens plusieurs fichier byn. Dans ce cas cette methode retourne le path du fichier byn compatible le plus récent. Si le parametre onlynew est vrais les firmware équivalent ou plus ancien au firmware installé sont ignorés.

Paramètres :

path le path sur un fichier byn ou un répertoire contenant plusieurs fichier byn
onlynew retourne uniquement les fichier strictement plus récent

Retourne :

: le path du fichier byn a utiliser ou une chaine vide si aucun firmware plus récent est disponible En cas d'erreur, déclenche une exception ou retourne une chaine de caractère qui comment par "error:".

module→**describe()**`module.describe()`

YModule

Retourne un court texte décrivant le module.

`String describe()`

Ce texte peut contenir soit le nom logique du module, soit son numéro de série.

Retourne :

une chaîne de caractères décrivant le module

module→**get_beacon()****YModule****module**→**beacon()**`module.get_beacon()`

Retourne l'état de la balise de localisation.

`int get_beacon()`

Retourne :

soit `Y_BEACON_OFF`, soit `Y_BEACON_ON`, selon l'état de la balise de localisation

En cas d'erreur, déclenche une exception ou retourne `Y_BEACON_INVALID`.

module→**get_errorMessage()**

YModule

module→**errorMessage()**

module.errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du module

module→**get_errorType()****YModule****module**→**errorType()**`module.get_errorType()`

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

`int` **get_errorType()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du module

module→**get_firmwareRelease()**

YModule

module→**firmwareRelease()**

module.get_firmwareRelease()

Retourne la version du logiciel embarqué du module.

String **get_firmwareRelease()**

Retourne :

une chaîne de caractères représentant la version du logiciel embarqué du module

En cas d'erreur, déclenche une exception ou retourne Y_FIRMWARERELEASE_INVALID.

module→**get_hardwareId()****YModule****module**→**hardwareId()**`module.get_hardwareId()`

Retourne l'identifiant unique du module.

`String` **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module suivi de la chaîne ".module".

Retourne :

une chaîne de caractères identifiant la fonction

module→**get_lastLogs()**

YModule

module→**lastLogs()****module.get_lastLogs()**

Retourne une chaine de caractère contenant les derniers logs du module.

String **get_lastLogs()**

Cette methode retourne les derniers logs qui sont encore stocké dans le module.

Retourne :

une chaine de caractère contenant les derniers logs du module. En cas d'erreur, déclenche une exception ou retourne YAPI_INVALID_STRING.

module→**get_logicalName()****YModule****module**→**logicalName()****module.get_logicalName()**

Retourne le nom logique du module.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du module

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

module→**get_luminosity()**

YModule

module→**luminosity()**`module.get_luminosity()`

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

`int get_luminosity()`

Retourne :

un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

En cas d'erreur, déclenche une exception ou retourne `Y_LUMINOSITY_INVALID`.

module→**get_persistentSettings()****YModule****module**→**persistentSettings()****module.get_persistentSettings()**

Retourne l'état courant des réglages persistents du module.

int **get_persistentSettings()**

Retourne :

une valeur parmi Y_PERSISTENTSETTINGS_LOADED, Y_PERSISTENTSETTINGS_SAVED et Y_PERSISTENTSETTINGS_MODIFIED représentant l'état courant des réglages persistents du module

En cas d'erreur, déclenche une exception ou retourne Y_PERSISTENTSETTINGS_INVALID.

module→**get_productId()**

YModule

module→**productId()**`module.productId()`

Retourne l'identifiant USB du module, préprogrammé en usine.

`int` **get_productId()** ()

Retourne :

un entier représentant l'identifiant USB du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne `Y_PRODUCTID_INVALID`.

module→**get_productName()**
module→**productName()**
module.get_productName()

YModule

Retourne le nom commercial du module, préprogrammé en usine.

String **get_productName()**

Retourne :

une chaîne de caractères représentant le nom commercial du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y_PRODUCTNAME_INVALID.

module→**get_productRelease()**

YModule

module→**productRelease()**

module.get_productRelease()

Retourne le numéro de version matériel du module, préprogrammé en usine.

int **get_productRelease()**

Retourne :

un entier représentant le numéro de version matériel du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y_PRODUCTRELEASE_INVALID.

module→**get_rebootCountdown()****YModule****module**→**rebootCountdown()****module.get_rebootCountdown()**

Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

```
int get_rebootCountdown( )
```

Retourne :

un entier représentant le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé

En cas d'erreur, déclenche une exception ou retourne `Y_REBOOTCOUNTDOWN_INVALID`.

module→**get_serialNumber()**

YModule

module→**serialNumber()**

module.get_serialNumber()

Retourne le numéro de série du module, préprogrammé en usine.

String **get_serialNumber()**

Retourne :

une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine

En cas d'erreur, déclenche une exception ou retourne Y_SERIALNUMBER_INVALID.

module→**get_upTime()****YModule****module**→**upTime()**`module.get_upTime()`

Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module

`long get_upTime()`

Retourne :

un entier représentant le nombre de millisecondes écoulées depuis la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne `Y_UPTIME_INVALID`.

module→**get_usbCurrent()**

YModule

module→**usbCurrent()**`module.get_usbCurrent()`

Retourne le courant consommé par le module sur le bus USB, en milliampères.

`int get_usbCurrent()`

Retourne :

un entier représentant le courant consommé par le module sur le bus USB, en milliampères

En cas d'erreur, déclenche une exception ou retourne `Y_USBCURRENT_INVALID`.

module→**get_userdata()****YModule****module**→**userData()****module.get_userdata()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

module→**get_userVar()**

YModule

module→**userVar()****module.get_userVar()**

Retourne la valeur entière précédemment stockée dans cet attribut.

```
int get_userVar( )
```

Au démarrage du module (ou après un redémarrage), la valeur est toujours zéro.

Retourne :

un entier représentant la valeur entière précédemment stockée dans cet attribut

En cas d'erreur, déclenche une exception ou retourne Y_USERVAR_INVALID.

module→**isOnline()****module.isOnline()**

YModule

Vérifie si le module est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs du module en cache sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le module est joignable, `false` sinon

module→**load()****module.load()****YModule**

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**nextModule()****module.nextModule()****YModule**

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

YModule **nextModule()**

Retourne :

un pointeur sur un objet `YModule` accessible en ligne, ou `null` lorsque l'énumération est terminée.

module→**reboot()**`module.reboot ()`**YModule**

Agende un simple redémarrage du module dans un nombre donné de secondes.

```
int reboot( int secBeforeReboot)
```

Paramètres :

secBeforeReboot nombre de secondes avant de redémarrer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→registerLogCallback()**YModule****module.registerLogCallback()**

Enregistre une fonction de callback qui sera appelée à chaque fois le module émet un message de log.

```
void registerLogCallback( LogCallback callback)
```

Utile pour déboguer le fonctionnement d'un module Yoctopuce.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet module qui a produit un log, un chaîne de caractère qui contient le log

module→revertFromFlash()

YModule

module.revertFromFlash()

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

int revertFromFlash()

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**saveToFlash()**`module.saveToFlash()`**YModule**

Sauve les réglages courants dans la mémoire non volatile du module.

`int saveToFlash()`

Attention le nombre total de sauvegardes possibles durant la vie du module est limité (environ 100000 cycles). N'appellez pas cette fonction dans une boucle.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**set_allSettings()**

YModule

module→**setAllSettings()**

module.set_allSettings()

Restore tous les paramètres du module.

```
int set_allSettings( )
```

Utile pour restaurer les noms logiques et les calibrations du module depuis un sauvgarde.

Paramètres :

settings un buffer binaire avec tous les paramètres

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**set_beacon()****YModule****module**→**setBeacon()**`module.set_beacon()`

Allume ou éteint la balise de localisation du module.

```
int set_beacon( int newval)
```

Paramètres :

newval soit Y_BEACON_OFF, soit Y_BEACON_ON

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**set_logicalName()****YModule****module**→**setLogicalName()****module.set_logicalName()**

Change le nom logique du module.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**set_luminosity()****YModule****module**→**setLuminosity()****module.set_luminosity()**

Modifie la luminosité des leds informatives du module.

```
int set_luminosity( int newval)
```

Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la luminosité des leds informatives du module

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**set_userdata()**

YModule

module→**setUserData()****module.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

module→**set_userVar()****YModule****module**→**setUserVar()**`module.set_userVar()`

Retourne la valeur entière précédemment stockée dans cet attribut.

```
int set_userVar( int newval)
```

Au démarrage du module (ou après un redémarrage), la valeur est toujours zéro.

Paramètres :

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→**triggerFirmwareUpdate()****YModule****module.triggerFirmwareUpdate()**

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

```
int triggerFirmwareUpdate( int secBeforeReboot)
```

Paramètres :

secBeforeReboot nombre de secondes avant de redémarrer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

module→updateFirmware()**YModule****module.updateFirmware()**

Prepare une mise à jour de firmware du module.

YFirmwareUpdate **updateFirmware**(String **path**)

Cette methode un object YFirmwareUpdate qui est utilisé pour mettre à jour le firmware du module.

Paramètres :

path le path sur un fichier byn

Retourne :

: Un object YFirmwareUpdate

3.27. Interface de la fonction Motor

La librairie de programmation yoctopuce permet de piloter la puissance envoyée au moteur pour le faire tourner aussi bien dans un sens que dans l'autre, mais aussi de piloter des accélérations linéaires: le moteur accélère alors tout seul sans que vous vous ayez à vous en occuper. La librairie permet aussi de freiner le moteur: cela est réalisé en court-circuitant les pôles du moteur, ce qui le transforme en frein électro-magnétique.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_motor.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YMotor = yoctolib.YMotor;</code>
php	<code>require_once('yocto_motor.php');</code>
c++	<code>#include "yocto_motor.h"</code>
m	<code>#import "yocto_motor.h"</code>
pas	<code>uses yocto_motor;</code>
vb	<code>yocto_motor.vb</code>
cs	<code>yocto_motor.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YMotor;</code>
py	<code>from yocto_motor import *</code>

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()**

Rearme la sécurité failsafe du contrôleur.

motor→**load(msValidity)**

Met en cache les valeurs courantes du moteur, avec une durée de validité spécifiée.

motor→**load_async(msValidity, callback, context)**

Met en cache les valeurs courantes du moteur, avec une durée de validité spécifiée.

motor→**nextMotor()**

Continue l'énumération des moteur commencée à l'aide de `yFirstMotor()`.

motor→**registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

motor→**resetStatus()**

Réinitialise l'état du contrôleur à IDLE.

motor→**set_brakingForce(newval)**

3. Reference

Modifie immédiatement la force de freinage appliquée au moteur (en pourcents).

motor→**set_cutOffVoltage(newval)**

Modifie la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

motor→**set_drivingForce(newval)**

Modifie immédiatement la puissance envoyée au moteur.

motor→**set_failSafeTimeout(newval)**

Modifie le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

motor→**set_frequency(newval)**

Modifie la fréquence du signal PWM utilisée pour contrôler le moteur.

motor→**set_logicalName(newval)**

Modifie le nom logique du moteur.

motor→**set_overCurrentLimit(newval)**

Modifie la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

motor→**set_starterTime(newval)**

Modifie la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

motor→**set_userData(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.

YMotor.FindMotor()**YMotor****yFindMotor()**`YMotor.FindMotor()`

Permet de retrouver un moteur d'après un identifiant donné.

`YMotor` **FindMotor**(`String` **func**)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le moteur soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YMotor.isOnline()` pour tester si le moteur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le moteur sans ambiguïté

Retourne :

un objet de classe `YMotor` qui permet ensuite de contrôler le moteur.

YMotor.FirstMotor()

YMotor

yFirstMotor()`YMotor.FirstMotor()`

Commence l'énumération des moteur accessibles par la librairie.

`YMotor` **FirstMotor()**

Utiliser la fonction `YMotor.nextMotor()` pour itérer sur les autres moteur.

Retourne :

un pointeur sur un objet `YMotor`, correspondant au premier moteur accessible en ligne, ou `null` si il n'y a pas de moteur disponibles.

motor→**brakingForceMove()****YMotor****motor.brakingForceMove()**

Modifie progressivement la force de freinage appliquée au moteur sur une durée donnée.

```
int brakingForceMove( double targetPower, int delay)
```

Paramètres :

targetPower force de freinage finale, en pourcentage

delay durée (en ms) sur laquelle le changement de puissance sera effectué

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→**describe()****motor.describe()****YMotor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du moteur au format
`TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le moteur (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

motor→**drivingForceMove()****YMotor****motor.drivingForceMove()**

Modifie progressivement la puissance envoyée au moteur sur une durée donnée.

```
int drivingForceMove( double targetPower, int delay)
```

Paramètres :

targetPower puissance finale désirée, en pourcentage de -100% à +100%

delay durée (en ms) sur laquelle le changement de puissance sera effectué

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→**get_advertisedValue()**

YMotor

motor→**advertisedValue()**

motor.get_advertisedValue()

Retourne la valeur courante du moteur (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du moteur (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

motor→**get_brakingForce()****YMotor****motor**→**brakingForce()**`motor.get_brakingForce()`

Retourne la force de freinage appliquée au moteur, sous forme de pourcentage.

double **get_brakingForce()**

La valeur 0 correspond ne pas freiner (moteur en roue libre).

Retourne :

une valeur numérique représentant la force de freinage appliquée au moteur, sous forme de pourcentage

En cas d'erreur, déclenche une exception ou retourne Y_BRAKINGFORCE_INVALID.

motor→**get_cutOffVoltage()**

YMotor

motor→**cutOffVoltage()**

motor.get_cutOffVoltage()

Retourne la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

double **get_cutOffVoltage()**

Ce réglage permet d'éviter d'endommager un accumulateur en continuant à l'utiliser une fois "vide".

Retourne :

une valeur numérique représentant la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation

En cas d'erreur, déclenche une exception ou retourne `Y_CUTOFFVOLTAGE_INVALID`.

motor→**get_drivingForce()****YMotor****motor**→**drivingForce()**`motor.get_drivingForce()`

Retourne la puissance actuelle envoyée au moteur, sous forme de nombre réel entre -100% et +100%.

double **get_drivingForce()** ()

Retourne :

une valeur numérique représentant la puissance actuelle envoyée au moteur, sous forme de nombre réel entre -100% et +100%

En cas d'erreur, déclenche une exception ou retourne Y_DRIVINGFORCE_INVALID.

motor→**get_errorMessage()**

YMotor

motor→**errorMessage()**

motor.errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moteur.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du moteur.

motor→**get_errorType()****YMotor****motor**→**errorType()****motor.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moteur.

int **get_errorType()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du moteur.

motor→**get_failSafeTimeout()**

YMotor

motor→**failSafeTimeout()**

motor.get_failSafeTimeout()

Retourne le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

int get_failSafeTimeout()

Passé ce delai, le contrôleur arrêtera le moteur et passera en mode erreur FAILSAFE. La sécurité failsafe est désactivée quand la valeur est à zéro.

Retourne :

un entier représentant le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle

En cas d'erreur, déclenche une exception ou retourne Y_FAILSAFETIMEOUT_INVALID.

motor→**get_frequency()****YMotor****motor**→**frequency()**`motor.get_frequency()`

Retourne la fréquence du signal PWM utilisé pour contrôler le moteur.

double **get_frequency()**

Retourne :

une valeur numérique représentant la fréquence du signal PWM utilisé pour contrôler le moteur

En cas d'erreur, déclenche une exception ou retourne `Y_FREQUENCY_INVALID`.

motor→**get_friendlyName()**

YMotor

motor→**friendlyName()****motor.get_friendlyName()**

Retourne un identifiant global du moteur au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du moteur si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du moteur (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le moteur en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

motor→**get_functionDescriptor()****YMotor****motor**→**functionDescriptor()****motor.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

motor→**get_functionId()**

YMotor

motor→**functionId()****motor.get_functionId()**

Retourne l'identifiant matériel du moteur, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le moteur (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

motor→**get_hardwareId()****YMotor****motor**→**hardwareId()**`motor.get_hardwareId()`

Retourne l'identifiant matériel unique du moteur au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du moteur (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le moteur (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

motor→**get_logicalName()**

YMotor

motor→**logicalName()**`motor.get_logicalName()`

Retourne le nom logique du moteur.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du moteur.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

motor→**get_module()****YMotor****motor**→**module()****motor.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

motor→**get_motorStatus()****YMotor****motor**→**motorStatus()****motor.get_motorStatus()**

Retourne l'état du contrôleur de moteur.

```
int get_motorStatus( )
```

Les états possibles sont: IDLE si le moteur est à l'arrêt/en roue libre, prêt à démarrer; FORWD si le contrôleur fait tourner le moteur en marche avant; BACKWD si le contrôleur fait tourner le moteur en marche arrière; BRAKE si le contrôleur est en train de freiner; LOVOLT si le contrôleur a détecté une tension trop basse; HICURR si le contrôleur a détecté une surconsommation; HIHEAT si le contrôleur a détecté une surchauffe; FAILSF si le contrôleur est passé en protection failsafe.

Si le contrôleur est en erreur (LOVOLT, HICURR, HIHEAT,FAILSF), il doit être explicitement réinitialisé avec la fonction `resetStatus`.

Retourne :

une valeur parmi Y_MOTORSTATUS_IDLE, Y_MOTORSTATUS_BRAKE, Y_MOTORSTATUS_FORWD, Y_MOTORSTATUS_BACKWD, Y_MOTORSTATUS_LOVOLT, Y_MOTORSTATUS_HICURR, Y_MOTORSTATUS_HIHEAT et Y_MOTORSTATUS_FAILSF représentant l'état du contrôleur de moteur

En cas d'erreur, déclenche une exception ou retourne Y_MOTORSTATUS_INVALID.

motor→**get_overCurrentLimit()****YMotor****motor**→**overCurrentLimit()****motor.get_overCurrentLimit()**

Retourne la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

```
int get_overCurrentLimit( )
```

Une valeur nulle signifie qu'aucune limite n'est définie.

Retourne :

un entier représentant la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur

En cas d'erreur, déclenche une exception ou retourne `Y_OVERCURRENTLIMIT_INVALID`.

motor→**get_starterTime()**

YMotor

motor→**starterTime()****motor.get_starterTime()**

Retourne la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

int **get_starterTime()** ()

Retourne :

un entier représentant la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage

En cas d'erreur, déclenche une exception ou retourne Y_STARTERTIME_INVALID.

motor→**get_userData()****YMotor****motor**→**userData()**`motor.getUserData()`

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

Object `get_userdata()`

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

motor→**isOnline()**`motor.isOnline()`

YMotor

Vérifie si le module hébergeant le moteur est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du moteur sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le moteur est joignable, `false` sinon

motor→**keepALive()****motor.keepALive()****YMotor**

Réarme la sécurité failsafe du contrôleur.

int keepALive()

Lorsque le moteur est en marche et que la sécurité failsafe est activée, cette fonction doit être appelée périodiquement pour confirmer le bon fonctionnement du processus de contrôle. A défaut, le moteur s'arrêtera automatiquement au bout du temps prévu. Notez que l'appel à une fonction de type *set* du moteur réarme aussi la sécurité failsafe.

motor→**load()****motor.load()****YMotor**

Met en cache les valeurs courantes du moteur, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→**nextMotor()**`motor.nextMotor()`**YMotor**

Continue l'énumération des moteur commencée à l'aide de `yFirstMotor()`.

`YMotor` **nextMotor()**

Retourne :

un pointeur sur un objet `YMotor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

motor→registerValueCallback()**YMotor****motor.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

motor→**resetStatus()****motor.resetStatus()****YMotor**

Réinitialise l'état du contrôleur à IDLE.

int **resetStatus()**

Cette fonction doit être explicitement appelée après toute condition d'erreur pour permettre au contrôleur de repartir.

motor→**set_brakingForce()**

YMotor

motor→**setBrakingForce()**

motor.set_brakingForce()

Modifie immédiatement la force de freinage appliquée au moteur (en pourcents).

```
int set_brakingForce( double newval)
```

La valeur 0 correspond à ne pas freiner (moteur en roue libre). Lorsque la force de freinage est changée, la puissance de traction est remise à zéro.

Paramètres :

newval une valeur numérique représentant immédiatement la force de freinage appliquée au moteur (en pourcents)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→**set_cutOffVoltage()****YMotor****motor**→**setCutOffVoltage()****motor.set_cutOffVoltage()**

Modifie la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

```
int set_cutOffVoltage( double newval)
```

Ce réglage permet d'éviter d'endommager un accumulateur en continuant à l'utiliser une fois "vide". Attention, quel que soit le réglage du cutoff, le variateur passera en erreur si l'alimentation passe (même brièvement) en dessous de 3V.

Paramètres :

newval une valeur numérique représentant la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→**set_drivingForce()****YMotor****motor**→**setDrivingForce()****motor.set_drivingForce()**

Modifie immédiatement la puissance envoyée au moteur.

```
int set_drivingForce( double newval)
```

La valeur est donnée en pourcentage de -100% à +100%. Si vous voulez ménager votre mécanique et éviter d'induire des consommations excessives qui pourraient dépasser les capacités du contrôleur, évitez les changements de régime trop brusques. Par exemple, passer brutalement de marche avant à marche arrière est une très mauvaise idée. A chaque fois que la puissance envoyée au moteur est changée, le freinage est remis à zéro.

Paramètres :

newval une valeur numérique représentant immédiatement la puissance envoyée au moteur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→**set_failSafeTimeout()****YMotor****motor**→**setFailSafeTimeout()****motor.set_failSafeTimeout()**

Modifie le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

```
int set_failSafeTimeout( int newval)
```

Passé ce délai, le contrôleur arrêtera le moteur et passera en mode erreur FAILSAFE. La sécurité failsafe est désactivée quand la valeur est à zéro.

Paramètres :

newval un entier représentant le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→**set_frequency()**

YMotor

motor→**setFrequency()**`motor.set_frequency()`

Modifie la fréquence du signal PWM utilisée pour contrôler le moteur.

```
int set_frequency( double newval)
```

Une fréquence basse est généralement plus efficace (les composant chauffent moins et le moteur démarre plus facilement), mais un bruit audible peut être généré. Une fréquence élevée peut réduire le bruit, mais il y a plus d'énergie perdue en chaleur.

Paramètres :

newval une valeur numérique représentant la fréquence du signal PWM utilisée pour contrôler le moteur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→**set_logicalName()****YMotor****motor**→**setLogicalName()****motor.set_logicalName()**

Modifie le nom logique du moteur.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du moteur.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→**set_overCurrentLimit()**

YMotor

motor→**setOverCurrentLimit()**

motor.set_overCurrentLimit()

Modifie la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

```
int set_overCurrentLimit( int newval)
```

Une valeur nulle signifie qu'aucune limite n'est définie. Attention, quel que soit le réglage choisi, le variateur passera en erreur si le courant passe, même brièvement, en dessus de 32A.

Paramètres :

newval un entier représentant la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→**set_starterTime()****YMotor****motor**→**setStarterTime()**`motor.set_starterTime()`

Modifie la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

`int set_starterTime(int newval)`**Paramètres :**

newval un entier représentant la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

motor→**set_userdata()**

YMotor

motor→**setUserData()****motor.set_userdata()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

void **set_userdata**(Object **data**)

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

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');
c++	#include "yocto_network.h"
m	#import "yocto_network.h"
pas	uses yocto_network;
vb	yocto_network.vb
cs	yocto_network.cs
java	import com.yoctopuce.YoctoAPI.YNetwork;
py	from yocto_network import *

Fonction globales

yFindNetwork(func)

Permet de retrouver une interface réseau d'après un identifiant donné.

yFirstNetwork()

Commence l'énumération des interfaces réseau accessibles par la librairie.

Méthodes des objets YNetwork

network→callbackLogin(username, password)

Contacte le callback de notification et sauvegarde un laisser-passer pour s'y connecter.

network→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

network→get_adminPassword()

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide.

network→get_advertisedValue()

Retourne la valeur courante de l'interface réseau (pas plus de 6 caractères).

network→get_callbackCredentials()

Retourne une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide.

network→get_callbackEncoding()

Retourne l'encodage à utiliser pour représenter les valeurs notifiées par callback.

network→get_callbackMaxDelay()

Retourne l'attente maximale entre deux notifications par callback, en secondes.

network→get_callbackMethod()

Retourne la méthode HTTP à utiliser pour signaler les changements d'état par callback.

network→get_callbackMinDelay()

Retourne l'attente minimale entre deux notifications par callback, en secondes.

network→get_callbackUrl()

Retourne l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

network→get_discoverable()

3. Reference

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

network→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

network→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

network→get_friendlyName()

Retourne un identifiant global de l'interface réseau au format NOM_MODULE . NOM_FONCTION.

network→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

network→get_functionId()

Retourne l'identifiant matériel de l'interface réseau, sans référence au module.

network→get_hardwareId()

Retourne l'identifiant matériel unique de l'interface réseau au format SERIAL . FUNCTIONID.

network→get_ipAddress()

Retourne l'adresse IP utilisée par le module Yoctopuce.

network→get_logicalName()

Retourne le nom logique de l'interface réseau.

network→get_macAddress()

Retourne l'adresse MAC de l'interface réseau, unique pour chaque module.

network→get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

network→get_module_async(callback, context)

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

network→get_poeCurrent()

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

network→get_primaryDNS()

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

network→get_readiness()

Retourne l'état de fonctionnement atteint par l'interface réseau.

network→get_router()

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

network→get_secondaryDNS()

Retourne l'adresse IP du serveur de noms secondaire que le module doit utiliser.

network→get_subnetMask()

Retourne le masque de sous-réseau utilisé par le module.

network→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.

network→get_userPassword()

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

network→get_wwwWatchdogDelay()

Retourne la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

network→isOnline()

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

network→isOnline_async(callback, context)

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

network→load(msValidity)

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

network→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

network→nextNetwork()

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

network→ping(host)

Ping `str_host` pour vérifier la connexion réseau.

network→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

network→set_adminPassword(newval)

Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.

network→set_callbackCredentials(newval)

Modifie le laisser-passer pour se connecter à l'adresse de callback.

network→set_callbackEncoding(newval)

Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.

network→set_callbackMaxDelay(newval)

Modifie l'attente maximale entre deux notifications par callback, en secondes.

network→set_callbackMethod(newval)

Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.

network→set_callbackMinDelay(newval)

Modifie l'attente minimale entre deux notifications par callback, en secondes.

network→set_callbackUrl(newval)

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

network→set_discoverable(newval)

Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

network→set_logicalName(newval)

Modifie le nom logique de l'interface réseau.

network→set_primaryDNS(newval)

Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.

network→set_secondaryDNS(newval)

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

network→set_userData(data)

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

network→set_userPassword(newval)

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

network→set_wwwWatchdogDelay(newval)

Modifie la durée de perte de connexion WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

network→useDHCP(fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)

3. Reference

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

network→useStaticIP(ipAddress, subnetMaskLen, router)

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

network→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YNetwork.FindNetwork()**YNetwork****yFindNetwork()**`YNetwork.FindNetwork()`

Permet de retrouver une interface réseau d'après un identifiant donné.

`YNetwork FindNetwork(String func)`

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'interface réseau soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YNetwork.isOnline()` pour tester si l'interface réseau est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence l'interface réseau sans ambiguïté

Retourne :

un objet de classe `YNetwork` qui permet ensuite de contrôler l'interface réseau.

YNetwork.FirstNetwork()

YNetwork

yFirstNetwork()`YNetwork.FirstNetwork()`

Commence l'énumération des interfaces réseau accessibles par la librairie.

`YNetwork` **FirstNetwork()**

Utiliser la fonction `YNetwork.nextNetwork()` pour itérer sur les autres interfaces réseau.

Retourne :

un pointeur sur un objet `YNetwork`, correspondant à la première interface réseau accessible en ligne, ou `null` si il n'y a pas de interfaces réseau disponibles.

network→callbackLogin()**YNetwork****network.callbackLogin()**

Contacte le callback de notification et sauvegarde un laisser-passer pour s'y connecter.

```
int callbackLogin( String username, String password)
```

Le mot de passe ne sera pas stocké dans le module, mais seulement une version hashée non réversible. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

username nom d'utilisateur pour s'identifier au callback

password mot de passe pour s'identifier au callback

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→describe()`network.describe()`**YNetwork**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant l'interface réseau (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

network→get_adminPassword()**YNetwork****network→adminPassword()****network.get_adminPassword()**

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide.

String **get_adminPassword()**

Retourne :

une chaîne de caractères représentant une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne Y_ADMINPASSWORD_INVALID.

network→**get_advertisedValue()**

YNetwork

network→**advertisedValue()**

network.get_advertisedValue()

Retourne la valeur courante de l'interface réseau (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante de l'interface réseau (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

network→get_callbackCredentials()**YNetwork****network→callbackCredentials()****network.get_callbackCredentials()**

Retourne une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide.

String get_callbackCredentials()**Retourne :**

une chaîne de caractères représentant une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne `Y_CALLBACKCREDENTIALS_INVALID`.

network→get_callbackEncoding()**YNetwork****network→callbackEncoding()****network.get_callbackEncoding()**

Retourne l'encodage à utiliser pour représenter les valeurs notifiées par callback.

```
int get_callbackEncoding( )
```

Retourne :

une valeur parmi Y_CALLBACKENCODING_FORM, Y_CALLBACKENCODING_JSON, Y_CALLBACKENCODING_JSON_ARRAY, Y_CALLBACKENCODING_CSV et Y_CALLBACKENCODING_YOCTO_API représentant l'encodage à utiliser pour représenter les valeurs notifiées par callback

En cas d'erreur, déclenche une exception ou retourne Y_CALLBACKENCODING_INVALID.

network→get_callbackMaxDelay()**YNetwork****network→callbackMaxDelay()****network.get_callbackMaxDelay()**

Retourne l'attente maximale entre deux notifications par callback, en secondes.

int **get_callbackMaxDelay()**

Retourne :

un entier représentant l'attente maximale entre deux notifications par callback, en secondes

En cas d'erreur, déclenche une exception ou retourne Y_CALLBACKMAXDELAY_INVALID.

network→get_callbackMethod()**YNetwork****network→callbackMethod()****network.get_callbackMethod()**

Retourne la méthode HTTP à utiliser pour signaler les changements d'état par callback.

```
int get_callbackMethod( )
```

Retourne :

une valeur parmi Y_CALLBACKMETHOD_POST, Y_CALLBACKMETHOD_GET et Y_CALLBACKMETHOD_PUT représentant la méthode HTTP à utiliser pour signaler les changements d'état par callback

En cas d'erreur, déclenche une exception ou retourne Y_CALLBACKMETHOD_INVALID.

network→get_callbackMinDelay()**YNetwork****network→callbackMinDelay()****network.get_callbackMinDelay()**

Retourne l'attente minimale entre deux notifications par callback, en secondes.

int **get_callbackMinDelay()**

Retourne :

un entier représentant l'attente minimale entre deux notifications par callback, en secondes

En cas d'erreur, déclenche une exception ou retourne Y_CALLBACKMINDELAY_INVALID.

network→get_callbackUrl()

YNetwork

network→callbackUrl()

network.get_callbackUrl()

Retourne l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

String **get_callbackUrl()**

Retourne :

une chaîne de caractères représentant l'adresse (URL) de callback à notifier lors de changement d'état significatifs

En cas d'erreur, déclenche une exception ou retourne Y_CALLBACKURL_INVALID.

network→get_discoverable()**YNetwork****network→discoverable()****network.get_discoverable()**

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

int get_discoverable()**Retourne :**

soit Y_DISCOVERABLE_FALSE, soit Y_DISCOVERABLE_TRUE, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour)

En cas d'erreur, déclenche une exception ou retourne Y_DISCOVERABLE_INVALID.

network→get_errorMessage()

YNetwork

network→errorMessage()

network.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

String get_errorMessage()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau.

network→**get_errorType()****YNetwork****network**→**errorType()****network.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau.

int **get_errorType()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau.

network→get_friendlyName()**YNetwork****network→friendlyName()****network.get_friendlyName()**

Retourne un identifiant global de l'interface réseau au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et de l'interface réseau si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'interface réseau (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'interface réseau en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

network→**get_functionDescriptor()****YNetwork****network**→**functionDescriptor()****network.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

network→**get_functionId()**

YNetwork

network→**functionId()****network.get_functionId()**

Retourne l'identifiant matériel de l'interface réseau, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant l'interface réseau (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

network→**get_hardwareId()****YNetwork****network**→**hardwareId()****network.get_hardwareId()**

Retourne l'identifiant matériel unique de l'interface réseau au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'interface réseau (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant l'interface réseau (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

network→**get_ipAddress()**

YNetwork

network→**ipAddress()****network.get_ipAddress()**

Retourne l'adresse IP utilisée par le module Yoctopuce.

String **get_ipAddress()**

Il peut s'agir d'une adresse configurée statiquement, ou d'une adresse reçue par un serveur DHCP.

Retourne :

une chaîne de caractères représentant l'adresse IP utilisée par le module Yoctopuce

En cas d'erreur, déclenche une exception ou retourne Y_IPADDRESS_INVALID.

network→**get_logicalName()**
network→**logicalName()**
network.get_logicalName()

YNetwork

Retourne le nom logique de l'interface réseau.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique de l'interface réseau.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

network→get_macAddress()

YNetwork

network→macAddress()

network.get_macAddress()

Retourne l'adresse MAC de l'interface réseau, unique pour chaque module.

String get_macAddress()

L'adresse MAC est aussi présente sur un autocollant sur le module, représentée en chiffres et en code-barres.

Retourne :

une chaîne de caractères représentant l'adresse MAC de l'interface réseau, unique pour chaque module

En cas d'erreur, déclenche une exception ou retourne Y_MACADDRESS_INVALID.

network→**get_module()****YNetwork****network**→**module()**`network.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

network→**get_poeCurrent()**

YNetwork

network→**poeCurrent()****network.get_poeCurrent()**

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

int **get_poeCurrent()**

La consommation est mesurée après conversion en 5 Volt, et ne doit jamais dépasser 1800 mA.

Retourne :

un entier représentant le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères

En cas d'erreur, déclenche une exception ou retourne Y_POECURRENT_INVALID.

network→get_primaryDNS()
network→primaryDNS()
network.get_primaryDNS()

YNetwork

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

String **get_primaryDNS()**

Retourne :

une chaîne de caractères représentant l'adresse IP du serveur de noms primaire que le module doit utiliser

En cas d'erreur, déclenche une exception ou retourne Y_PRIMARYDNS_INVALID.

network→**get_readiness()****YNetwork****network**→**readiness()****network.get_readiness()**

Retourne l'état de fonctionnement atteint par l'interface réseau.

```
int get_readiness( )
```

Le niveau zéro (DOWN_0) signifie qu'aucun support réseau matériel n'a été détecté. Soit il n'y a pas de signal sur le câble réseau, soit le point d'accès sans fil choisi n'est pas détecté. Le niveau 1 (LIVE_1) est atteint lorsque le réseau est détecté, mais n'est pas encore connecté. Pour un réseau sans fil, cela confirme l'existence du SSID configuré. Le niveau 2 (LINK_2) est atteint lorsque le support matériel du réseau est fonctionnel. Pour une connection réseau filaire, le niveau 2 signifie que le câble est connecté aux deux bouts. Pour une connection à un point d'accès réseau sans fil, il démontre que les paramètres de sécurité configurés sont corrects. Pour une connection sans fil en mode ad-hoc, cela signifie qu'il y a au moins un partenaire sur le réseau ad-hoc. Le niveau 3 (DHCP_3) est atteint lorsque qu'une adresse IP a été obtenue par DHCP. Le niveau 4 (DNS_4) est atteint lorsqu'un serveur DNS est joignable par le réseau. Le niveau 5 (WWW_5) est atteint lorsque la connectivité globale à internet est avérée par l'obtention de l'heure courante sur un serveur NTP.

Retourne :

une valeur parmi Y_READINESS_DOWN, Y_READINESS_EXISTS, Y_READINESS_LINKED, Y_READINESS_LAN_OK et Y_READINESS_WWW_OK représentant l'état de fonctionnement atteint par l'interface réseau

En cas d'erreur, déclenche une exception ou retourne Y_READINESS_INVALID.

network→get_router()**YNetwork****network→router()**`network.get_router()`

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

String `get_router()`

Retourne :

une chaîne de caractères représentant l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*)

En cas d'erreur, déclenche une exception ou retourne `Y_ROUTER_INVALID`.

network→get_secondaryDNS()

YNetwork

network→secondaryDNS()

network.get_secondaryDNS()

Retourne l'adresse IP du serveur de noms secondaire que le module doit utiliser.

String **get_secondaryDNS()**

Retourne :

une chaîne de caractères représentant l'adresse IP du serveur de noms secondaire que le module doit utiliser

En cas d'erreur, déclenche une exception ou retourne Y_SECONDARYDNS_INVALID.

network→get_subnetMask()**YNetwork****network→subnetMask()****network.get_subnetMask()**

Retourne le masque de sous-réseau utilisé par le module.

String **get_subnetMask()**

Retourne :

une chaîne de caractères représentant le masque de sous-réseau utilisé par le module

En cas d'erreur, déclenche une exception ou retourne Y_SUBNETMASK_INVALID.

network→**get_userdata()**

YNetwork

network→**userData()****network.get_userdata()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

network→get_userPassword()**YNetwork****network→userPassword()****network.get_userPassword()**

Retourne une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

String **get_userPassword()**

Retourne :

une chaîne de caractères représentant une chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide

En cas d'erreur, déclenche une exception ou retourne Y_USERPASSWORD_INVALID.

network→**get_wwwWatchdogDelay()**

YNetwork

network→**wwwWatchdogDelay()**

network.get_wwwWatchdogDelay()

Retourne la durée de perte de connection WWW tolérée (en secondes) avant de déclancher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

int **get_wwwWatchdogDelay()**

Une valeur nulle désactive le redémarrage automatique en cas de perte de connectivité WWW.

Retourne :

un entier représentant la durée de perte de connection WWW tolérée (en secondes) avant de déclancher un redémarrage automatique pour tenter de récupérer la connectivité Internet

En cas d'erreur, déclenche une exception ou retourne Y_WWWWATCHDOGDELAY_INVALID.

network→**isOnline()****network.isOnline()****YNetwork**

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache de l'interface réseau sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'interface réseau est joignable, `false` sinon

network→**load()****network.load()****YNetwork**

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**nextNetwork()****network.nextNetwork()****YNetwork**

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

YNetwork **nextNetwork()**

Retourne :

un pointeur sur un objet YNetwork accessible en ligne, ou `null` lorsque l'énumération est terminée.

network→**ping()****network.ping()****YNetwork**

Ping str_host pour vérifier la connexion réseau.

String ping(String host)

Envoie quatre requêtes ICMP ECHO_RESPONDER à la cible str_host depuis le module. Cette méthode retourne une chaîne de caractères avec le résultat des 4 requêtes ICMP ECHO_RESPONSE.

Paramètres :

host le nom d'hôte ou l'adresse IP de la cible

Retourne :

une chaîne de caractères contenant le résultat du ping.

network→registerValueCallback()**YNetwork****network.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

network→**set_adminPassword()****YNetwork****network**→**setAdminPassword()****network.set_adminPassword()**

Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module.

```
int set_adminPassword( String newval)
```

Si la valeur fournie est une chaîne vide, plus aucun mot de passe n'est nécessaire. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute altération de l'état du module

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_callbackCredentials()**YNetwork****network→setCallbackCredentials()****network.set_callbackCredentials()**

Modifie le laisser-passer pour se connecter à l'adresse de callback.

```
int set_callbackCredentials( String newval)
```

Le laisser-passer doit être fourni tel que retourné par la fonction `get_callbackCredentials`, sous la forme `username:hash`. La valeur du hash dépend de la méthode d'autorisation implémentée par le callback. Pour une autorisation de type Basic, le hash est le MD5 de la chaîne `username:password`. Pour une autorisation de type Digest, le hash est le MD5 de la chaîne `username:realm:password`. Pour une utilisation simplifiée, utilisez la fonction `callbackLogin`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le laisser-passer pour se connecter à l'adresse de callback

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_callbackEncoding()****YNetwork****network**→**setCallbackEncoding()****network.set_callbackEncoding()**

Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.

```
int set_callbackEncoding( int newval )
```

Paramètres :

newval une valeur parmi Y_CALLBACKENCODING_FORM, Y_CALLBACKENCODING_JSON, Y_CALLBACKENCODING_JSON_ARRAY, Y_CALLBACKENCODING_CSV et Y_CALLBACKENCODING_YOCTO_API représentant l'encodage à utiliser pour représenter les valeurs notifiées par callback

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_callbackMaxDelay()**YNetwork****network→setCallbackMaxDelay()****network.set_callbackMaxDelay()**

Modifie l'attente maximale entre deux notifications par callback, en secondes.

```
int set_callbackMaxDelay( int newval)
```

Paramètres :

newval un entier représentant l'attente maximale entre deux notifications par callback, en secondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_callbackMethod()****YNetwork****network**→**setCallbackMethod()****network.set_callbackMethod()**

Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.

```
int set_callbackMethod( int newval )
```

Paramètres :

newval une valeur parmi Y_CALLBACKMETHOD_POST, Y_CALLBACKMETHOD_GET et Y_CALLBACKMETHOD_PUT représentant la méthode HTTP à utiliser pour signaler les changements d'état par callback

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_callbackMinDelay()**YNetwork****network→setCallbackMinDelay()****network.set_callbackMinDelay()**

Modifie l'attente minimale entre deux notifications par callback, en secondes.

```
int set_callbackMinDelay( int newval)
```

Paramètres :

newval un entier représentant l'attente minimale entre deux notifications par callback, en secondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_callbackUrl()**

YNetwork

network→**setCallbackUrl()**

network.set_callbackUrl()

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

```
int set_callbackUrl( String newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant l'adresse (URL) de callback à notifier lors de changement d'état significatifs

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_discoverable()****YNetwork****network**→**setDiscoverable()****network.set_discoverable()**

Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour).

```
int set_discoverable( int newval )
```

Paramètres :

newval soit Y_DISCOVERABLE_FALSE, soit Y_DISCOVERABLE_TRUE, selon l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocoles uPnP/Bonjour)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_logicalName()****YNetwork****network**→**setLogicalName()****network.set_logicalName()**

Modifie le nom logique de l'interface réseau.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'interface réseau.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→set_primaryDNS()
network→setPrimaryDNS()
network.set_primaryDNS()

YNetwork

Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.

```
int set_primaryDNS( String newval)
```

En mode DHCP, si une valeur est spécifiée, elle remplacera celle reçue du serveur DHCP. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

newval une chaîne de caractères représentant l'adresse IP du serveur de noms primaire que le module doit utiliser

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_secondaryDNS()****YNetwork****network**→**setSecondaryDNS()****network.set_secondaryDNS()**

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

```
int set_secondaryDNS( String newval)
```

En mode DHCP, si une valeur est spécifiée, elle remplacera celle reçue du serveur DHCP. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

newval une chaîne de caractères représentant l'adresse IP du serveur de nom secondaire que le module doit utiliser

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_userdata()****YNetwork****network**→**setUserData()****network.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

network→**set_userPassword()****YNetwork****network**→**setUserPassword()****network.set_userPassword()**

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

```
int set_userPassword( String newval)
```

Si la valeur fournie est une chaîne vide, plus aucun mot de passe n'est nécessaire. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→**set_wwwWatchdogDelay()**
network→**setWwwWatchdogDelay()**
network.set_wwwWatchdogDelay()

YNetwork

Modifie la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

```
int set_wwwWatchdogDelay( int newval)
```

Une valeur nulle désactive le redémarrage automatique en cas de perte de connectivité WWW. La plus petite durée non-nulle utilisable est 90 secondes.

Paramètres :

newval un entier représentant la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→useDHCP()**network.useDHCP()****YNetwork**

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

```
int useDHCP( String fallbackIpAddr,  
             int fallbackSubnetMaskLen,  
             String fallbackRouter)
```

En attendant qu'une adresse soit reçue (et indéfiniment si aucun serveur DHCP ne répond), le module utilisera les paramètres IP spécifiés à cette fonction. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

fallbackIpAddr	adresse IP à utiliser si aucun serveur DHCP ne répond
fallbackSubnetMaskLen	longueur du masque de sous-réseau à utiliser si aucun serveur DHCP ne répond. Par exemple, la valeur 24 représente 255.255.255.0.
fallbackRouter	adresse de la passerelle à utiliser si aucun serveur DHCP ne répond

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

network→useStaticIP()`network.useStaticIP()`**YNetwork**

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

```
int useStaticIP( String ipAddress,  
                int subnetMaskLen,  
                String router)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

ipAddress	adresse IP à utiliser par le module
subnetMaskLen	longueur du masque de sous-réseau à utiliser. Par exemple, la valeur 24 représente 255.255.255.0.
router	adresse IP de la passerelle à utiliser ("default gateway")

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.29. contrôle d'OS

L'objet `OsControl` permet de contrôler le système d'exploitation sur lequel tourne un VirtualHub. `OsControl` n'est disponible que dans le VirtualHub software. Attention, cette fonctionnalité doit être explicitement activée au lancement du VirtualHub, avec l'option `-o`.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_oscontrol.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YOsControl = yoctolib.YOsControl;</code>
php	<code>require_once('yocto_oscontrol.php');</code>
c++	<code>#include "yocto_oscontrol.h"</code>
m	<code>#import "yocto_oscontrol.h"</code>
pas	<code>uses yocto_oscontrol;</code>
vb	<code>yocto_oscontrol.vb</code>
cs	<code>yocto_oscontrol.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YOsControl;</code>
py	<code>from yocto_oscontrol import *</code>

Fonction globales

yFindOsControl(func)

Permet de retrouver un contrôle d'OS d'après un identifiant donné.

yFirstOsControl()

Commence l'énumération des contrôle d'OS accessibles par la librairie.

Méthodes des objets YOsControl

oscontrol→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'OS au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

oscontrol→get_advertisedValue()

Retourne la valeur courante du contrôle d'OS (pas plus de 6 caractères).

oscontrol→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

oscontrol→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

oscontrol→get_friendlyName()

Retourne un identifiant global du contrôle d'OS au format `NOM_MODULE . NOM_FONCTION`.

oscontrol→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

oscontrol→get_functionId()

Retourne l'identifiant matériel du contrôle d'OS, sans référence au module.

oscontrol→get_hardwareId()

Retourne l'identifiant matériel unique du contrôle d'OS au format `SERIAL . FUNCTIONID`.

oscontrol→get_logicalName()

Retourne le nom logique du contrôle d'OS.

oscontrol→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

oscontrol→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`oscontrol→get_shutdownCountdown()`

Retourne le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé.

`oscontrol→get_userData()`

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

`oscontrol→isOnline()`

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

`oscontrol→isOnline_async(callback, context)`

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

`oscontrol→load(msValidity)`

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

`oscontrol→load_async(msValidity, callback, context)`

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

`oscontrol→nextOsControl()`

Continue l'énumération des contrôle d'OS commencée à l'aide de `yFirstOsControl()`.

`oscontrol→registerValueCallback(callback)`

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

`oscontrol→set_logicalName(newval)`

Modifie le nom logique du contrôle d'OS.

`oscontrol→set_userData(data)`

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

`oscontrol→shutdown(secBeforeShutDown)`

Agende un arrêt de l'OS dans un nombre donné de secondes.

`oscontrol→wait_async(callback, context)`

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YOsControl.FindOsControl()**YOsControl****yFindOsControl()**`YOsControl.FindOsControl()`

Permet de retrouver un contrôle d'OS d'après un identifiant donné.

`YOsControl` **FindOsControl**(`String func`)

L'identifiant peut être spécifié sous plusieurs formes:

- `NomLogiqueFonction`
- `NoSerieModule.IdentifiantFonction`
- `NoSerieModule.NomLogiqueFonction`
- `NomLogiqueModule.IdentifiantMatériel`
- `NomLogiqueModule.NomLogiqueFonction`

Cette fonction n'exige pas que le contrôle d'OS soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YOsControl.isOnline()` pour tester si le contrôle d'OS est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le contrôle d'OS sans ambiguïté

Retourne :

un objet de classe `YOsControl` qui permet ensuite de contrôler le contrôle d'OS.

YOsControl.FirstOsControl()**YOsControl****yFirstOsControl()**`YOsControl.FirstOsControl()`

Commence l'énumération des contrôle d'OS accessibles par la librairie.

`YOsControl FirstOsControl()`

Utiliser la fonction `YOsControl.nextOsControl()` pour itérer sur les autres contrôle d'OS.

Retourne :

un pointeur sur un objet `YOsControl`, correspondant au premier contrôle d'OS accessible en ligne, ou `null` si il n'y a pas de contrôle d'OS disponibles.

oscontrol→describe()`oscontrol.describe()`**YOsControl**

Retourne un court texte décrivant de manière non-ambigüe l'instance du contrôle d'OS au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le contrôle d'OS (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

oscontrol→get_advertisedValue()**YOsControl****oscontrol→advertisedValue()****oscontrol.get_advertisedValue()**

Retourne la valeur courante du contrôle d'OS (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du contrôle d'OS (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

oscontrol→get_errorMessage()

YOsControl

oscontrol→errorMessage()

oscontrol.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

String get_errorMessage()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'OS.

oscontrol→get_errorType()**YOsControl****oscontrol→errorType()****oscontrol.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du contrôle d'OS.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du contrôle d'OS.

oscontrol→get_friendlyName()

YOsControl

oscontrol→friendlyName()

oscontrol.get_friendlyName()

Retourne un identifiant global du contrôle d'OS au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du contrôle d'OS si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du contrôle d'OS (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le contrôle d'OS en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

oscontrol→get_functionDescriptor()**YOsControl****oscontrol→functionDescriptor()****oscontrol.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

oscontrol→**get_functionId()**

YOsControl

oscontrol→**functionId()**

oscontrol.get_functionId()

Retourne l'identifiant matériel du contrôle d'OS, sans référence au module.

String **get_functionId()** ()

Par exemple relay1.

Retourne :

une chaîne de caractères identifiant le contrôle d'OS (ex: relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FUNCTIONID_INVALID.

oscontrol→get_hardwareId()
oscontrol→hardwareId()
oscontrol.get_hardwareId()

YOsControl

Retourne l'identifiant matériel unique du contrôle d'OS au format `SERIAL.FUNCTIONID`.

`String get_hardwareId()`

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du contrôle d'OS (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le contrôle d'OS (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

oscontrol→get_logicalName()

YOsControl

oscontrol→logicalName()

oscontrol.get_logicalName()

Retourne le nom logique du contrôle d'OS.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du contrôle d'OS.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

oscontrol→**get_module()****YOsControl****oscontrol**→**module()**`oscontrol.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

oscontrol→get_shutdownCountdown()

YOsControl

oscontrol→shutdownCountdown()

oscontrol.get_shutdownCountdown()

Retourne le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé.

int get_shutdownCountdown()

Retourne :

un entier représentant le nombre de secondes restantes avant un arrêt de l'OS, ou zéro si aucun arrêt n'a été agendé

En cas d'erreur, déclenche une exception ou retourne Y_SHUTDOWNCOUNTDOWN_INVALID.

oscontrol→**get_userdata()****YOsControl****oscontrol**→**userData()****oscontrol.userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

oscontrol→isOnline()`oscontrol.isOnline()`

YOsControl

Vérifie si le module hébergeant le contrôle d'OS est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du contrôle d'OS sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le contrôle d'OS est joignable, `false` sinon

oscontrol→load()`oscontrol.load()`**YOsControl**

Met en cache les valeurs courantes du contrôle d'OS, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

oscontrol→**nextOsControl()**

YOsControl

oscontrol.nextOsControl()

Continue l'énumération des contrôle d'OS commencée à l'aide de `yFirstOsControl()`.

YOsControl **nextOsControl()**

Retourne :

un pointeur sur un objet `YOsControl` accessible en ligne, ou `null` lorsque l'énumération est terminée.

oscontrol→registerValueCallback()**YOsControl****oscontrol.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

oscontrol→**set_logicalName()**

YOsControl

oscontrol→**setLogicalName()**

oscontrol.set_logicalName()

Modifie le nom logique du contrôle d'OS.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du contrôle d'OS.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

oscontrol→set_userdata()**YOsControl****oscontrol→setUserData()****oscontrol.set_userdata()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

```
void set_userdata( Object data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

oscontrol→**shutdown()**`oscontrol.shutdown()`**YOsControl**

Agende un arrêt de l'OS dans un nombre donné de secondes.

```
int shutdown( int secBeforeShutDown)
```

Paramètres :

secBeforeShutDown nombre de secondes avant l'arrêt

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.30. Interface de la fonction Power

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_power.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YPower = yoctolib.YPower;</code>
php	<code>require_once('yocto_power.php');</code>
c++	<code>#include "yocto_power.h"</code>
m	<code>#import "yocto_power.h"</code>
pas	<code>uses yocto_power;</code>
vb	<code>yocto_power.vb</code>
cs	<code>yocto_power.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YPower;</code>
py	<code>from yocto_power import *</code>

Fonction globales

yFindPower(func)

Permet de retrouver un capteur de puissance électrique d'après un identifiant donné.

yFirstPower()

Commence l'énumération des capteurs de puissance électrique accessibles par la librairie.

Méthodes des objets YPower

power→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

power→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de puissance électrique au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

power→get_advertisedValue()

Retourne la valeur courante du capteur de puissance électrique (pas plus de 6 caractères).

power→get_cosPhi()

Retourne le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA).

power→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Watt, sous forme de nombre à virgule.

power→get_currentValue()

Retourne la valeur actuelle de la puissance électrique, en Watt, sous forme de nombre à virgule.

power→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

power→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

power→get_friendlyName()

Retourne un identifiant global du capteur de puissance électrique au format `NOM_MODULE . NOM_FONCTION`.

power→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

power→get_functionId()

Retourne l'identifiant matériel du capteur de puissance électrique, sans référence au module.

power→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de puissance électrique au format `SERIAL.FUNCTIONID`.

power→get_highestValue()

Retourne la valeur maximale observée pour la puissance électrique depuis le démarrage du module.

power→get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

power→get_logicalName()

Retourne le nom logique du capteur de puissance électrique.

power→get_lowestValue()

Retourne la valeur minimale observée pour la puissance électrique depuis le démarrage du module.

power→get_meter()

Retourne la valeur actuelle du compteur d'énergie, calculée par le wattmètre en intégrant la consommation instantanée.

power→get_meterTimer()

Retourne le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

power→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

power→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

power→get_recordedData(startTime, endTime)

Retourne un objet `DataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du `DataLogger`, pour l'intervalle de temps spécifié.

power→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

power→get_resolution()

Retourne la résolution des valeurs mesurées.

power→get_unit()

Retourne l'unité dans laquelle la puissance électrique est exprimée.

power→get_userData()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

power→isOnline()

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

power→isOnline_async(callback, context)

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

power→load(msValidity)

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

power→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

power→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

power→nextPower()

Continue l'énumération des capteurs de puissance électrique commencée à l'aide de `yFirstPower()`.

power→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

power→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

power→reset()

Réinitialise le compteur d'énergie.

power→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

power→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

power→set_logicalName(newval)

Modifie le nom logique du capteur de puissance électrique.

power→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

power→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

power→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

power→set_userData(data)

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

power→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YPower.FindPower()**YPower****yFindPower()**`YPower.FindPower()`

Permet de retrouver un capteur de puissance électrique d'après un identifiant donné.

`YPower` **FindPower**(`String` **func**)

L'identifiant peut être spécifié sous plusieurs formes:

- `NomLogiqueFonction`
- `NoSerieModule.IdentifiantFonction`
- `NoSerieModule.NomLogiqueFonction`
- `NomLogiqueModule.IdentifiantMatériel`
- `NomLogiqueModule.NomLogiqueFonction`

Cette fonction n'exige pas que le capteur de puissance électrique soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPower.isOnline()` pour tester si le capteur de puissance électrique est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de puissance électrique sans ambiguïté

Retourne :

un objet de classe `YPower` qui permet ensuite de contrôler le capteur de puissance électrique.

YPower.FirstPower()**YPower****yFirstPower()**`YPower.FirstPower()`

Commence l'énumération des capteurs de puissance électrique accessibles par la librairie.

`YPower FirstPower()`

Utiliser la fonction `YPower.nextPower()` pour itérer sur les autres capteurs de puissance électrique.

Retourne :

un pointeur sur un objet `YPower`, correspondant au premier capteur de puissance électrique accessible en ligne, ou `null` si il n'y a pas de capteurs de puissance électrique disponibles.

power→calibrateFromPoints()**YPower****power.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                        ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→describe()`power.describe()`**YPower**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de puissance électrique au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String `describe()`

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le capteur de puissance électrique (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

power→**get_advertisedValue()**

YPower

power→**advertisedValue()**

power.get_advertisedValue()

Retourne la valeur courante du capteur de puissance électrique (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de puissance électrique (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

power→**get_cosPhi()****YPower****power**→**cosPhi()****power.get_cosPhi()**

Retourne le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA).

`double get_cosPhi()`**Retourne :**

une valeur numérique représentant le facteur de puissance (rapport entre la puissance réelle consommée, en W, et la puissance apparente fournie, en VA)

En cas d'erreur, déclenche une exception ou retourne `Y_COSPHI_INVALID`.

power→**get_currentRawValue()**

YPower

power→**currentRawValue()**

power.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Watt, sous forme de nombre à virgule.

double **get_currentRawValue()**

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en Watt, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

power→**get_currentValue()****YPower****power**→**currentValue()****power.get_currentValue()**

Retourne la valeur actuelle de la puissance électrique, en Watt, sous forme de nombre à virgule.

```
double get_currentValue( )
```

Retourne :

une valeur numérique représentant la valeur actuelle de la puissance électrique, en Watt, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

power→**get_errorMessage()**

YPower

power→**errorMessage()**

power.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de puissance électrique.

power→**get_errorType()****YPower****power**→**errorType()****power.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de puissance électrique.

int **get_errorType()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de puissance électrique.

power→**get_friendlyName()****YPower****power**→**friendlyName()****power.get_friendlyName()**

Retourne un identifiant global du capteur de puissance électrique au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du capteur de puissance électrique si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de puissance électrique (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de puissance électrique en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

power→**get_functionDescriptor()****YPower****power**→**functionDescriptor()****power.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

power→**get_functionId()**

YPower

power→**functionId()**`power.get_functionId()`

Retourne l'identifiant matériel du capteur de puissance électrique, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de puissance électrique (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

power→**get_hardwareId()****YPower****power**→**hardwareId()****power.get_hardwareId()**

Retourne l'identifiant matériel unique du capteur de puissance électrique au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de puissance électrique (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur de puissance électrique (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

power→**get_highestValue()**

YPower

power→**highestValue()**`power.get_highestValue()`

Retourne la valeur maximale observée pour la puissance électrique depuis le démarrage du module.

double **get_highestValue()**

Retourne :

une valeur numérique représentant la valeur maximale observée pour la puissance électrique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

power→**get_logFrequency()****YPower****power**→**logFrequency()****power.get_logFrequency()**

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.

String **get_logFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

power→**get_logicalName()**

YPower

power→**logicalName()**`power.get_logicalName()`

Retourne le nom logique du capteur de puissance electrique.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du capteur de puissance electrique.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

power→**get_lowestValue()****YPower****power**→**lowestValue()**`power.get_lowestValue()`

Retourne la valeur minimale observée pour la puissance électrique depuis le démarrage du module.

double **get_lowestValue()**

Retourne :

une valeur numérique représentant la valeur minimale observée pour la puissance électrique depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

power→**get_meter()**

YPower

power→**meter()**`power.get_meter()`

Retourne la valeur actuelle du compteur d'energie, calculée par le wattmètre en intégrant la consommation instantanée.

double **get_meter()**

Ce compteur est réinitialisé à chaque démarrage du module.

Retourne :

une valeur numérique représentant la valeur actuelle du compteur d'energie, calculée par le wattmètre en intégrant la consommation instantanée

En cas d'erreur, déclenche une exception ou retourne Y_METER_INVALID.

power→**get_meterTimer()****YPower****power**→**meterTimer()**`power.get_meterTimer()`

Retourne le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

`int get_meterTimer()`**Retourne :**

un entier représentant le temps écoulé depuis la dernière initialisation du compteur d'énergie, en secondes

En cas d'erreur, déclenche une exception ou retourne Y_METERTIMER_INVALID.

power→**get_module()**

YPower

power→**module()**`power.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

power→**get_recordedData()****YPower****power**→**recordedData()****power.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

YDataSet **get_recordedData(** long **startTime**, long **endTime****)**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

power→**get_reportFrequency()**

YPower

power→**reportFrequency()**

power.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String **get_reportFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

power→**get_resolution()****YPower****power**→**resolution()**`power.get_resolution()`

Retourne la résolution des valeurs mesurées.

`double` **get_resolution()** ()

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

power→**get_unit()**

YPower

power→**unit()**`power.get_unit()`

Retourne l'unité dans laquelle la puissance électrique est exprimée.

String **get_unit()**

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la puissance électrique est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

power→**get_userdata()****YPower****power**→**userData()**`power.get_userdata()`

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

power→**isOnline()****power.isOnline()**

YPower

Vérifie si le module hébergeant le capteur de puissance électrique est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du capteur de puissance électrique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

true si le capteur de puissance électrique est joignable, false sinon

power→**load()**`power.load()`**YPower**

Met en cache les valeurs courantes du capteur de puissance électrique, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→loadCalibrationPoints()**YPower****power.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**nextPower()****power.nextPower()****YPower**

Continue l'énumération des capteurs de puissance electrique commencée à l'aide de `yFirstPower()`.

`YPower` **nextPower()**

Retourne :

un pointeur sur un objet `YPower` accessible en ligne, ou `null` lorsque l'énumération est terminée.

power→registerTimedReportCallback()**YPower****power.registerTimedReportCallback()**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

power→registerValueCallback()**YPower****power.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

power→**reset()**`power.reset ()`

YPower

Réinitialise le compteur d'énergie.

`int reset()`

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**set_highestValue()****YPower****power**→**setHighestValue()****power.set_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**set_logFrequency()****YPower****power**→**setLogFrequency()****power.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**set_logicalName()****YPower****power**→**setLogicalName()****power.set_logicalName()**

Modifie le nom logique du capteur de puissance électrique.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de puissance électrique.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**set_lowestValue()**

YPower

power→**setLowestValue()**

power.set_lowestValue()

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**set_reportFrequency()**
power→**setReportFrequency()**
power.set_reportFrequency()

YPower

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**set_resolution()****YPower****power**→**setResolution()**`power.set_resolution()`

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

power→**set_userdata()****YPower****power**→**setUserData()**`power.set_userdata()`

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

3.31. Interface de la fonction Pressure

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_pressure.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YPressure = yoctolib.YPressure;
php	require_once('yocto_pressure.php');
c++	#include "yocto_pressure.h"
m	#import "yocto_pressure.h"
pas	uses yocto_pressure;
vb	yocto_pressure.vb
cs	yocto_pressure.cs
java	import com.yoctopuce.YoctoAPI.YPressure;
py	from yocto_pressure import *

Fonction globales

yFindPressure(func)

Permet de retrouver un capteur de pression d'après un identifiant donné.

yFirstPressure()

Commence l'énumération des capteurs de pression accessibles par la librairie.

Méthodes des objets YPressure

pressure→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

pressure→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de pression au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

pressure→get_advertisedValue()

Retourne la valeur courante du capteur de pression (pas plus de 6 caractères).

pressure→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), 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)

3. Reference

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

pressure→**set_logicalName(newval)**

Modifie le nom logique du capteur de pression.

pressure→**set_lowestValue(newval)**

Modifie la mémoire de valeur minimale observée.

pressure→**set_reportFrequency(newval)**

Modifie la fréquence de notification périodique des valeurs mesurées.

pressure→**set_resolution(newval)**

Modifie la résolution des valeurs physique mesurées.

pressure→**set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

pressure→**wait_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YPressure.FindPressure()**YPressure****yFindPressure()**`YPressure.FindPressure()`

Permet de retrouver un capteur de pression d'après un identifiant donné.

`YPressure` **FindPressure**(String **func**)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de pression soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPressure.isOnline()` pour tester si le capteur de pression est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de pression sans ambiguïté

Retourne :

un objet de classe `YPressure` qui permet ensuite de contrôler le capteur de pression.

YPressure.FirstPressure()

YPressure

yFirstPressure()**YPressure.FirstPressure()**

Commence l'énumération des capteurs de pression accessibles par la librairie.

YPressure **FirstPressure()**

Utiliser la fonction `YPressure.nextPressure()` pour itérer sur les autres capteurs de pression.

Retourne :

un pointeur sur un objet `YPressure`, correspondant au premier capteur de pression accessible en ligne, ou `null` si il n'y a pas de capteurs de pression disponibles.

pressure→calibrateFromPoints()**YPressure****pressure.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                        ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**describe()****pressure.describe()****YPressure**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de pression au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le capteur de pression (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

pressure→**get_advertisedValue()****YPressure****pressure**→**advertisedValue()****pressure.get_advertisedValue()**

Retourne la valeur courante du capteur de pression (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de pression (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

pressure→**get_currentRawValue()**

YPressure

pressure→**currentRawValue()**

pressure.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en millibar (hPa), sous forme de nombre à virgule.

double **get_currentRawValue()**

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en millibar (hPa), sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

pressure→**get_currentValue()****YPressure****pressure**→**currentValue()****pressure.get_currentValue()**

Retourne la valeur actuelle de la pression, en millibar (hPa), sous forme de nombre à virgule.

double **get_currentValue()**

Retourne :

une valeur numérique représentant la valeur actuelle de la pression, en millibar (hPa), sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

pressure→**get_errorMessage()**

YPressure

pressure→**errorMessage()**

pressure.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de pression.

pressure→**get_errorType()****YPressure****pressure**→**errorType()**`pressure.get_errorType()`

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de pression.

`int` **get_errorType()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de pression.

pressure→**get_friendlyName()**

YPressure

pressure→**friendlyName()**

pressure.get_friendlyName()

Retourne un identifiant global du capteur de pression au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du capteur de pression si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de pression (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de pression en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

pressure→**get_functionDescriptor()****YPressure****pressure**→**functionDescriptor()****pressure.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

pressure→**get_functionId()**

YPressure

pressure→**functionId()**

pressure.get_functionId()

Retourne l'identifiant matériel du capteur de pression, sans référence au module.

String **get_functionId()** ()

Par exemple relay1.

Retourne :

une chaîne de caractères identifiant le capteur de pression (ex: relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FUNCTIONID_INVALID.

pressure→**get_hardwareId()****YPressure****pressure**→**hardwareId()****pressure.get_hardwareId()**

Retourne l'identifiant matériel unique du capteur de pression au format `SERIAL.FUNCTIONID`.

`String` **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de pression (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur de pression (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

pressure→**get_highestValue()**

YPressure

pressure→**highestValue()**

pressure.get_highestValue()

Retourne la valeur maximale observée pour la pression depuis le démarrage du module.

double **get_highestValue()**

Retourne :

une valeur numérique représentant la valeur maximale observée pour la pression depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

pressure→**get_logFrequency()****YPressure****pressure**→**logFrequency()****pressure.get_logFrequency()**

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.

String **get_logFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

pressure→**get_logicalName()**

YPressure

pressure→**logicalName()**

pressure.get_logicalName()

Retourne le nom logique du capteur de pression.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du capteur de pression.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

pressure→**get_lowestValue()****YPressure****pressure**→**lowestValue()****pressure.get_lowestValue()**

Retourne la valeur minimale observée pour la pression depuis le démarrage du module.

double **get_lowestValue()**

Retourne :

une valeur numérique représentant la valeur minimale observée pour la pression depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_LOWESTVALUE_INVALID.

pressure→**get_module()**

YPressure

pressure→**module()**`pressure.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

pressure→**get_recordedData()****YPressure****pressure**→**recordedData()****pressure.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

YDataSet **get_recordedData**(long **startTime**, long **endTime**)

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

pressure→**get_reportFrequency()**

YPressure

pressure→**reportFrequency()**

pressure.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String **get_reportFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

pressure→**get_resolution()**
pressure→**resolution()**
pressure.get_resolution()

YPressure

Retourne la résolution des valeurs mesurées.

```
double get_resolution()
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_RESOLUTION_INVALID.

pressure→**get_unit()**

YPressure

pressure→**unit()**`pressure.get_unit()`

Retourne l'unité dans laquelle la pression est exprimée.

String **get_unit()**

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la pression est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

pressure→**get_userData()****YPressure****pressure**→**userData()****pressure.get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

Object **get_userData()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

pressure→**isOnline()****pressure.isOnline()**

YPressure

Vérifie si le module hébergeant le capteur de pression est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache du capteur de pression sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur de pression est joignable, `false` sinon

pressure→**load()****pressure.load()****YPressure**

Met en cache les valeurs courantes du capteur de pression, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→loadCalibrationPoints()**YPressure****pressure.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**nextPressure()****YPressure****pressure.nextPressure()**

Continue l'énumération des capteurs de pression commencée à l'aide de `yFirstPressure()`.

`YPressure` **nextPressure()**

Retourne :

un pointeur sur un objet `YPressure` accessible en ligne, ou `null` lorsque l'énumération est terminée.

pressure→registerTimedReportCallback()**YPressure****pressure.registerTimedReportCallback()**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

pressure→registerValueCallback()**YPressure****pressure.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

pressure→**set_highestValue()**

YPressure

pressure→**setHighestValue()**

pressure.set_highestValue()

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**set_logFrequency()**
pressure→**setLogFrequency()**
pressure.set_logFrequency()

YPressure

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**set_logicalName()**

YPressure

pressure→**setLogicalName()**

pressure.set_logicalName()

Modifie le nom logique du capteur de pression.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de pression.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**set_lowestValue()**
pressure→**setLowestValue()**
pressure.set_lowestValue()

YPressure

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**set_reportFrequency()**
pressure→**setReportFrequency()**
pressure.set_reportFrequency()

YPressure

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**set_resolution()**
pressure→**setResolution()**
pressure.set_resolution()

YPressure

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval )
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pressure→**set_userdata()**

YPressure

pressure→**setUserData()**

pressure.set_userdata()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

void **set_userdata**(Object **data**)

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

3.32. Interface de la fonction PwmlInput

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_pwmlinput.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib'); var YPwmInput = yoctolib.YPwmInput;</code>
php	<code>require_once('yocto_pwmlinput.php');</code>
c++	<code>#include "yocto_pwmlinput.h"</code>
m	<code>#import "yocto_pwmlinput.h"</code>
pas	<code>uses yocto_pwmlinput;</code>
vb	<code>yocto_pwmlinput.vb</code>
cs	<code>yocto_pwmlinput.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YPwmInput;</code>
py	<code>from yocto_pwmlinput import *</code>

Fonction globales

yFindPwmlInput(func)

Permet de retrouver un capteur de tension d'après un identifiant donné.

yFirstPwmlInput()

Commence l'énumération des capteurs de tension accessibles par la librairie.

Méthodes des objets YPwmInput

pwmlinput→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.

pwmlinput→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

pwmlinput→get_advertisedValue()

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

pwmlinput→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule.

pwmlinput→get_currentValue()

Retourne la valeur courante de la fonctionnalité PwmlInput, sous forme de nombre à virgule.

pwmlinput→get_dutyCycle()

Retourne le duty cycle du PWM, en pour cents.

pwmlinput→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

pwmlinput→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

pwmlinput→get_frequency()

Retourne la fréquence du PWM en Hz.

pwmlinput→get_friendlyName()

Retourne un identifiant global du capteur de tension au format `NOM_MODULE . NOM_FONCTION`.

pwmlinput→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.

YPwmInput.FindPwmInput()**YPwmInput****yFindPwmInput()**`YPwmInput.FindPwmInput ()`

Permet de retrouver un capteur de tension d'après un identifiant donné.

`YPwmInput` **FindPwmInput**(`String func`)

L'identifiant peut être spécifié sous plusieurs formes:

- `NomLogiqueFonction`
- `NoSerieModule.IdentifiantFonction`
- `NoSerieModule.NomLogiqueFonction`
- `NomLogiqueModule.IdentifiantMatériel`
- `NomLogiqueModule.NomLogiqueFonction`

Cette fonction n'exige pas que le capteur de tension soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmInput.isOnline()` pour tester si le capteur de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de tension sans ambiguïté

Retourne :

un objet de classe `YPwmInput` qui permet ensuite de contrôler le capteur de tension.

YPwmInput.FirstPwmInput()**YPwmInput****yFirstPwmInput()**`YPwmInput.FirstPwmInput ()`

Commence l'énumération des capteurs de tension accessibles par la librairie.

`YPwmInput FirstPwmInput()`

Utiliser la fonction `YPwmInput.nextPwmInput ()` pour itérer sur les autres capteurs de tension.

Retourne :

un pointeur sur un objet `YPwmInput`, correspondant au premier capteur de tension accessible en ligne, ou `null` si il n'y a pas de capteurs de tension disponibles.

pwminput→calibrateFromPoints()**YPwmInput****pwminput.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→describe()`pwminput.describe()`**YPwmInput**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String `describe()`

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le capteur de tension (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

pwminput→get_advertisedValue()

YPwmInput

pwminput→advertisedValue()

pwminput.get_advertisedValue()

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de tension (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

pwminput→get_currentRawValue()**YPwmInput****pwminput→currentRawValue()****pwminput.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule.

```
double get_currentRawValue( )
```

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

pwminput→get_currentValue()

YPwmInput

pwminput→currentValue()

pwminput.get_currentValue()

Retourne la valeur courante de la fonctionnalité PwmInput, sous forme de nombre à virgule.

`double get_currentValue()`

En fonction du réglage pwmReportMode, cela peut être soit la fréquence en Hz, le duty cycle en % ou encore la longueur d'impulsion en ms.

Retourne :

une valeur numérique représentant la valeur courante de la fonctionnalité PwmInput, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

pwminput→**get_dutyCycle()****YPwmInput****pwminput**→**dutyCycle()****pwminput.get_dutyCycle()**

Retourne le duty cycle du PWM, en pour cents.

```
double get_dutyCycle() 
```

Retourne :

une valeur numérique représentant le duty cycle du PWM, en pour cents

En cas d'erreur, déclenche une exception ou retourne Y_DUTYCYCLE_INVALID.

pwminput→get_errorMessage()

YPwmInput

pwminput→errorMessage()

pwminput.errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

String get_errorMessage()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de tension.

pwminput→get_errorType()**YPwmInput****pwminput→errorType()****pwminput.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de tension.

pwminput→get_frequency()

YPwmInput

pwminput→frequency()

pwminput.get_frequency()

Retourne la fréquence du PWM en Hz.

double **get_frequency()** ()

Retourne :

une valeur numérique représentant la fréquence du PWM en Hz

En cas d'erreur, déclenche une exception ou retourne Y_FREQUENCY_INVALID.

pwminput→get_friendlyName()**YPwmInput****pwminput→friendlyName()****pwminput.get_friendlyName()**

Retourne un identifiant global du capteur de tension au format `NOM_MODULE.NOM_FONCTION`.

String `get_friendlyName()`

Le chaîne retournée utilise soit les noms logiques du module et du capteur de tension si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de tension (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de tension en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

pwminput→get_functionDescriptor()

YPwmInput

pwminput→functionDescriptor()

pwminput.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

pwminput→get_functionId()**YPwmInput****pwminput→functionId()****pwminput.get_functionId()**

Retourne l'identifiant matériel du capteur de tension, sans référence au module.

String **get_functionId()** ()

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de tension (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

pwminput→get_hardwareId()

YPwmInput

pwminput→hardwareId()

pwminput.get_hardwareId()

Retourne l'identifiant matériel unique du capteur de tension au format `SERIAL.FUNCTIONID`.

String get_hardwareId()

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de tension (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur de tension (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

pwminput→get_highestValue()**YPwmInput****pwminput→highestValue()****pwminput.get_highestValue()**

Retourne la valeur maximale observée pour la tension depuis le démarrage du module.

double **get_highestValue()**

Retourne :

une valeur numérique représentant la valeur maximale observée pour la tension depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

pwminput→get_logFrequency()

YPwmInput

pwminput→logFrequency()

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.

String get_logFrequency()

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

pwminput→**get_logicalName()****YPwmInput****pwminput**→**logicalName()****pwminput.get_logicalName()**

Retourne le nom logique du capteur de tension.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du capteur de tension.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

pwminput→**get_lowestValue()**

YPwmInput

pwminput→**lowestValue()**

pwminput.get_lowestValue()

Retourne la valeur minimale observée pour la tension depuis le démarrage du module.

double **get_lowestValue()** ()

Retourne :

une valeur numérique représentant la valeur minimale observée pour la tension depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_LOWESTVALUE_INVALID.

pwminput→**get_module()****YPwmInput****pwminput**→**module()**`pwminput.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

pwminput→**get_period()**

YPwmInput

pwminput→**period()**`pwminput.get_period()`

Retourne la période du PWM en millisecondes.

double **get_period()** ()

Retourne :

une valeur numérique représentant la période du PWM en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y_PERIOD_INVALID.

pwminput→get_pulseCounter()
pwminput→pulseCounter()
pwminput.get_pulseCounter()

YPwmInput

Retourne la valeur du compteur d'impulsions.

long get_pulseCounter()

Ce compteur est en réalité incrémenté deux fois par période. Ce compteur est limité à 1 milliard.

Retourne :

un entier représentant la valeur du compteur d'impulsions

En cas d'erreur, déclenche une exception ou retourne Y_PULSECOUNTER_INVALID.

pwminput→get_pulseDuration()

YPwmInput

pwminput→pulseDuration()

pwminput.get_pulseDuration()

Retourne la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule.

double get_pulseDuration()

Retourne :

une valeur numérique représentant la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_PULSEDURATION_INVALID.

pwminput→get_pulseTimer()**YPwmInput****pwminput→pulseTimer()****pwminput.get_pulseTimer()**

Retourne le timer du compteur d'impulsions (ms)

```
long get_pulseTimer( )
```

Retourne :

un entier représentant le timer du compteur d'impulsions (ms)

En cas d'erreur, déclenche une exception ou retourne Y_PULSETIMER_INVALID.

pwminput→get_pwmReportMode()**YPwmInput****pwminput→pwmReportMode()****pwminput.get_pwmReportMode()**

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.

int get_pwmReportMode()**Retourne :**

une valeur parmi Y_PWMREPORTMODE_PWM_DUTYCYCLE, Y_PWMREPORTMODE_PWM_FREQUENCY, Y_PWMREPORTMODE_PWM_PULSEDURATION et Y_PWMREPORTMODE_PWM_EDGECOUNT représentant le type de paramètre (fréquence, duty cycle , longueur d'impulsion ou nombre de changement d'état) renvoyé par la fonction get_currentValue et les callback

En cas d'erreur, déclenche une exception ou retourne Y_PWMREPORTMODE_INVALID.

pwminput→get_recordedData()**YPwmInput****pwminput→recordedData()****pwminput.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

YDataSet **get_recordedData**(long **startTime**, long **endTime**)

Veuillez vous référer à la documentation de la classe DataSet pour plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

pwminput→get_reportFrequency()

YPwmInput

pwminput→reportFrequency()

pwminput.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String **get_reportFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

pwminput→**get_resolution()****YPwmInput****pwminput**→**resolution()****pwminput.get_resolution()**

Retourne la résolution des valeurs mesurées.

```
double get_resolution()
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_RESOLUTION_INVALID.

pwminput→get_unit()

YPwmInput

pwminput→unit()`pwminput.get_unit()`

Retourne l'unité dans laquelle la valeur retournée par `get_currentValue` et les callback est exprimée.

String **get_unit()** ()

Cette unité dépend du réglage `pwmReportMode`.

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la valeur retournée par `get_currentValue` et les callback est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

pwminput→get_userdata()**YPwmInput****pwminput→userData()**`pwminput.getUserData()`

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

Object `get_userdata()`

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

pwminput→**isOnline()**`pwminput.isOnline()`

YPwmInput

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache du capteur de tension sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur de tension est joignable, `false` sinon

pwminput→**load()**`pwminput.load()`**YPwmInput**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→loadCalibrationPoints()**YPwmInput****pwminput.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→**nextPwmInput()****YPwmInput****pwminput.nextPwmInput()**

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstPwmInput()`.

`YPwmInput nextPwmInput()`

Retourne :

un pointeur sur un objet `YPwmInput` accessible en ligne, ou `null` lorsque l'énumération est terminée.

pwminput→registerTimedReportCallback()**YPwmInput****pwminput.registerTimedReportCallback()**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

pwminput→registerValueCallback()**YPwmInput****pwminput.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

pwminput→resetCounter()

YPwmInput

pwminput.resetCounter()

réinitialise le compteur d'impulsions et son timer

int resetCounter()

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→**set_highestValue()****YPwmInput****pwminput**→**setHighestValue()****pwminput.set_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→**set_logFrequency()****YPwmInput****pwminput**→**setLogFrequency()****pwminput.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→**set_logicalName()****YPwmInput****pwminput**→**setLogicalName()****pwminput.set_logicalName()**

Modifie le nom logique du capteur de tension.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de tension.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→**set_lowestValue()**

YPwmInput

pwminput→**setLowestValue()**

pwminput.set_lowestValue()

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→set_pwmReportMode()**YPwmInput****pwminput→setPwmReportMode()****pwminput.set_pwmReportMode()**

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.

```
int set_pwmReportMode( int newval)
```

Seule les six digit de droite du nombre de changement d'état sont transmis, pour les valeurs plus grandes que un million, utiliser get_pulseCounter().

Paramètres :

newval une valeur parmi Y_PWMREPORTMODE_PWM_DUTYCYCLE,
Y_PWMREPORTMODE_PWM_FREQUENCY,
Y_PWMREPORTMODE_PWM_PULSEDURATION et
Y_PWMREPORTMODE_PWM_EDGECOUNT

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→set_reportFrequency()**YPwmInput****pwminput→setReportFrequency()****pwminput.set_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→**set_resolution()**
pwminput→**setResolution()**
pwminput.set_resolution()

YPwmInput

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwminput→set_userdata()

YPwmInput

pwminput→setUserData()

pwminput.set_userdata()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

void **set_userdata**(Object **data**)

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

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');
c++	#include "yocto_pwmoutput.h"
m	#import "yocto_pwmoutput.h"
pas	uses yocto_pwmoutput;
vb	yocto_pwmoutput.vb
cs	yocto_pwmoutput.cs
java	import com.yoctopuce.YoctoAPI.YPwmOutput;
py	from yocto_pwmoutput import *

Fonction globales

yFindPwmOutput(func)

Permet de retrouver un PWM d'après un identifiant donné.

yFirstPwmOutput()

Commence l'énumération des PWM accessibles par la librairie.

Méthodes des objets YPwmOutput

pwmoutput→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du PWM au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

pwmoutput→dutyCycleMove(target, ms_duration)

Déclenche une variation progressive de la longueur des impulsions vers une valeur donnée.

pwmoutput→get_advertisedValue()

Retourne la valeur courante du PWM (pas plus de 6 caractères).

pwmoutput→get_dutyCycle()

Retourne le duty cycle du PWM, en pour cents.

pwmoutput→get_dutyCycleAtPowerOn()

Retourne le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

pwmoutput→get_enabled()

Retourne l'état de fonctionnement du PWM.

pwmoutput→get_enabledAtPowerOn()

Retourne l'état de fonctionnement du PWM à la mise sous tension du module.

pwmoutput→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

pwmoutput→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

pwmoutput→get_frequency()

Retourne la fréquence du PWM en Hz.

pwmoutput→get_friendlyName()

Retourne un identifiant global du PWM au format `NOM_MODULE . NOM_FONCTION`.

pwmoutput→get_functionDescriptor()

	Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
pwmoutput→get_functionId()	Retourne l'identifiant matériel du PWM, sans référence au module.
pwmoutput→get_hardwareId()	Retourne l'identifiant matériel unique du PWM au format SERIAL . FUNCTIONID.
pwmoutput→get_logicalName()	Retourne le nom logique du PWM.
pwmoutput→get_module()	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
pwmoutput→get_module_async(callback, context)	Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
pwmoutput→get_period()	Retourne la période du PWM en millisecondes.
pwmoutput→get_pulseDuration()	Retourne la longueur d'une impulsion du PWM en millisecondes, 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.

YPwmOutput.FindPwmOutput()**YPwmOutput****yFindPwmOutput()**`YPwmOutput.FindPwmOutput()`

Permet de retrouver un PWM d'après un identifiant donné.

`YPwmOutput` **FindPwmOutput**(`String func`)

L'identifiant peut être spécifié sous plusieurs formes:

- `NomLogiqueFonction`
- `NoSerieModule.IdentifiantFonction`
- `NoSerieModule.NomLogiqueFonction`
- `NomLogiqueModule.IdentifiantMatériel`
- `NomLogiqueModule.NomLogiqueFonction`

Cette fonction n'exige pas que le PWM soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmOutput.isOnline()` pour tester si le PWM est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le PWM sans ambiguïté

Retourne :

un objet de classe `YPwmOutput` qui permet ensuite de contrôler le PWM.

YPwmOutput.FirstPwmOutput()**YPwmOutput****yFirstPwmOutput()**`YPwmOutput.FirstPwmOutput()`

Commence l'énumération des PWM accessibles par la librairie.

`YPwmOutput` **FirstPwmOutput()**

Utiliser la fonction `YPwmOutput.nextPwmOutput()` pour itérer sur les autres PWM.

Retourne :

un pointeur sur un objet `YPwmOutput`, correspondant au premier PWM accessible en ligne, ou `null` si il n'y a pas de PWM disponibles.

pwmoutput→describe()`pwmoutput.describe()`**YPwmOutput**

Retourne un court texte décrivant de manière non-ambigüe l'instance du PWM au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le PWM (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

pwmoutput→dutyCycleMove()**YPwmOutput****pwmoutput.dutyCycleMove()**

Déclenche une variation progressive de la longueur des impulsions vers une valeur donnée.

```
int dutyCycleMove( double target, int ms_duration)
```

Paramètres :

target nouveau duty cycle à la fin de la transition (nombre flottant, entre 0 et 1)

ms_duration durée totale de la transition, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→get_advertisedValue()
pwmoutput→advertisedValue()
pwmoutput.get_advertisedValue()

YPwmOutput

Retourne la valeur courante du PWM (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du PWM (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

pwmoutput→get_dutyCycle()**YPwmOutput****pwmoutput→dutyCycle()****pwmoutput.get_dutyCycle()**

Retourne le duty cycle du PWM, en pour cents.

double **get_dutyCycle()**

Retourne :

une valeur numérique représentant le duty cycle du PWM, en pour cents

En cas d'erreur, déclenche une exception ou retourne Y_DUTYCYCLE_INVALID.

pwmoutput→get_dutyCycleAtPowerOn()

YPwmOutput

pwmoutput→dutyCycleAtPowerOn()

pwmoutput.get_dutyCycleAtPowerOn()

Retourne le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

double **get_dutyCycleAtPowerOn()**

Retourne :

une valeur numérique représentant le duty cycle du PWM au démarrage du module, sous la forme d'un nombre à virgule entre 0 et 100

En cas d'erreur, déclenche une exception ou retourne Y_DUTYCYCLEATPOWERON_INVALID.

pwmoutput→**get_enabled()****YPwmOutput****pwmoutput**→**enabled()**`pwmoutput.get_enabled()`

Retourne l'état de fonctionnement du PWM.

int **get_enabled()**

Retourne :

soit `Y_ENABLED_FALSE`, soit `Y_ENABLED_TRUE`, selon l'état de fonctionnement du PWM

En cas d'erreur, déclenche une exception ou retourne `Y_ENABLED_INVALID`.

pwmoutput→get_enabledAtPowerOn()

YPwmOutput

pwmoutput→enabledAtPowerOn()

pwmoutput.get_enabledAtPowerOn()

Retourne l'état de fonctionnement du PWM à la mise sous tension du module.

int get_enabledAtPowerOn()

Retourne :

soit Y_ENABLEDATPOWERON_FALSE, soit Y_ENABLEDATPOWERON_TRUE, selon l'état de fonctionnement du PWM à la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne Y_ENABLEDATPOWERON_INVALID.

pwmoutput→get_errorMessage()**YPwmOutput****pwmoutput→errorMessage()****pwmoutput.errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du PWM.

pwmoutput→get_errorType()
pwmoutput→errorType()
pwmoutput.get_errorType()

YPwmOutput

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du PWM.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du PWM.

pwmoutput→get_frequency()**YPwmOutput****pwmoutput→frequency()****pwmoutput.get_frequency()**

Retourne la fréquence du PWM en Hz.

double **get_frequency()**

Retourne :

une valeur numérique représentant la fréquence du PWM en Hz

En cas d'erreur, déclenche une exception ou retourne Y_FREQUENCY_INVALID.

pwmoutput→get_friendlyName()

YPwmOutput

pwmoutput→friendlyName()

pwmoutput.get_friendlyName()

Retourne un identifiant global du PWM au format `NOM_MODULE.NOM_FONCTION`.

String get_friendlyName()

Le chaîne retournée utilise soit les noms logiques du module et du PWM si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du PWM (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le PWM en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

pwmoutput→get_functionDescriptor()
pwmoutput→functionDescriptor()
pwmoutput.get_functionDescriptor()

YPwmOutput

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

pwmoutput→get_functionId()

YPwmOutput

pwmoutput→functionId()

pwmoutput.get_functionId()

Retourne l'identifiant matériel du PWM, sans référence au module.

String **get_functionId()** ()

Par exemple relay1.

Retourne :

une chaîne de caractères identifiant le PWM (ex: relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FUNCTIONID_INVALID.

pwmoutput→**get_hardwareId()**
pwmoutput→**hardwareId()**
pwmoutput.**get_hardwareId()**

YPwmOutput

Retourne l'identifiant matériel unique du PWM au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du PWM (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le PWM (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

pwmoutput→get_logicalName()

YPwmOutput

pwmoutput→logicalName()

pwmoutput.get_logicalName()

Retourne le nom logique du PWM.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du PWM.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

pwmoutput→**get_module()****YPwmOutput****pwmoutput**→**module()**`pwmoutput.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

pwmoutput→get_period()

YPwmOutput

pwmoutput→period()`pwmoutput.get_period()`

Retourne la période du PWM en millisecondes.

double **get_period()** ()

Retourne :

une valeur numérique représentant la période du PWM en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y_PERIOD_INVALID.

pwmoutput→get_pulseDuration()**YPwmOutput****pwmoutput→pulseDuration()****pwmoutput.get_pulseDuration()**

Retourne la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule.

double **get_pulseDuration()**

Retourne :

une valeur numérique représentant la longueur d'une impulsion du PWM en millisecondes, sous forme d'un chiffre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_PULSEDURATION_INVALID.

pwmoutput→get_userdata()

YPwmOutput

pwmoutput→userData()

pwmoutput.getUserData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.

Object **get_userdata()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

pwmoutput→**isOnline()**`pwmoutput.isOnline()`**YPwmOutput**

Vérifie si le module hébergeant le PWM est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du PWM sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le PWM est joignable, `false` sinon

pwmoutput→load()`pwmoutput.load()`**YPwmOutput**

Met en cache les valeurs courantes du PWM, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→**nextPwmOutput()****YPwmOutput****pwmoutput.nextPwmOutput()**

Continue l'énumération des PWM commencée à l'aide de `yFirstPwmOutput()`.

`YPwmOutput` **nextPwmOutput()**

Retourne :

un pointeur sur un objet `YPwmOutput` accessible en ligne, ou `null` lorsque l'énumération est terminée.

pwmoutput→pulseDurationMove()**YPwmOutput****pwmoutput.pulseDurationMove()**

Déclenche une transition progressive de la longueur des impulsions vers une valeur donnée.

```
int pulseDurationMove( double ms_target, int ms_duration)
```

N'importe quel changement de fréquence, duty cycle, période ou encore de longueur d'impulsion annulera tout processus de transition en cours.

Paramètres :

- ms_target** nouvelle longueur des impulsions à la fin de la transition (nombre flottant, représentant la longueur en millisecondes)
- ms_duration** durée totale de la transition, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→registerValueCallback()**YPwmOutput****pwmoutput.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

pwmoutput→set_dutyCycle()
pwmoutput→setDutyCycle()
pwmoutput.set_dutyCycle()

YPwmOutput

Modifie le duty cycle du PWM, en pour cents.

```
int set_dutyCycle( double newval)
```

Paramètres :

newval une valeur numérique représentant le duty cycle du PWM, en pour cents

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→**set_dutyCycleAtPowerOn()**
pwmoutput→**setDutyCycleAtPowerOn()**
pwmoutput.set_dutyCycleAtPowerOn()

YPwmOutput

Modifie le duty cycle du PWM au démarrage du module.

```
int set_dutyCycleAtPowerOn( double newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval une valeur numérique représentant le duty cycle du PWM au démarrage du module

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

`pwmoutput→set_enabled()`
`pwmoutput→setEnabled()`
`pwmoutput.set_enabled()`

YPwmOutput

Démarre ou arrête le PWM.

```
int set_enabled( int newval)
```

Paramètres :

`newval` soit `Y_ENABLED_FALSE`, soit `Y_ENABLED_TRUE`

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→**set_enabledAtPowerOn()**
pwmoutput→**setEnabledAtPowerOn()**
pwmoutput.set_enabledAtPowerOn()

YPwmOutput

Modifie l'état du fonctionnement du PWM à la mise sous tension du module.

```
int set_enabledAtPowerOn( int newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval soit `Y_ENABLEDATPOWERON_FALSE`, soit `Y_ENABLEDATPOWERON_TRUE`, selon l'état du fonctionnement du PWM à la mise sous tension du module

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→set_frequency()
pwmoutput→setFrequency()
pwmoutput.set_frequency()

YPwmOutput

Modifie la fréquence du PWM.

```
int set_frequency( double newval)
```

Le duty cycle est conservé grâce à un changement automatique de la longueur des impulsions.

Paramètres :

newval une valeur numérique représentant la fréquence du PWM

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→set_logicalName()
pwmoutput→setLogicalName()
pwmoutput.set_logicalName()

YPwmOutput

Modifie le nom logique du PWM.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du PWM.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→set_period()

YPwmOutput

pwmoutput→setPeriod()`pwmoutput.set_period()`

Modifie la période du PWM en millisecondes.

```
int set_period( double newval)
```

Paramètres :

newval une valeur numérique représentant la période du PWM en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→set_pulseDuration()**YPwmOutput****pwmoutput→setPulseDuration()****pwmoutput.set_pulseDuration()**

Modifie la longueur des impulsions du PWM, en millisecondes.

```
int set_pulseDuration( double newval)
```

Attention, la longueur d'une impulsion ne peut pas être plus grande que la période, sinon la longueur sera automatiquement tronquée à la période.

Paramètres :

newval une valeur numérique représentant la longueur des impulsions du PWM, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmoutput→set_userdata()

YPwmOutput

pwmoutput→setUserData()

pwmoutput.set_userdata()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

void **set_userdata**(Object **data**)

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

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');
c++	#include "yocto_pwmpowersource.h"
m	#import "yocto_pwmpowersource.h"
pas	uses yocto_pwmpowersource;
vb	yocto_pwmpowersource.vb
cs	yocto_pwmpowersource.cs
java	import com.yoctopuce.YoctoAPI.YPwmPowerSource;
py	from yocto_pwmpowersource import *

Fonction globales

yFindPwmPowerSource(func)

Permet de retrouver une source de tension d'après un identifiant donné.

yFirstPwmPowerSource()

Commence l'énumération des Source de tension accessibles par la librairie.

Méthodes des objets YPwmPowerSource

pwmpowersource→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de la source de tension au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

pwmpowersource→get_advertisedValue()

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

pwmpowersource→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

pwmpowersource→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

pwmpowersource→get_friendlyName()

Retourne un identifiant global de la source de tension au format `NOM_MODULE . NOM_FONCTION`.

pwmpowersource→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

pwmpowersource→get_functionId()

Retourne l'identifiant matériel de la source de tension, sans référence au module.

pwmpowersource→get_hardwareId()

Retourne l'identifiant matériel unique de la source de tension au format `SERIAL . FUNCTIONID`.

pwmpowersource→get_logicalName()

Retourne le nom logique de la source de tension.

pwmpowersource→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

pwmpowersource→get_module_async(callback, context)

	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<code>pwmpowersource→get_powerMode()</code>	Retourne la source de tension utilisé par tous les PWM du même module.
<code>pwmpowersource→get_userData()</code>	Retourne le contenu de l'attribut <code>userData</code> , précédemment stocké à l'aide de la méthode <code>set_userData</code> .
<code>pwmpowersource→isOnline()</code>	Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.
<code>pwmpowersource→isOnline_async(callback, context)</code>	Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.
<code>pwmpowersource→load(msValidity)</code>	Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.
<code>pwmpowersource→load_async(msValidity, callback, context)</code>	Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.
<code>pwmpowersource→nextPwmPowerSource()</code>	Continue l'énumération des Source de tension commencée à l'aide de <code>yFirstPwmPowerSource()</code> .
<code>pwmpowersource→registerValueCallback(callback)</code>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<code>pwmpowersource→set_logicalName(newval)</code>	Modifie le nom logique de la source de tension.
<code>pwmpowersource→set_powerMode(newval)</code>	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.
<code>pwmpowersource→set_userData(data)</code>	Enregistre un contexte libre dans l'attribut <code>userData</code> de la fonction, afin de le retrouver plus tard à l'aide de la méthode <code>get_userData</code> .
<code>pwmpowersource→wait_async(callback, context)</code>	Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

**YPwmPowerSource.FindPwmPowerSource()
yFindPwmPowerSource()
YPwmPowerSource.FindPwmPowerSource()**

YPwmPowerSource

Permet de retrouver une source de tension d'après un identifiant donné.

`YPwmPowerSource` **FindPwmPowerSource**(String **func**)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la source de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPwmPowerSource.isOnline()` pour tester si la source de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence la source de tension sans ambiguïté

Retourne :

un objet de classe `YPwmPowerSource` qui permet ensuite de contrôler la source de tension.

**YPwmPowerSource.FirstPwmPowerSource()
yFirstPwmPowerSource()**

YPwmPowerSource

YPwmPowerSource.FirstPwmPowerSource()

Commence l'énumération des Source de tension accessibles par la librairie.

YPwmPowerSource **FirstPwmPowerSource()**

Utiliser la fonction `YPwmPowerSource.nextPwmPowerSource()` pour itérer sur les autres Source de tension.

Retourne :

un pointeur sur un objet `YPwmPowerSource`, correspondant à la première source de tension accessible en ligne, ou `null` si il n'y a pas de Source de tension disponibles.

pwmpowersource→describe()**YPwmPowerSource****pwmpowersource.describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance de la source de tension au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

String **describe()**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant la source de tension (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

pwmpowersource→get_advertisedValue()

YPwmPowerSource

pwmpowersource→advertisedValue()

pwmpowersource.get_advertisedValue()

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante de la source de tension (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

pwmpowersource→get_errorMessage()**YPwmPowerSource****pwmpowersource→errorMessage()****pwmpowersource.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la source de tension.

pwmpowersource→get_errorType()

YPwmPowerSource

pwmpowersource→errorType()

pwmpowersource.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la source de tension.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la source de tension.

pwmpowersource→get_friendlyName()
pwmpowersource→friendlyName()
pwmpowersource.get_friendlyName()

YPwmPowerSource

Retourne un identifiant global de la source de tension au format `NOM_MODULE.NOM_FONCTION`.

String get_friendlyName()

Le chaîne retournée utilise soit les noms logiques du module et de la source de tension si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de la source de tension (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant la source de tension en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

pwmpowersource→get_functionDescriptor()

YPwmPowerSource

pwmpowersource→functionDescriptor()

pwmpowersource.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

pwmpowersource→get_functionId()**YPwmPowerSource****pwmpowersource→functionId()****pwmpowersource.get_functionId()**

Retourne l'identifiant matériel de la source de tension, sans référence au module.

String **get_functionId()** ()

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant la source de tension (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

pwmpowersource→**get_hardwareId()**

YPwmPowerSource

pwmpowersource→**hardwareId()**

pwmpowersource.get_hardwareId()

Retourne l'identifiant matériel unique de la source de tension au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()** ()

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la source de tension (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant la source de tension (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

pwmpowersource→get_logicalName()**YPwmPowerSource****pwmpowersource→logicalName()****pwmpowersource.get_logicalName()**

Retourne le nom logique de la source de tension.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique de la source de tension.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

pwmpowersource→**get_module()**
pwmpowersource→**module()**
pwmpowersource.get_module()

YPwmPowerSource

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

YModule **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :

une instance de YModule

pwmpowersource→get_powerMode()**YPwmPowerSource****pwmpowersource→powerMode()****pwmpowersource.get_powerMode()**

Retourne la source de tension utilisé par tous les PWM du même module.

int **get_powerMode()**

Retourne :

une valeur parmi Y_POWERMODE_USB_5V, Y_POWERMODE_USB_3V, Y_POWERMODE_EXT_V et Y_POWERMODE_OPNDRN représentant la source de tension utilisé par tous les PWM du même module

En cas d'erreur, déclenche une exception ou retourne Y_POWERMODE_INVALID.

pwmpowersource→get_userdata()

YPwmPowerSource

pwmpowersource→userData()

pwmpowersource.get_userdata()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.

Object **get_userdata()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

pwmpowersource→isOnline()**YPwmPowerSource****pwmpowersource.isOnline()**

Vérifie si le module hébergeant la source de tension est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache de la source de tension sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

true si la source de tension est joignable, false sinon

pwmpowersource→**load()**`pwmpowersource.load()`**YPwmPowerSource**

Met en cache les valeurs courantes de la source de tension, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmpowersource→nextPwmPowerSource()**YPwmPowerSource****pwmpowersource.nextPwmPowerSource()**

Continue l'énumération des Source de tension commencée à l'aide de `yFirstPwmPowerSource()`.

`YPwmPowerSource` **nextPwmPowerSource()**

Retourne :

un pointeur sur un objet `YPwmPowerSource` accessible en ligne, ou `null` lorsque l'énumération est terminée.

pwmpowersource→**registerValueCallback()**

YPwmPowerSource

pwmpowersource.registerValueCallback()

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

pwmpowersource→set_logicalName()
pwmpowersource→setLogicalName()
pwmpowersource.set_logicalName()

YPwmPowerSource

Modifie le nom logique de la source de tension.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de la source de tension.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmpowersource→**set_powerMode()****YPwmPowerSource****pwmpowersource**→**setPowerMode()****pwmpowersource.set_powerMode()**

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.

```
int set_powerMode( int newval)
```

Le PWM peut aussi en mode open drain, dans ce code il tire activement la ligne à zéro volts. Attention ce paramètre est commun à tous les PWM du module, si vous changez le valeur de ce paramètre, tous les PWM situés sur le même module seront affectés. Si vous souhaitez que le changement de ce paramètre soit conservé après un redémarrage du module, n'oubliez pas d'appeler la méthode `saveToFlash()`.

Paramètres :

newval une valeur parmi `Y_POWERMODE_USB_5V`, `Y_POWERMODE_USB_3V`, `Y_POWERMODE_EXT_V` et `Y_POWERMODE_OPNDRN` représentant le mode fonctionnement des PWM qui peut sortir du 5 volts isolé issu de l'USB, du 3V isolé issu de l'USB, une tension arbitraire issue de l'alimentation externe

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

pwmpowersource→set_userdata()**YPwmPowerSource****pwmpowersource→setUserData()****pwmpowersource.set_userdata()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

```
void set_userdata( Object data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

3.35. Interface du quaternion

La class YQt de la librairie Yoctopuce permet d'accéder à l'estimation de l'orientation tridimensionnelle du Yocto-3D sous forme d'un quaternion. Il n'est en général pas nécessaire d'y accéder directement, la classe YGyro offrant une abstraction de plus haut niveau.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_gyro.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YGyro = yoctolib.YGyro;</code>
php	<code>require_once('yocto_gyro.php');</code>
cpp	<code>#include "yocto_gyro.h"</code>
m	<code>#import "yocto_gyro.h"</code>
pas	<code>uses yocto_gyro;</code>
vb	<code>yocto_gyro.vb</code>
cs	<code>yocto_gyro.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YGyro;</code>
py	<code>from yocto_gyro import *</code>

Fonction globales

yFindQt(func)

Permet de retrouver un élément de quaternion d'après un identifiant donné.

yFirstQt()

Commence l'énumération des éléments de quaternion accessibles par la librairie.

Méthodes des objets YQt

qt→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

qt→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'élément de quaternion au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

qt→get_advertisedValue()

Retourne la valeur courante de l'élément de quaternion (pas plus de 6 caractères).

qt→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), 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 <code>SERIAL.FUNCTIONID</code> .
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 <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
qt→get_module_async(callback, context)	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
qt→get_recordedData(startTime, endTime)	Retourne un objet <code>DataSet</code> représentant des mesures de ce capteur précédemment enregistrées à l'aide du <code>DataLogger</code> , pour l'intervalle de temps spécifié.
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 <code>userData</code> , précédemment stocké à l'aide de la méthode <code>set_userData</code> .
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 <code>calibrateFromPoints</code> .
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 <code>yFirstQt()</code> .
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)	

3. Reference

Modifie la mémoire de valeur maximale observée.

qt→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

qt→set_logicalName(newval)

Modifie le nom logique de l'élément de quaternion.

qt→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

qt→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

qt→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

qt→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userData.

qt→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YQt.FindQt()**YQt****yFindQt()****YQt.FindQt()**

Permet de retrouver un élément de quaternion d'après un identifiant donné.

YQt **FindQt**(String **func**)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'élément de quaternion soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YQt.isOnline()` pour tester si l'élément de quaternion est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence l'élément de quaternion sans ambiguïté

Retourne :

un objet de classe `YQt` qui permet ensuite de contrôler l'élément de quaternion.

YQt.FirstQt()

YQt

yFirstQt()`YQt.FirstQt()`

Commence l'énumération des éléments de quaternion accessibles par la librairie.

YQt FirstQt()

Utiliser la fonction `YQt.NextQt()` pour itérer sur les autres éléments de quaternion.

Retourne :

un pointeur sur un objet `YQt`, correspondant au premier élément de quaternion accessible en ligne, ou `null` si il n'y a pas de éléments de quaternion disponibles.

qt→calibrateFromPoints()**YQt****qt.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                        ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→describe()`qt.describe()`

YQt

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'élément de quaternion au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant l'élément de quaternion (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

qt→get_advertisedValue()**YQt****qt→advertisedValue()**`qt.get_advertisedValue()`

Retourne la valeur courante de l'élément de quaternion (pas plus de 6 caractères).

String `get_advertisedValue()`

Retourne :

une chaîne de caractères représentant la valeur courante de l'élément de quaternion (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

qt→get_currentRawValue()

YQt

qt→currentRawValue()**qt.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en unités, sous forme de nombre à virgule.

double **get_currentRawValue()**

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en unités, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

qt→get_currentValue()**YQt****qt→currentValue()**`qt.get_currentValue()`

Retourne la valeur actuelle de la coordonnée, en unités, sous forme de nombre à virgule.

`double get_currentValue()`**Retourne :**

une valeur numérique représentant la valeur actuelle de la coordonnée, en unités, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTVALUE_INVALID`.

qt→get_errorMessage()

YQt

qt→errorMessage()`qt.get_errorMessage()`

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

String `get_errorMessage()`

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'élément de quaternion.

qt→get_errorType()**YQt****qt→errorType()**`qt.get_errorType()`

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'élément de quaternion.

`int get_errorType()`

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'élément de quaternion.

qt→get_friendlyName()**YQt****qt→friendlyName()**`qt.get_friendlyName()`

Retourne un identifiant global de l'élément de quaternion au format `NOM_MODULE.NOM_FONCTION`.

String `get_friendlyName()`

Le chaîne retournée utilise soit les noms logiques du module et de l'élément de quaternion si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'élément de quaternion (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'élément de quaternion en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

qt→get_functionDescriptor()
qt→functionDescriptor()
qt.get_functionDescriptor()

YQt

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String get_functionDescriptor()

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

qt→get_functionId()

YQt

qt→functionId()`qt.get_functionId()`

Retourne l'identifiant matériel de l'élément de quaternion, sans référence au module.

String `get_functionId()`

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant l'élément de quaternion (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

qt→get_hardwareId()**YQt****qt→hardwareId()**`qt.get_hardwareId()`

Retourne l'identifiant matériel unique de l'élément de quaternion au format `SERIAL.FUNCTIONID`.

`String` **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'élément de quaternion (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant l'élément de quaternion (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

qt→get_highestValue()

YQt

qt→highestValue()`qt.get_highestValue()`

Retourne la valeur maximale observée pour la coordonnée depuis le démarrage du module.

double **get_highestValue()**

Retourne :

une valeur numérique représentant la valeur maximale observée pour la coordonnée depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

qt→get_logFrequency()**YQt****qt→logFrequency()**`qt.get_logFrequency()`

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

String **get_logFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

qt→get_logicalName()

YQt

qt→logicalName()`qt.get_logicalName()`

Retourne le nom logique de l'élément de quaternion.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique de l'élément de quaternion.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

qt→get_lowestValue()**YQt****qt→lowestValue()**`qt.get_lowestValue()`

Retourne la valeur minimale observée pour la coordonnée depuis le démarrage du module.

double **get_lowestValue()**

Retourne :

une valeur numérique représentant la valeur minimale observée pour la coordonnée depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

qt→get_module()

YQt

qt→module()`qt.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

qt→get_recordedData()**YQt****qt→recordedData()**`qt.get_recordedData()`

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

`YDataSet` **get_recordedData**(long **startTime**, long **endTime**)

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intercalé de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

qt→get_reportFrequency()

YQt

qt→reportFrequency()`qt.get_reportFrequency()`

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String **get_reportFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

qt→get_resolution()**YQt****qt→resolution()**`qt.get_resolution()`

Retourne la résolution des valeurs mesurées.

double **get_resolution()** ()

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_RESOLUTION_INVALID.

qt→get_unit()

YQt

qt→unit()`qt.get_unit()`

Retourne l'unité dans laquelle la coordonnée est exprimée.

String **get_unit()**

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la coordonnée est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

qt→get_userdata()**YQt****qt→userdata()**`qt.get_userdata()`

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

Object `get_userdata()`

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

qt→isOnline()`qt.isOnline()`**YQt**

Vérifie si le module hébergeant l'élément de quaternion est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache de l'élément de quaternion sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'élément de quaternion est joignable, `false` sinon

qt→load()`qt.load()`**YQt**

Met en cache les valeurs courantes de l'élément de quaternion, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→loadCalibrationPoints()

YQt

qt.loadCalibrationPoints()

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→**nextQt()****qt.nextQt()****YQt**

Continue l'énumération des éléments de quaternion commencée à l'aide de `yFirstQt()`.

YQt **nextQt()**

Retourne :

un pointeur sur un objet **YQt** accessible en ligne, ou `null` lorsque l'énumération est terminée.

qt→registerTimedReportCallback()

YQt

qt.registerTimedReportCallback()

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

qt→registerValueCallback()**YQt****qt.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

qt→set_highestValue()

YQt

qt→setHighestValue()`qt.set_highestValue()`

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→set_logFrequency()**YQt****qt→setLogFrequency()**`qt.set_logFrequency()`

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

`int set_logFrequency(String newval)`

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→set_logicalName()**YQt****qt→setLogicalName()**`qt.set_logicalName()`

Modifie le nom logique de l'élément de quaternion.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'élément de quaternion.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→set_lowestValue()**YQt****qt→setLowestValue()**`qt.set_lowestValue()`

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→set_reportFrequency()**YQt****qt→setReportFrequency()****qt.set_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→set_resolution()**YQt****qt→setResolution()**`qt.set_resolution()`

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

qt→set_userdata()

YQt

qt→setUserData()`qt.set_userdata()`

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

`void set_userdata(Object data)`

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

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');
c++	#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()

3. Reference

	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<code>realtimeclock→get_module_async(callback, context)</code>	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<code>realtimeclock→get_timeSet()</code>	Retourne vrai si l'horloge à été mise à l'heure, sinon faux.
<code>realtimeclock→get_unixTime()</code>	Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).
<code>realtimeclock→get_userData()</code>	Retourne le contenu de l'attribut <code>userData</code> , précédemment stocké à l'aide de la méthode <code>set_userData</code> .
<code>realtimeclock→get_utcOffset()</code>	Retourne le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).
<code>realtimeclock→isOnline()</code>	Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.
<code>realtimeclock→isOnline_async(callback, context)</code>	Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.
<code>realtimeclock→load(msValidity)</code>	Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.
<code>realtimeclock→load_async(msValidity, callback, context)</code>	Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.
<code>realtimeclock→nextRealTimeClock()</code>	Continue l'énumération des horloge commencée à l'aide de <code>yFirstRealTimeClock()</code> .
<code>realtimeclock→registerValueCallback(callback)</code>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<code>realtimeclock→set_logicalName(newval)</code>	Modifie le nom logique de l'horloge.
<code>realtimeclock→set_unixTime(newval)</code>	Modifie l'heure courante.
<code>realtimeclock→set_userData(data)</code>	Enregistre un contexte libre dans l'attribut <code>userData</code> de la fonction, afin de le retrouver plus tard à l'aide de la méthode <code>get_userData</code> .
<code>realtimeclock→set_utcOffset(newval)</code>	Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).
<code>realtimeclock→wait_async(callback, context)</code>	Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YRealTimeClock.FindRealTimeClock()**YRealTimeClock****yFindRealTimeClock()****YRealTimeClock.FindRealTimeClock()**

Permet de retrouver une horloge d'après un identifiant donné.

```
YRealTimeClock FindRealTimeClock( String func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'horloge soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRealTimeClock.isOnline()` pour tester si l'horloge est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence l'horloge sans ambiguïté

Retourne :

un objet de classe `YRealTimeClock` qui permet ensuite de contrôler l'horloge.

YRealTimeClock.FirstRealTimeClock()

YRealTimeClock

yFirstRealTimeClock()

YRealTimeClock.FirstRealTimeClock()

Commence l'énumération des horloge accessibles par la librairie.

[YRealTimeClock](#) **FirstRealTimeClock()**

Utiliser la fonction `YRealTimeClock.nextRealTimeClock()` pour itérer sur les autres horloge.

Retourne :

un pointeur sur un objet `YRealTimeClock`, correspondant à la première horloge accessible en ligne, ou `null` si il n'y a pas de horloge disponibles.

realtimeclock→describe()**YRealTimeClock****realtimeclock.describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'horloge au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String **describe()**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant l'horloge (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

realtimeclock→get_advertisedValue()

YRealTimeClock

realtimeclock→advertisedValue()

realtimeclock.get_advertisedValue()

Retourne la valeur courante de l'horloge (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante de l'horloge (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

realtimeclock→get_dateTime()**YRealTimeClock****realtimeclock→dateTime()****realtimeclock.get_dateTime()**

Retourne l'heure courante au format "AAAA/MM/JJ hh:mm:ss"

String **get_dateTime()**

Retourne :

une chaîne de caractères représentant l'heure courante au format "AAAA/MM/JJ hh:mm:ss"

En cas d'erreur, déclenche une exception ou retourne Y_DATETIME_INVALID.

realtimeclock→get_errorMessage()

YRealTimeClock

realtimeclock→errorMessage()

realtimeclock.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'horloge.

realtimeclock→get_errorType()**YRealTimeClock****realtimeclock→errorType()****realtimeclock.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'horloge.

realtimeclock→get_friendlyName()

YRealTimeClock

realtimeclock→friendlyName()

realtimeclock.get_friendlyName()

Retourne un identifiant global de l'horloge au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et de l'horloge si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'horloge (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'horloge en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

realtimeclock→get_functionDescriptor()**YRealTimeClock****realtimeclock→functionDescriptor()****realtimeclock.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

realtimeclock→get_functionId()

YRealTimeClock

realtimeclock→functionId()

realtimeclock.get_functionId()

Retourne l'identifiant matériel de l'horloge, sans référence au module.

String **get_functionId()** ()

Par exemple relay1.

Retourne :

une chaîne de caractères identifiant l'horloge (ex: relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FUNCTIONID_INVALID.

realtimeclock→get_hardwareId()**YRealTimeClock****realtimeclock→hardwareId()****realtimeclock.get_hardwareId()**

Retourne l'identifiant matériel unique de l'horloge au format `SERIAL.FUNCTIONID`.

String `get_hardwareId()`

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'horloge (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant l'horloge (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

realtimeclock→get_logicalName()

YRealTimeClock

realtimeclock→logicalName()

realtimeclock.get_logicalName()

Retourne le nom logique de l'horloge.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique de l'horloge.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

realtimeclock→get_module()**YRealTimeClock****realtimeclock→module()****realtimeclock.get_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

YModule **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :

une instance de YModule

realtimeclock→get_timeSet()

YRealTimeClock

realtimeclock→timeSet()

realtimeclock.get_timeSet()

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

int get_timeSet()

Retourne :

soit Y_TIMESET_FALSE, soit Y_TIMESET_TRUE, selon vrai si l'horloge à été mise à l'heure, sinon faux

En cas d'erreur, déclenche une exception ou retourne Y_TIMESET_INVALID.

realtimeclock→get_unixTime()**YRealTimeClock****realtimeclock→unixTime()****realtimeclock.get_unixTime()**

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

long **get_unixTime()** ()

Retourne :

un entier représentant l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970)

En cas d'erreur, déclenche une exception ou retourne Y_UNIXTIME_INVALID.

realtimeclock→get_userdata()

YRealTimeClock

realtimeclock→userData()

realtimeclock.get_userdata()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

realtimeclock→get_utcOffset()**YRealTimeClock****realtimeclock→utcOffset()****realtimeclock.get_utcOffset()**

Retourne le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

int **get_utcOffset()**

Retourne :

un entier représentant le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone)

En cas d'erreur, déclenche une exception ou retourne Y_UTCOffset_INVALID.

realtimeclock→isOnline()

YRealTimeClock

realtimeclock.isOnline()

Vérifie si le module hébergeant l'horloge est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache de l'horloge sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

true si l'horloge est joignable, false sinon

realtimeclock→**load()****realtimeclock.load()****YRealTimeClock**

Met en cache les valeurs courantes de l'horloge, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

realtimeclock→nextRealTimeClock()

YRealTimeClock

realtimeclock.nextRealTimeClock()

Continue l'énumération des horloge commencée à l'aide de `yFirstRealTimeClock()`.

`YRealTimeClock` **nextRealTimeClock()**

Retourne :

un pointeur sur un objet `YRealTimeClock` accessible en ligne, ou `null` lorsque l'énumération est terminée.

realtimeclock→registerValueCallback()**YRealTimeClock****realtimeclock.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

realtimeclock→set_logicalName()

YRealTimeClock

realtimeclock→setLogicalName()

realtimeclock.set_logicalName()

Modifie le nom logique de l'horloge.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'horloge.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

realtimeclock→**set_unixTime()**
realtimeclock→**setUnixTime()**
realtimeclock.set_unixTime()

YRealTimeClock

Modifie l'heure courante.

```
int set_unixTime( long newval)
```

L'heure est passée au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970). Si l'heure UTC est connue, l'attribut `utcOffset` sera automatiquement ajusté en fonction de l'heure configurée.

Paramètres :

newval un entier représentant l'heure courante

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

realtimeclock→set_userdata()

YRealTimeClock

realtimeclock→setUserData()

realtimeclock.set_userdata()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

void **set_userdata**(Object **data**)

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

realtimeclock→set_utcOffset()**YRealTimeClock****realtimeclock→setUtcOffset()****realtimeclock.set_utcOffset()**

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

```
int set_utcOffset( int newval)
```

Le décalage est automatiquement arrondi au quart d'heure le plus proche. Si l'heure UTC est connue, l'heure courante sera automatiquement adaptée en fonction du décalage choisi.

Paramètres :

newval un entier représentant le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

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	<code><script type='text/javascript' src='yocto_reframe.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YRefFrame = yoctolib.YRefFrame;</code>
php	<code>require_once('yocto_reframe.php');</code>
c++	<code>#include "yocto_reframe.h"</code>
m	<code>#import "yocto_reframe.h"</code>
pas	<code>uses yocto_reframe;</code>
vb	<code>yocto_reframe.vb</code>
cs	<code>yocto_reframe.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YRefFrame;</code>
py	<code>from yocto_reframe import *</code>

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

reframe→cancel3DCalibration()

Annule la calibration tridimensionnelle en cours, et rétabli les réglages normaux.

reframe→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du référentiel au format `TYPE (NAME) = SERIAL.FUNCTIONID`.

reframe→get_3DCalibrationHint()

Retourne les instructions à suivre pour procéder à la calibration tridimensionnelle initiée avec la méthode `start3DCalibration`.

reframe→get_3DCalibrationLogMsg()

Retourne le dernier message de log produit par le processus de calibration.

reframe→get_3DCalibrationProgress()

Retourne l'avancement global du processus de calibration tridimensionnelle initié avec la méthode `start3DCalibration`.

reframe→get_3DCalibrationStage()

Retourne l'index de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.

reframe→get_3DCalibrationStageProgress()

Retourne l'avancement de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.

reframe→get_advertisedValue()

Retourne la valeur courante du référentiel (pas plus de 6 caractères).

reframe→get_bearing()

Retourne le cap de référence utilisé par le compas.

refframe→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

refframe→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

refframe→get_friendlyName()

Retourne un identifiant global du référentiel au format `NOM_MODULE . NOM_FONCTION`.

refframe→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

refframe→get_functionId()

Retourne l'identifiant matériel du référentiel, sans référence au module.

refframe→get_hardwareId()

Retourne l'identifiant matériel unique du référentiel au format `SERIAL . FUNCTIONID`.

refframe→get_logicalName()

Retourne le nom logique du référentiel.

refframe→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

refframe→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

refframe→get_mountOrientation()

Retourne l'orientation à l'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

refframe→get_mountPosition()

Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

refframe→get_userData()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

refframe→isOnline()

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

refframe→isOnline_async(callback, context)

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

refframe→load(msValidity)

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

refframe→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

refframe→more3DCalibration()

Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode `start3DCalibration`.

refframe→nextRefFrame()

Continue l'énumération des référentiels commencée à l'aide de `yFirstRefFrame()`.

refframe→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

refframe→save3DCalibration()

Applique les paramètres de calibration tridimensionnelle précédemment calculés.

refframe→set_bearing(newval)

3. Reference

Modifie le cap de référence utilisé par le compas.

refframe→**set_logicalName**(**newval**)

Modifie le nom logique du référentiel.

refframe→**set_mountPosition**(**position**, **orientation**)

Modifie le référentiel de la boussole et des inclinomètres.

refframe→**set_userData**(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

refframe→**start3DCalibration**()

Initie le processus de calibration tridimensionnelle des capteurs.

refframe→**wait_async**(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YRefFrame.FindRefFrame()**YRefFrame****yFindRefFrame()**`YRefFrame.FindRefFrame()`

Permet de retrouver un référentiel d'après un identifiant donné.

```
YRefFrame FindRefFrame( String func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le référentiel soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRefFrame.isOnline()` pour tester si le référentiel est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le référentiel sans ambiguïté

Retourne :

un objet de classe `YRefFrame` qui permet ensuite de contrôler le référentiel.

YRefFrame.FirstRefFrame()

YRefFrame

yFirstRefFrame()`YRefFrame.FirstRefFrame()`

Commence l'énumération des référentiels accessibles par la librairie.

`YRefFrame` **FirstRefFrame()**

Utiliser la fonction `YRefFrame.nextRefFrame()` pour itérer sur les autres référentiels.

Retourne :

un pointeur sur un objet `YRefFrame`, correspondant au premier référentiel accessible en ligne, ou `null` si il n'y a pas de référentiels disponibles.

refframe→cancel3DCalibration()**YRefFrame****refframe.cancel3DCalibration()**

Annule la calibration tridimensionnelle en cours, et rétabli les réglages normaux.

int cancel3DCalibration()

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→describe()`refframe.describe()`**YRefFrame**

Retourne un court texte décrivant de manière non-ambigüe l'instance du référentiel au format
`TYPE (NAME) = SERIAL . FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le référentiel (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

refframe→get_3DCalibrationHint()**YRefFrame****refframe→3DCalibrationHint()****refframe.get_3DCalibrationHint()**

Retourne les instructions à suivre pour procéder à la calibration tridimensionnelle initiée avec la méthode `start3DCalibration`.

String `get_3DCalibrationHint()`

Retourne :

une chaîne de caractères.

refframe→get_3DCalibrationLogMsg()

YRefFrame

refframe→3DCalibrationLogMsg()

refframe.get_3DCalibrationLogMsg()

Retourne le dernier message de log produit par le processus de calibration.

String get_3DCalibrationLogMsg()

Si aucun nouveau message n'est disponible, retourne une chaîne vide.

Retourne :

une chaîne de caractères.

refframe→get_3DCalibrationProgress()**YRefFrame****refframe→3DCalibrationProgress()****refframe.get_3DCalibrationProgress()**

Retourne l'avancement global du processus de calibration tridimensionnelle initié avec la méthode `start3DCalibration`.

int **get_3DCalibrationProgress()**

Retourne :

une nombre entier entre 0 (pas commencé) et 100 (terminé).

refframe→get_3DCalibrationStage()

YRefFrame

refframe→3DCalibrationStage()

refframe.get_3DCalibrationStage()

Retourne l'index de l'étape courante de la calibration initiée avec la méthode start3DCalibration.

int get_3DCalibrationStage()

Retourne :

une nombre entier, croissant au fur et à mesure de la complétion des étapes.

refframe→get_3DCalibrationStageProgress()**YRefFrame****refframe→3DCalibrationStageProgress()****refframe.get_3DCalibrationStageProgress()**

Retourne l'avancement de l'étape courante de la calibration initiée avec la méthode `start3DCalibration`.

int `get_3DCalibrationStageProgress()`

Retourne :

une nombre entier entre 0 (pas commencé) et 100 (terminé).

refframe→get_advertisedValue()

YRefFrame

refframe→advertisedValue()

refframe.get_advertisedValue()

Retourne la valeur courante du référentiel (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du référentiel (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

refframe→**get_bearing()****YRefFrame****refframe**→**bearing()****refframe.get_bearing()**

Retourne le cap de référence utilisé par le compas.

double **get_bearing()**

Le cap relatif indiqué par le compas est la différence entre le Nord magnétique mesuré et le cap de référence spécifié ici.

Retourne :

une valeur numérique représentant le cap de référence utilisé par le compas

En cas d'erreur, déclenche une exception ou retourne `Y_BEARING_INVALID`.

refframe→get_errorMessage()

YRefFrame

refframe→errorMessage()

refframe.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

String get_errorMessage()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du référentiel.

refframe→get_errorType()**YRefFrame****refframe→errorType()**`refframe.errorType()`

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du référentiel.

`int` **get_errorType()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du référentiel.

refframe→get_friendlyName()

YRefFrame

refframe→friendlyName()

refframe.get_friendlyName()

Retourne un identifiant global du référentiel au format `NOM_MODULE.NOM_FONCTION`.

String get_friendlyName()

Le chaîne retournée utilise soit les noms logiques du module et du référentiel si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du référentiel (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le référentiel en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

refframe→**get_functionDescriptor()****YRefFrame****refframe**→**functionDescriptor()****refframe.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

refframe→**get_functionId()**

YRefFrame

refframe→**functionId()**

refframe.get_functionId()

Retourne l'identifiant matériel du référentiel, sans référence au module.

String **get_functionId()** ()

Par exemple relay1.

Retourne :

une chaîne de caractères identifiant le référentiel (ex: relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FUNCTIONID_INVALID.

refframe→get_hardwareId()**YRefFrame****refframe→hardwareId()****refframe.get_hardwareId()**

Retourne l'identifiant matériel unique du référentiel au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du référentiel (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le référentiel (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

refframe→get_logicalName()

YRefFrame

refframe→logicalName()

refframe.get_logicalName()

Retourne le nom logique du référentiel.

String get_logicalName()

Retourne :

une chaîne de caractères représentant le nom logique du référentiel.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

refframe→**get_module()****YRefFrame****refframe**→**module()****refframe.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

refframe→get_mountOrientation()

YRefFrame

refframe→mountOrientation()

refframe.get_mountOrientation()

Retourne l'orientation à l'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

MOUNTORIENTATION **get_mountOrientation()**

Retourne :

une valeur parmi l'énumération Y_MOUNTORIENTATION (Y_MOUNTORIENTATION_TWELVE, Y_MOUNTORIENTATION_THREE, Y_MOUNTORIENTATION_SIX, Y_MOUNTORIENTATION_NINE) correspondant à la l'orientation de la flèche "X" sur le module par rapport à un cadran d'horloge vu par un observateur au centre de la boîte. Sur la face BOTTOM le 12h pointe vers l'avant, tandis que sur la face TOP le 12h pointe vers l'arrière.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→get_mountPosition()**YRefFrame****refframe→mountPosition()****refframe.get_mountPosition()**

Retourne la position d'installation du module, telle que configurée afin de définir le référentiel de la boussole et des inclinomètres.

MOUNTPOSITION **get_mountPosition()****Retourne :**

une valeur parmi l'énumération **Y_MOUNTPOSITION** (**Y_MOUNTPOSITION_BOTTOM**, **Y_MOUNTPOSITION_TOP**, **Y_MOUNTPOSITION_FRONT**, **Y_MOUNTPOSITION_RIGHT**, **Y_MOUNTPOSITION_REAR**, **Y_MOUNTPOSITION_LEFT**), correspondant à l'installation dans une boîte, sur l'une des six faces

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→**get_userdata()**

YRefFrame

refframe→**userData()****refframe.get_userdata()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

refframe→**isOnline()****refframe.isOnline()****YRefFrame**

Vérifie si le module hébergeant le référentiel est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache du référentiel sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le référentiel est joignable, `false` sinon

refframe→**load()****refframe.load()****YRefFrame**

Met en cache les valeurs courantes du référentiel, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→more3DCalibration()**YRefFrame****refframe.more3DCalibration()**

Continue le processus de calibration tridimensionnelle des capteurs initié avec la méthode `start3DCalibration`.

int more3DCalibration()

Cette méthode doit être appelée environ 5 fois par secondes après avoir positionné le module selon les instructions fournies par la méthode `get_3DCalibrationHint` (les instructions changent pendant la procédure de calibration). En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→**nextRefFrame()**

YRefFrame

refframe.nextRefFrame()

Continue l'énumération des référentiels commencée à l'aide de `yFirstRefFrame()`.

`YRefFrame` **nextRefFrame()**

Retourne :

un pointeur sur un objet `YRefFrame` accessible en ligne, ou `null` lorsque l'énumération est terminée.

refframe→registerValueCallback()**YRefFrame****refframe.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

refframe→save3DCalibration()

YRefFrame

refframe.save3DCalibration()

Applique les paramètres de calibration tridimensionnelle précédemment calculés.

int save3DCalibration()

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé après le redémarrage du module. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→**set_bearing()****YRefFrame****refframe**→**setBearing()****refframe.set_bearing()**

Modifie le cap de référence utilisé par le compas.

```
int set_bearing( double newval)
```

Le cap relatif indiqué par le compas est la différence entre le Nord magnétique mesuré et le cap de référence spécifié ici. Par exemple, si vous indiquez comme cap de référence la valeur de la déclinaison magnétique terrestre, le compas donnera l'orientation par rapport au Nord géographique. De même, si le capteur n'est pas positionné dans une des directions standard à cause d'un angle de lacet supplémentaire, vous pouvez le configurer comme cap de référence afin que le compas donne la direction naturelle attendue.

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur numérique représentant le cap de référence utilisé par le compas

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→**set_logicalName()****YRefFrame****refframe**→**setLogicalName()****refframe.set_logicalName()**

Modifie le nom logique du référentiel.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du référentiel.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

refframe→set_mountPosition()**YRefFrame****refframe→setMountPosition()****refframe.set_mountPosition()**

Modifie le référentiel de la boussole et des inclinomètres.

```
int set_mountPosition( MOUNTPOSITION position,
                      MOUNTORIENTATION orientation)
```

La boussole magnétique et les inclinomètres gravitationnels fonctionnent par rapport au plan parallèle à la surface terrestre. Dans les cas où le module n'est pas utilisé horizontalement et à l'endroit, il faut indiquer son orientation de référence (parallèle à la surface terrestre) afin que les mesures soient faites relativement à cette position.

Paramètres :

position une valeur parmi l'énumération Y_MOUNTPOSITION (Y_MOUNTPOSITION_BOTTOM, Y_MOUNTPOSITION_TOP, Y_MOUNTPOSITION_FRONT, Y_MOUNTPOSITION_RIGHT, Y_MOUNTPOSITION_REAR, Y_MOUNTPOSITION_LEFT), correspondant à l'installation dans une boîte, sur l'une des six faces.

orientation une valeur parmi l'énumération Y_MOUNTORIENTATION (Y_MOUNTORIENTATION_TWELVE, Y_MOUNTORIENTATION_THREE, Y_MOUNTORIENTATION_SIX, Y_MOUNTORIENTATION_NINE) correspondant à la l'orientation de la flèche "X" sur le module par rapport à un cadran d'horloge vu par un observateur au centre de la boîte. Sur la face BOTTOM le 12h pointe vers l'avant, tandis que sur la face TOP le 12h pointe vers l'arrière. N'oubliez pas d'appeler la méthode saveToFlash() du module si le réglage doit être préservé.

refframe→set_userdata()

YRefFrame

refframe→setUserData()**refframe.set_userdata()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

void **set_userdata**(Object **data**)

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

refframe→start3DCalibration()**YRefFrame****refframe.start3DCalibration()**

Initie le processus de calibration tridimensionnelle des capteurs.

int start3DCalibration()

Cette calibration est utilisée à bas niveau pour l'estimation inertielle de position et pour améliorer la précision des mesures d'inclinaison. Après avoir appelé cette méthode, il faut positionner le module selon les instructions fournies par la méthode `get_3DCalibrationHint` et appeler `more3DCalibration` environ 5 fois par secondes. La procédure de calibration est terminée lorsque la méthode `get_3DCalibrationProgress` retourne 100. Il est alors possible d'appliquer les paramètres calculés, à l'aide de la méthode `save3DCalibration`. A tout moment, la calibration peut être abandonnée à l'aide de `cancel3DCalibration`. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.38. Interface de la fonction Relay

La librairie de programmation Yoctopuce permet simplement de changer l'état du relais. Le changement d'état n'est pas persistant: le relais retournera spontanément à sa position de repos dès que le module est mis hors tension ou redémarré. La librairie permet aussi de créer des courtes impulsions de durée déterminée. Pour les modules dotés de deux sorties par relais (relai inverseur), les deux sorties sont appelées A et B, la sortie A correspondant à la position de repos (hors tension) et la sortie B correspondant à l'état actif. Si vous préférez l'état par défaut opposé, vous pouvez simplement changer vos fils sur le bornier.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_relay.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YRelay = yoctolib.YRelay;
php	require_once('yocto_relay.php');
c++	#include "yocto_relay.h"
m	#import "yocto_relay.h"
pas	uses yocto_relay;
vb	yocto_relay.vb
cs	yocto_relay.cs
java	import com.yoctopuce.YoctoAPI.YRelay;
py	from yocto_relay import *

Fonction globales

yFindRelay(func)

Permet de retrouver un relais d'après un identifiant donné.

yFirstRelay()

Commence l'énumération des relais accessibles par la librairie.

Méthodes des objets YRelay

relay→delayedPulse(ms_delay, ms_duration)

Pré-programme une impulsion

relay→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du relais au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

relay→get_advertisedValue()

Retourne la valeur courante du relais (pas plus de 6 caractères).

relay→get_countdown()

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à `delayedPulse()`.

relay→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du relais.

relay→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du relais.

relay→get_friendlyName()

Retourne un identifiant global du relais au format `NOM_MODULE . NOM_FONCTION`.

relay→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

relay→get_functionId()

Retourne l'identifiant matériel du relais, sans référence au module.

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 une 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)

3. Reference

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

relay→set_output(newval)

Modifie l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

relay→set_state(newval)

Modifie l'état du relais (A pour la position de repos, B pour l'état actif).

relay→set_stateAtPowerOn(newval)

Pré-programme l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

relay→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userData.

relay→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YRelay.FindRelay()**YRelay****yFindRelay()**`YRelay.FindRelay()`

Permet de retrouver un relais d'après un identifiant donné.

`YRelay FindRelay(String func)`

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le relais soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRelay.isOnline()` pour tester si le relais est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le relais sans ambiguïté

Retourne :

un objet de classe `YRelay` qui permet ensuite de contrôler le relais.

YRelay.FirstRelay()

YRelay

yFirstRelay()YRelay.FirstRelay()

Commence l'énumération des relais accessibles par la librairie.

YRelay **FirstRelay**()

Utiliser la fonction `YRelay.nextRelay()` pour itérer sur les autres relais.

Retourne :

un pointeur sur un objet `YRelay`, correspondant au premier relais accessible en ligne, ou `null` si il n'y a pas de relais disponibles.

relay→**delayedPulse()****relay.delayedPulse()**

YRelay

Pré-programme une impulsion

```
int delayedPulse( int ms_delay, int ms_duration)
```

Paramètres :

ms_delay délai d'attente avant l'impulsion, en millisecondes

ms_duration durée de l'impulsion, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→describe()**relay.describe()****YRelay**

Retourne un court texte décrivant de manière non-ambigüe l'instance du relais au format `TYPE (NAME) = SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le relais (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

relay→get_advertisedValue()
relay→advertisedValue()
relay.get_advertisedValue()

YRelay

Retourne la valeur courante du relais (pas plus de 6 caractères).

String get_advertisedValue()

Retourne :

une chaîne de caractères représentant la valeur courante du relais (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

relay→**get_countdown()**

YRelay

relay→**countdown()****relay.get_countdown()**

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à `delayedPulse()`.

`long` **get_countdown()**

Si aucune impulsion n'est programmée, retourne zéro.

Retourne :

un entier représentant le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à `delayedPulse()`

En cas d'erreur, déclenche une exception ou retourne `Y_COUNTDOWN_INVALID`.

relay→**get_errorMessage()****YRelay****relay**→**errorMessage()****relay.errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du relais.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du relais.

relay→get_errorType()

YRelay

relay→errorType()`relay.get_errorType()`

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du relais.

`int get_errorType()`

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du relais.

relay→**get_friendlyName()****YRelay****relay**→**friendlyName()****relay.get_friendlyName()**

Retourne un identifiant global du relais au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du relais si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du relais (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le relais en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

relay→**get_functionDescriptor()**

YRelay

relay→**functionDescriptor()**

relay.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

relay→**get_functionId()****YRelay****relay**→**functionId()****relay.get_functionId()**

Retourne l'identifiant matériel du relais, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le relais (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

relay→**get_hardwareId()**

YRelay

relay→**hardwareId()****relay.get_hardwareId()**

Retourne l'identifiant matériel unique du relais au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du relais (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le relais (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

relay→**get_logicalName()****YRelay****relay**→**logicalName()****relay.get_logicalName()**

Retourne le nom logique du relais.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du relais.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

relay→**get_maxTimeOnStateA()**

YRelay

relay→**maxTimeOnStateA()**

relay.get_maxTimeOnStateA()

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

long get_maxTimeOnStateA()

Zéro signifie qu'il n'y a pas de limitation

Retourne :

un entier représentant le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B

En cas d'erreur, déclenche une exception ou retourne Y_MAXTIMEONSTATEA_INVALID.

relay→**get_maxTimeOnStateB()****YRelay****relay**→**maxTimeOnStateB()****relay.get_maxTimeOnStateB()**

Retourne le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
long get_maxTimeOnStateB( )
```

Zéro signifie qu'il n'y a pas de limitation

Retourne :

un entier représentant le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A

En cas d'erreur, déclenche une exception ou retourne `Y_MAXTIMEONSTATEB_INVALID`.

relay→**get_module()**

YRelay

relay→**module()****relay.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

relay→**get_output()****YRelay****relay**→**output()****relay.get_output()**

Retourne l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

int **get_output()**

Retourne :

soit Y_OUTPUT_OFF, soit Y_OUTPUT_ON, selon l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur

En cas d'erreur, déclenche une exception ou retourne Y_OUTPUT_INVALID.

relay→**get_pulseTimer()**

YRelay

relay→**pulseTimer()****relay.get_pulseTimer()**

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

long **get_pulseTimer()**

Si aucune impulsion n'est en cours, retourne zéro.

Retourne :

un entier représentant le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée

En cas d'erreur, déclenche une exception ou retourne Y_PULSETIMER_INVALID.

relay→**get_state()****YRelay****relay**→**state()****relay.get_state()**

Retourne l'état du relais (A pour la position de repos, B pour l'état actif).

int **get_state()**

Retourne :

soit Y_STATE_A, soit Y_STATE_B, selon l'état du relais (A pour la position de repos, B pour l'état actif)

En cas d'erreur, déclenche une exception ou retourne Y_STATE_INVALID.

relay→**get_stateAtPowerOn()**

YRelay

relay→**stateAtPowerOn()**

relay.get_stateAtPowerOn()

Retourne l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

int **get_stateAtPowerOn()**

Retourne :

une valeur parmi Y_STATEATPOWERON_UNCHANGED, Y_STATEATPOWERON_A et Y_STATEATPOWERON_B représentant l'état du relais au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement)

En cas d'erreur, déclenche une exception ou retourne Y_STATEATPOWERON_INVALID.

relay→**get_userData()****YRelay****relay**→**userData()****relay.userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

Object **get_userData()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

relay→**isOnline()**`relay.isOnline()`

YRelay

Vérifie si le module hébergeant le relais est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du relais sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le relais est joignable, `false` sinon

relay→**load()****relay.load()****YRelay**

Met en cache les valeurs courantes du relais, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**nextRelay()****relay.nextRelay()**

YRelay

Continue l'énumération des relais commencée à l'aide de `yFirstRelay()`.

YRelay **nextRelay()**

Retourne :

un pointeur sur un objet YRelay accessible en ligne, ou `null` lorsque l'énumération est terminée.

relay→**pulse()****relay.pulse()****YRelay**

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

```
int pulse( int ms_duration)
```

Paramètres :

ms_duration durée de l'impulsion, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**registerValueCallback()****YRelay****relay.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

relay→**set_logicalName()****YRelay****relay**→**setLogicalName()****relay.set_logicalName()**

Modifie le nom logique du relais.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du relais.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**set_maxTimeOnStateA()**

YRelay

relay→**setMaxTimeOnStateA()**

relay.set_maxTimeOnStateA()

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état A avant de basculer automatiquement dans l'état B.

```
int set_maxTimeOnStateA( long newval)
```

Zéro signifie qu'il n'y a pas de limitation

Paramètres :

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**set_maxTimeOnStateB()****YRelay****relay**→**setMaxTimeOnStateB()****relay.set_maxTimeOnStateB()**

Règle le temps maximal (en ms) pendant lequel le relais peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
int set_maxTimeOnStateB( long newval)
```

Zéro signifie qu'il n'y a pas de limitation

Paramètres :

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**set_output()**

YRelay

relay→**setOutput()****relay.set_output()**

Modifie l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

```
int set_output( int newval)
```

Paramètres :

newval soit Y_OUTPUT_OFF, soit Y_OUTPUT_ON, selon l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**set_state()****YRelay****relay**→**setState()****relay.set_state()**

Modifie l'état du relais (A pour la position de repos, B pour l'état actif).

```
int set_state( int newval)
```

Paramètres :

newval soit Y_STATE_A, soit Y_STATE_B, selon l'état du relais (A pour la position de repos, B pour l'état actif)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**set_stateAtPowerOn()**

YRelay

relay→**setStateAtPowerOn()**

relay.set_stateAtPowerOn()

Pré-programme l'état du relais au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
int set_stateAtPowerOn( int newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval une valeur parmi Y_STATEATPOWERON_UNCHANGED, Y_STATEATPOWERON_A et Y_STATEATPOWERON_B

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

relay→**set_userdata()****YRelay****relay**→**setUserData()****relay.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

3.39. Interface des fonctions de type senseur

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_api.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YAPI = yoctolib.YAPI;</code> <code>var YModule = yoctolib.YModule;</code>
php	<code>require_once('yocto_api.php');</code>
c++	<code>#include "yocto_api.h"</code>
m	<code>#import "yocto_api.h"</code>
pas	<code>uses yocto_api;</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>

Fonction globales

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 mesure, en l'unité spécifiée, sous forme de nombre à virgule.

sensor→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

sensor→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

sensor→get_friendlyName()

Retourne un identifiant global du senseur au format `NOM_MODULE . NOM_FONCTION`.

sensor→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

sensor→get_functionId()

Retourne l'identifiant matériel du senseur, sans référence au module.

sensor→get_hardwareId()

Retourne l'identifiant matériel unique du capteur au format `SERIAL.FUNCTIONID`.

sensor→get_highestValue()

Retourne la valeur maximale observée pour la mesure 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 capteur.

sensor→get_lowestValue()

Retourne la valeur minimale observée pour la mesure 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 mesure 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 capteur est joignable, sans déclencher d'erreur.

sensor→isOnline_async(callback, context)

Vérifie si le module hébergeant le capteur est joignable, sans déclencher d'erreur.

sensor→load(msValidity)

Met en cache les valeurs courantes du capteur, 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 capteur, avec une durée de validité spécifiée.

sensor→nextSensor()

Continue l'énumération des capteurs 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)

3. Reference

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

sensor→**set_logicalName**(newval)

Modifie le nom logique du capteur.

sensor→**set_lowestValue**(newval)

Modifie la mémoire de valeur minimale observée.

sensor→**set_reportFrequency**(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

sensor→**set_resolution**(newval)

Modifie la résolution des valeurs physiques mesurées.

sensor→**set_userData**(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

sensor→**wait_async**(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YSensor.FindSensor()**YSensor****yFindSensor()**`YSensor.FindSensor()`

Permet de retrouver un senseur d'après un identifiant donné.

`YSensor FindSensor(String func)`

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le senseur soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YSensor.isOnline()` pour tester si le senseur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le senseur sans ambiguïté

Retourne :

un objet de classe `YSensor` qui permet ensuite de contrôler le senseur.

YSensor.FirstSensor()

YSensor

yFirstSensor()`YSensor.FirstSensor()`

Commence l'énumération des senseurs accessibles par la librairie.

`YSensor` **FirstSensor()**

Utiliser la fonction `YSensor.nextSensor()` pour itérer sur les autres senseurs.

Retourne :

un pointeur sur un objet `YSensor`, correspondant au premier senseur accessible en ligne, ou `null` si il n'y a pas de senseurs disponibles.

sensor→calibrateFromPoints()**YSensor****sensor.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                        ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**describe()****sensor.describe()****YSensor**

Retourne un court texte décrivant de manière non-ambigüe l'instance du senseur au format
`TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le senseur (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

sensor→**get_advertisedValue()****YSensor****sensor**→**advertisedValue()****sensor.get_advertisedValue()**

Retourne la valeur courante du senseur (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du senseur (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

sensor→**get_currentRawValue()**

YSensor

sensor→**currentRawValue()**

sensor.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule.

double **get_currentRawValue()**

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en l'unité spécifiée, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

sensor→**get_currentValue()****YSensor****sensor**→**currentValue()****sensor.get_currentValue()**

Retourne la valeur actuelle de la mesure, en l'unité spécifiée, sous forme de nombre à virgule.

double **get_currentValue()**

Retourne :

une valeur numérique représentant la valeur actuelle de la mesure, en l'unité spécifiée, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

sensor→get_errorMessage()

YSensor

sensor→errorMessage()

sensor.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du senseur.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du senseur.

sensor→**get_errorType()****YSensor****sensor**→**errorType()****sensor.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur.

int **get_errorType()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur.

sensor→**get_friendlyName()**

YSensor

sensor→**friendlyName()**

sensor.get_friendlyName()

Retourne un identifiant global du senseur au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du senseur si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du senseur (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le senseur en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

sensor→**get_functionDescriptor()**
sensor→**functionDescriptor()**
sensor.get_functionDescriptor()

YSensor

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

sensor→**get_functionId()**

YSensor

sensor→**functionId()****sensor.get_functionId()**

Retourne l'identifiant matériel du senseur, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le senseur (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

sensor→**get_hardwareId()****YSensor****sensor**→**hardwareId()****sensor.get_hardwareId()**

Retourne l'identifiant matériel unique du senseur au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du senseur (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le senseur (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

sensor→**get_highestValue()**

YSensor

sensor→**highestValue()**

sensor.get_highestValue()

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

double **get_highestValue()**

Retourne :

une valeur numérique représentant la valeur maximale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

sensor→**get_logFrequency()****YSensor****sensor**→**logFrequency()****sensor.get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

String **get_logFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

sensor→**get_logicalName()**

YSensor

sensor→**logicalName()**`sensor.get_logicalName()`

Retourne le nom logique du senseur.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du senseur.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

sensor→**get_lowestValue()****YSensor****sensor**→**lowestValue()****sensor.get_lowestValue()**

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

double **get_lowestValue()**

Retourne :

une valeur numérique représentant la valeur minimale observée pour la mesure depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_LOWESTVALUE_INVALID.

sensor→**get_module()**

YSensor

sensor→**module()**`sensor.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` [get_module\(\)](#)

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

sensor→**get_recordedData()**
sensor→**recordedData()**
sensor.get_recordedData()

YSensor

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

YDataSet **get_recordedData(long startTime, long endTime)**

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

sensor→get_reportFrequency()

YSensor

sensor→reportFrequency()

sensor.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String **get_reportFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

sensor→**get_resolution()****YSensor****sensor**→**resolution()****sensor.get_resolution()**

Retourne la résolution des valeurs mesurées.

double **get_resolution()** ()

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_RESOLUTION_INVALID.

sensor→**get_unit()**

YSensor

sensor→**unit()**`sensor.get_unit()`

Retourne l'unité dans laquelle la mesure est exprimée.

String **get_unit()**

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la mesure est exprimée

En cas d'erreur, déclenche une exception ou retourne `Y_UNIT_INVALID`.

sensor→**get_userData()****YSensor****sensor**→**userData()****sensor.get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

Object **get_userData()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

sensor→**isOnline()****sensor.isOnline()**

YSensor

Vérifie si le module hébergeant le senseur est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache du senseur sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le senseur est joignable, `false` sinon

sensor→**load()****sensor.load()****YSensor**

Met en cache les valeurs courantes du senseur, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→loadCalibrationPoints()**YSensor****sensor.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**nextSensor()****sensor.nextSensor()****YSensor**

Continue l'énumération des senseurs commencée à l'aide de `yFirstSensor()`.

`YSensor` **nextSensor()**

Retourne :

un pointeur sur un objet `YSensor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

sensor→registerTimedReportCallback()**YSensor****sensor.registerTimedReportCallback()**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

sensor→registerValueCallback()**YSensor****sensor.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

sensor→**set_highestValue()**

YSensor

sensor→**setHighestValue()**

sensor.set_highestValue()

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**set_logFrequency()**
sensor→**setLogFrequency()**
sensor.set_logFrequency()

YSensor

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

int **set_logFrequency**(String **newval**)

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**set_logicalName()****YSensor****sensor**→**setLogicalName()****sensor.set_logicalName()**

Modifie le nom logique du senseur.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du senseur.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**set_lowestValue()**
sensor→**setLowestValue()**
sensor.set_lowestValue()

YSensor

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**set_reportFrequency()****YSensor****sensor**→**setReportFrequency()****sensor.set_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**set_resolution()****YSensor****sensor**→**setResolution()****sensor.set_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

sensor→**set_userdata()**

YSensor

sensor→**setUserData()****sensor.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

`void set_userdata(Object data)`

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

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');
c++	#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_DESCRIPTOR` 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 en lecture 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 protocole 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)

3. Reference

Envoie une chaîne de caractères sur le port série, suivie d'un saut de ligne (CR LF).

serialport→**writeMODBUS(hexString)**

Envoie une commande MODBUS (fournie sous forme de chaîne hexadécimale) sur le port série.

serialport→**writeStr(text)**

Envoie une chaîne de caractères telle quelle sur le port série.

YSerialPort.FindSerialPort()**YSerialPort****yFindSerialPort()**`YSerialPort.FindSerialPort()`

Permet de retrouver une port série d'après un identifiant donné.

`YSerialPort FindSerialPort(String func)`

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le port série soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YSerialPort.isOnline()` pour tester si le port série est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le port série sans ambiguïté

Retourne :

un objet de classe `YSerialPort` qui permet ensuite de contrôler le port série.

YSerialPort.FirstSerialPort()

YSerialPort

yFirstSerialPort()

YSerialPort.FirstSerialPort()

Commence l'énumération des le port série accessibles par la librairie.

[YSerialPort](#) **FirstSerialPort()**

Utiliser la fonction `YSerialPort.nextSerialPort()` pour itérer sur les autres le port série.

Retourne :

un pointeur sur un objet `YSerialPort`, correspondant au premier port série accessible en ligne, ou `null` si il n'y a pas du port série disponibles.

serialport→describe()`serialport.describe()`**YSerialPort**

Retourne un court texte décrivant de manière non-ambigüe l'instance du port série au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String `describe()`

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le port série (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

serialport→**get_CTS()**

YSerialPort

serialport→**CTS()****serialport.get_CTS()**

Lit l'état de la ligne CTS.

int **get_CTS()**

La ligne CTS est habituellement pilotée par le signal RTS du périphérique série connecté.

Retourne :

1 si le CTS est signalé (niveau haut), 0 si le CTS n'est pas actif (niveau bas).

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→**get_advertisedValue()****YSerialPort****serialport**→**advertisedValue()****serialport.get_advertisedValue()**

Retourne la valeur courante du port série (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du port série (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

serialport→**get_errCount()**

YSerialPort

serialport→**errCount()**

serialport.get_errCount()

Retourne le nombre d'erreurs de communication détectées depuis la dernière mise à zéro.

int **get_errCount()**

Retourne :

un entier représentant le nombre d'erreurs de communication détectées depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne `Y_ERRCOUNT_INVALID`.

serialport→get_errorMessage()**YSerialPort****serialport→errorMessage()****serialport.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port série.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du port série.

serialport→get_errorType()

YSerialPort

serialport→errorType()

serialport.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du port série.

```
int get_errorType()
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du port série.

serialport→**get_friendlyName()****YSerialPort****serialport**→**friendlyName()****serialport.get_friendlyName()**

Retourne un identifiant global du port série au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du port série si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du port série (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le port série en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

serialport→**get_functionDescriptor()**

YSerialPort

serialport→**functionDescriptor()**

serialport.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

serialport→**get_functionId()****YSerialPort****serialport**→**functionId()****serialport.get_functionId()**

Retourne l'identifiant matériel du port série, sans référence au module.

String **get_functionId()** ()

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le port série (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

serialport→**get_hardwareId()**

YSerialPort

serialport→**hardwareId()**

serialport.get_hardwareId()

Retourne l'identifiant matériel unique du port série au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()** ()

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du port série (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le port série (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

serialport→**get_lastMsg()****YSerialPort****serialport**→**lastMsg()****serialport.get_lastMsg()**

Retourne le dernier message reçu (pour les protocoles de type Line, Frame et Modbus).

String **get_lastMsg()**

Retourne :

une chaîne de caractères représentant le dernier message reçu (pour les protocoles de type Line, Frame et Modbus)

En cas d'erreur, déclenche une exception ou retourne `Y_LASTMSG_INVALID`.

serialport→**get_logicalName()**

YSerialPort

serialport→**logicalName()**

serialport.get_logicalName()

Retourne le nom logique du port série.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du port série.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

serialport→**get_module()****YSerialPort****serialport**→**module()**`serialport.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

serialport→get_msgCount()

YSerialPort

serialport→msgCount()

serialport.get_msgCount()

Retourne le nombre de messages reçus depuis la dernière mise à zéro.

int get_msgCount()

Retourne :

un entier représentant le nombre de messages reçus depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne Y_MSGCOUNT_INVALID.

serialport→**get_protocol()****YSerialPort****serialport**→**protocol()**`serialport.get_protocol()`

Retourne le type de protocole utilisé sur la communication série, sous forme d'une chaîne de caractères.

String **get_protocol()**

Les valeurs possibles sont "Line" pour des messages ASCII séparés par des retours de ligne, "Frame:[timeout]ms" pour des messages binaires séparés par une temporisation, "Modbus-ASCII" pour des messages MODBUS en mode ASCII, "Modbus-RTU" pour des messages MODBUS en mode RTU, "Char" pour un flux ASCII continu ou "Byte" pour un flux binaire continue.

Retourne :

une chaîne de caractères représentant le type de protocole utilisé sur la communication série, sous forme d'une chaîne de caractères

En cas d'erreur, déclenche une exception ou retourne `Y_PROTOCOL_INVALID`.

serialport→get_rxCount()

YSerialPort

serialport→rxCount()`serialport.get_rxCount()`

Retourne le nombre d'octets reçus depuis la dernière mise à zéro.

`int get_rxCount()`

Retourne :

un entier représentant le nombre d'octets reçus depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne `Y_RXCOUNT_INVALID`.

serialport→get_serialMode()
serialport→serialMode()
serialport.get_serialMode()

YSerialPort

Retourne les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1".

String **get_serialMode()** ()

La chaîne contient le taux de transfert, le nombre de bits de données, la parité et le nombre de bits d'arrêt. Un suffixe supplémentaire optionnel est inclus si une option de contrôle de flux est active: "CtsRts" pour le contrôle de flux matériel, "XOnXOff" pour le contrôle de flux logique et "Simplex" pour l'utilisation du signal RTS pour l'acquisition d'un bus partagé (tel qu'utilisé pour certains adaptateurs RS485 par exemple).

Retourne :

une chaîne de caractères représentant les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1"

En cas d'erreur, déclenche une exception ou retourne Y_SERIALMODE_INVALID.

serialport→get_txCount()

YSerialPort

serialport→txCount()`serialport.get_txCount()`

Retourne le nombre d'octets transmis depuis la dernière mise à zéro.

`int get_txCount()`

Retourne :

un entier représentant le nombre d'octets transmis depuis la dernière mise à zéro

En cas d'erreur, déclenche une exception ou retourne `Y_TXCOUNT_INVALID`.

serialport→get_userdata()**YSerialPort****serialport→userData()****serialport.get_userdata()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

serialport→**isOnline()****serialport.isOnline()**

YSerialPort

Vérifie si le module hébergeant le port série est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du port série sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

true si le port série est joignable, false sinon

serialport→load()`serialport.load()`**YSerialPort**

Met en cache les valeurs courantes du port série, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→modbusReadBits()**YSerialPort****serialport.modbusReadBits()**

Lit un ou plusieurs bits contigus depuis un périphérique MODBUS.

```
ArrayList<Integer> modbusReadBits( int slaveNo,  
                                   int pduAddr,  
                                   int nBits)
```

Cette méthode utilise le code de fonction MODBUS 0x01 (Read Coils).

Paramètres :

- slaveNo** adresse du périphérique MODBUS esclave à interroger
- pduAddr** adresse relative du premier bit à lire (indexé à partir de zéro).
- nBits** nombre de bits à lire

Retourne :

un vecteur d'entiers, correspondant chacun à un bit.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

serialport→modbusReadInputBits()**YSerialPort****serialport.modbusReadInputBits()**

Lit un ou plusieurs bits contigus depuis un périphérique MODBUS.

```
ArrayList<Integer> modbusReadInputBits( int slaveNo,  
                                         int pduAddr,  
                                         int nBits)
```

Cette méthode utilise le code de fonction MODBUS 0x02 (Read Discrete Inputs).

Paramètres :

slaveNo adresse du périphérique MODBUS esclave à interroger
pduAddr adresse relative du premier bit à lire (indexé à partir de zéro).
nBits nombre de bits à lire

Retourne :

un vecteur d'entiers, correspondant chacun à un bit.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

serialport→modbusReadInputRegisters()**YSerialPort****serialport.modbusReadInputRegisters()**

Lit un ou plusieurs registres d'entrée (registre en lecture seule) depuis un périphérique MODBUS.

```
ArrayList<Integer> modbusReadInputRegisters( int slaveNo,  
                                             int pduAddr,  
                                             int nWords)
```

Cette méthode utilise le code de fonction MODBUS 0x04 (Read Input Registers).

Paramètres :

- slaveNo** adresse du périphérique MODBUS esclave à interroger
- pduAddr** adresse relative du premier registre d'entrée à lire (indexé à partir de zéro).
- nWords** nombre de registres d'entrée à lire

Retourne :

un vecteur d'entiers, correspondant chacun à une valeur d'entrée (16 bits).

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

serialport→modbusReadRegisters()**YSerialPort****serialport.modbusReadRegisters()**

Lit un ou plusieurs registres interne depuis un périphérique MODBUS.

```
ArrayList<Integer> modbusReadRegisters( int slaveNo,  
                                         int pduAddr,  
                                         int nWords)
```

Cette méthode utilise le code de fonction MODBUS 0x03 (Read Holding Registers).

Paramètres :

- slaveNo** adresse du périphérique MODBUS esclave à interroger
- pduAddr** adresse relative du premier registre interne à lire (indexé à partir de zéro).
- nWords** nombre de registres internes à lire

Retourne :

un vecteur d'entiers, correspondant chacun à une valeur de registre (16 bits).

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

serialport→modbusWriteAndReadRegisters()**YSerialPort****serialport.modbusWriteAndReadRegisters()**

Modifie l'état de plusieurs bits (ou relais) contigus sur un périphérique MODBUS.

```
ArrayList<Integer> modbusWriteAndReadRegisters( int slaveNo,  
                                                int pduWriteAddr,  
                                                ArrayList<Integer> values,  
                                                int pduReadAddr,  
                                                int nReadWords)
```

Cette méthode utilise le code de fonction MODBUS 0x17 (Read/Write Multiple Registers).

Paramètres :

slaveNo adresse du périphérique MODBUS esclave à piloter
pduWriteAddr adresse relative du premier registre interne à modifier (indexé à partir de zéro).
values vecteur de valeurs 16 bits à appliquer
pduReadAddr adresse relative du premier registre interne à lire (indexé à partir de zéro).
nReadWords nombre de registres internes à lire

Retourne :

un vecteur d'entiers, correspondant chacun à une valeur de registre (16 bits) lue.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

serialport→modbusWriteBit()**YSerialPort****serialport.modbusWriteBit()**

Modifie l'état d'un seul bit (ou relais) sur un périphérique MODBUS.

```
int modbusWriteBit( int slaveNo, int pduAddr, int value)
```

Cette méthode utilise le code de fonction MODBUS 0x05 (Write Single Coil).

Paramètres :

- slaveNo** adresse du périphérique MODBUS esclave à piloter
- pduAddr** adresse relative du bit à modifier (indexé à partir de zéro).
- value** la valeur à appliquer (0 pour l'état OFF, non-zéro pour l'état ON)

Retourne :

le nombre de bits affectés sur le périphérique (1)

En cas d'erreur, déclenche une exception ou retourne zéro.

serialport→modbusWriteBits()**YSerialPort****serialport.modbusWriteBits()**

Modifie l'état de plusieurs bits (ou relais) contigus sur un périphérique MODBUS.

```
int modbusWriteBits( int slaveNo,  
                    int pduAddr,  
                    ArrayList<Integer> bits)
```

Cette méthode utilise le code de fonction MODBUS 0x0f (Write Multiple Coils).

Paramètres :

- slaveNo** adresse du périphérique MODBUS esclave à piloter
- pduAddr** adresse relative du premier bit à modifier (indexé à partir de zéro).
- bits** vecteur de bits à appliquer (un entier par bit)

Retourne :

le nombre de bits affectés sur le périphérique

En cas d'erreur, déclenche une exception ou retourne zéro.

serialport→modbusWriteRegister()**YSerialPort****serialport.modbusWriteRegister()**

Modifie la valeur d'un registre interne 16 bits sur un périphérique MODBUS.

```
int modbusWriteRegister( int slaveNo, int pduAddr, int value)
```

Cette méthode utilise le code de fonction MODBUS 0x06 (Write Single Register).

Paramètres :

- slaveNo** adresse du périphérique MODBUS esclave à piloter
- pduAddr** adresse relative du registre à modifier (indexé à partir de zéro).
- value** la valeur 16 bits à appliquer

Retourne :

le nombre de registres affectés sur le périphérique (1)

En cas d'erreur, déclenche une exception ou retourne zéro.

serialport→modbusWriteRegisters()**YSerialPort****serialport.modbusWriteRegisters()**

Modifie l'état de plusieurs registres internes 16 bits contigus sur un périphérique MODBUS.

```
int modbusWriteRegisters( int slaveNo,  
                          int pduAddr,  
                          ArrayList<Integer> values)
```

Cette méthode utilise le code de fonction MODBUS 0x10 (Write Multiple Registers).

Paramètres :

- slaveNo** adresse du périphérique MODBUS esclave à piloter
- pduAddr** adresse relative du premier registre interne à modifier (indexé à partir de zéro).
- values** vecteur de valeurs 16 bits à appliquer

Retourne :

le nombre de registres affectés sur le périphérique

En cas d'erreur, déclenche une exception ou retourne zéro.

serialport→**nextSerialPort()****YSerialPort****serialport.nextSerialPort()**

Continue l'énumération des le port série commencée à l'aide de `yFirstSerialPort()`.

`YSerialPort` **nextSerialPort()**

Retourne :

un pointeur sur un objet `YSerialPort` accessible en ligne, ou `null` lorsque l'énumération est terminée.

serialport→**queryLine()****serialport.queryLine()****YSerialPort**

Envoie un message sous forme de ligne de texte sur le port série, et lit la réponse reçue.

String queryLine(String query, int maxWait)

Cette fonction ne peut être utilisée que lorsque le module est configuré en protocole 'Line'.

Paramètres :

query le message à envoyer (sans le retour de chariot)

maxWait le temps maximum d'attente pour obtenir une réponse (en millisecondes).

Retourne :

la première ligne de texte reçue après l'envoi du message. Les lignes suivantes peuvent être obtenues avec des appels à `readLine` ou `readMessages`.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→queryMODBUS()**YSerialPort****serialport.queryMODBUS()**

Envoie un message à un périphérique MODBUS esclave connecté au port série, et lit la réponse reçue.

```
ArrayList<Integer> queryMODBUS( int slaveNo,  
                                ArrayList<Integer> pduBytes)
```

Le contenu du message est le PDU, fourni sous forme de vecteur d'octets.

Paramètres :

slaveNo adresse du périphérique MODBUS esclave

pduBytes message à envoyer (PDU), sous forme de vecteur d'octets. Le premier octet du PDU est le code de fonction MODBUS.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un tableau vide (ou une réponse d'erreur).

serialport→**readHex()****serialport.readHex()****YSerialPort**

Lit le contenu du tampon de réception sous forme hexadécimale, à partir de la position courante dans le flux de donnée.

String **readHex**(int **nBytes**)

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction ne retournera que les données disponibles.

Paramètres :

nBytes le nombre maximal d'octets à lire

Retourne :

une chaîne de caractère avec le contenu du tampon de réception, encodé en hexadécimal

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→readLine()`serialport.readLine()`**YSerialPort**

Lit la prochaine ligne (ou le prochain message) du tampon de réception, à partir de la position courante dans le flux de donnée.

String **readLine()**

Cette fonction ne peut être utilisée que lorsque le module est configuré pour gérer un protocole basé message, comme en mode 'Line' ou en protocole MODBUS. Elle ne fonctionne pas dans les modes de flux continu ('Char' et 'Byte'), pour lesquels le début d'un message n'est pas défini.

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction retournera la plus ancienne ligne disponible et déplacera le pointeur de position juste après. Si aucune nouvelle ligne entière n'est disponible, la fonction retourne un chaîne vide.

Retourne :

une chaîne de caractère avec une ligne de texte

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→readMessages()**YSerialPort****serialport.readMessages()**

Cherche les messages entrants dans le tampon de réception correspondant à un format donné, à partir de la position courante.

```
ArrayList<String> readMessages( String pattern, int maxWait)
```

Cette fonction ne peut être utilisée que lorsque le module est configuré pour gérer un protocole basé message, comme en mode 'Line' ou en protocole MODBUS. Elle ne fonctionne pas dans les modes de flux continu ('Char' et 'Byte'), pour lesquels le début d'un message n'est pas défini.

La recherche retourne tous les messages trouvés qui correspondent au format. Tant qu'aucun message adéquat n'est trouvé, la fonction attendra, au maximum pour le temps spécifié en argument (en millisecondes).

Paramètres :

pattern une expression régulière limitée décrivant le format de message désiré, ou une chaîne vide si aucun filtrage des messages n'est désiré. Pour les protocoles binaires, le format est appliqué à la représentation hexadécimale du message.

maxWait le temps maximum d'attente pour obtenir un message, tant qu'aucun n'est trouvé dans le tampon de réception (en millisecondes).

Retourne :

un tableau de chaînes de caractères contenant les messages trouvés. Les messages binaires sont convertis automatiquement en représentation hexadécimale.

En cas d'erreur, déclenche une exception ou retourne un tableau vide.

serialport→**readStr()****serialport.readStr()****YSerialPort**

Lit le contenu du tampon de réception sous forme de string, à partir de la position courante dans le flux de donnée.

String **readStr**(int **nChars**)

Si le contenu à la position n'est plus disponible dans le tampon de réception, la fonction ne retournera que les données disponibles.

Paramètres :

nChars le nombre maximum de caractères à lire

Retourne :

une chaîne de caractère avec le contenu du tampon de réception.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→**read_seek()****serialport.read_seek()**

YSerialPort

Change le pointeur de position courante dans le flux de donnée à la valeur spécifiée.

```
int read_seek( int rxCountVal)
```

Cette fonction n'a pas d'effet sur le module, elle ne fait que changer la valeur stockée dans l'objet YSerialPort qui sera utilisée pour les prochaines operations de lecture.

Paramètres :

rxCountVal l'index de position absolue (valeur de rxCount) pour les opérations de lecture suivantes.

Retourne :

rien du tout.

serialport→registerValueCallback()**YSerialPort****serialport.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

serialport→**reset()**`serialport.reset()`

YSerialPort

Remet à zéro tous les compteurs et efface les tampons.

`int reset()`

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→**set_RTS()****YSerialPort****serialport**→**setRTS()****serialport.set_RTS()**

Change manuellement l'état de la ligne RTS.

```
int set_RTS( int val)
```

Cette fonction n'a pas d'effet lorsque le contrôle du flux par CTS/RTS est actif, car la ligne RTS est alors pilotée automatiquement.

Paramètres :

val 1 pour activer la ligne RTS, 0 pour la désactiver

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→**set_logicalName()****YSerialPort****serialport**→**setLogicalName()****serialport.set_logicalName()**

Modifie le nom logique du port série.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du port série.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→**set_protocol()****YSerialPort****serialport**→**setProtocol()****serialport.set_protocol()**

Modifie le type de protocole utilisé sur la communication série.

```
int set_protocol( String newval)
```

Les valeurs possibles sont "Line" pour des messages ASCII séparés par des retours de ligne, "Frame:[timeout]ms" pour des messages binaires séparés par une temporisation, "Modbus-ASCII" pour des messages MODBUS en mode ASCII, "Modbus-RTU" pour des messages MODBUS en mode RTU, "Char" pour un flux ASCII continu ou "Byte" pour un flux binaire continue.

Paramètres :

newval une chaîne de caractères représentant le type de protocole utilisé sur la communication série

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→**set_serialMode()****YSerialPort****serialport**→**setSerialMode()****serialport.set_serialMode()**

Modifie les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1".

```
int set_serialMode( String newval)
```

La chaîne contient le taux de transfert, le nombre de bits de données, la parité et le nombre de bits d'arrêt. Un suffixe supplémentaire optionnel peut être inclus pour activer une option de contrôle de flux: "CtsRts" pour le contrôle de flux matériel, "XOnXOff" pour le contrôle de flux logique et "Simplex" pour l'utilisation du signal RTS pour l'acquisition d'un bus partagé (tel qu'utilisé pour certains adaptateurs RS485 par exemple).

Paramètres :

newval une chaîne de caractères représentant les paramètres de communication du port, sous forme d'une chaîne de caractères du type "9600,8N1"

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→set_userdata()**YSerialPort****serialport→setUserData()****serialport.set_userdata()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

```
void set_userdata( Object data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

serialport→**writeArray()****serialport.writeArray()****YSerialPort**

Envoie une séquence d'octets (fournie sous forme d'une liste) sur le port série.

```
int writeArray( ArrayList<Integer> byteList)
```

Paramètres :

byteList la liste d'octets à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→**writeBin()**`serialport.writeBin()`**YSerialPort**

Envoie un objet binaire tel quel sur le port série.

int **writeBin**()

Paramètres :

buff l'objet binaire à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→**writeHex()****serialport.writeHex()**

YSerialPort

Envoie une séquence d'octets (fournie sous forme de chaîne hexadécimale) sur le port série.

```
int writeHex( String hexString)
```

Paramètres :

hexString la chaîne hexadécimale à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→**writeLine()**`serialport.writeLine()`**YSerialPort**

Envoie une chaîne de caractères sur le port série, suivie d'un saut de ligne (CR LF).

```
int writeLine( String text)
```

Paramètres :

text la chaîne de caractères à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→**writeMODBUS()****YSerialPort****serialport.writeMODBUS()**

Envoie une commande MODBUS (fournie sous forme de chaîne hexadécimale) sur le port série.

```
int writeMODBUS( String hexString)
```

Le message doit commencer par l'adresse de destination. Le CRC (ou LRC) MODBUS est ajouté automatiquement par la fonction. Cette fonction n'attend pas de réponse.

Paramètres :

hexString le message à envoyer, en hexadécimal, sans le CRC/LRC

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

serialport→writeStr()`serialport.writeStr()`**YSerialPort**

Envoie une chaîne de caractères telle quelle sur le port série.

```
int writeStr( String text)
```

Paramètres :

text la chaîne de caractères à envoyer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.41. Interface de la fonction Servo

La librairie de programmation Yoctopuce permet non seulement de déplacer le servo vers une position donnée, mais aussi de spécifier l'intervalle de temps dans lequel le mouvement doit être fait, de sorte à pouvoir synchroniser un mouvement sur plusieurs servos.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_servo.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib');</code> <code>var YServo = yoctolib.YServo;</code>
php	<code>require_once('yocto_servo.php');</code>
c++	<code>#include "yocto_servo.h"</code>
m	<code>#import "yocto_servo.h"</code>
pas	<code>uses yocto_servo;</code>
vb	<code>yocto_servo.vb</code>
cs	<code>yocto_servo.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YServo;</code>
py	<code>from yocto_servo import *</code>

Fonction globales

yFindServo(func)

Permet de retrouver un servo d'après un identifiant donné.

yFirstServo()

Commence l'énumération des servo accessibles par la librairie.

Méthodes des objets YServo

servo→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du servo au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

servo→get_advertisedValue()

Retourne la valeur courante du servo (pas plus de 6 caractères).

servo→get_enabled()

Retourne l'état de fonctionnement du \$FUNCTION\$.

servo→get_enabledAtPowerOn()

Retourne l'état du générateur de signal de commande du servo au démarrage du module.

servo→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du servo.

servo→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du servo.

servo→get_friendlyName()

Retourne un identifiant global du servo au format `NOM_MODULE . NOM_FONCTION`.

servo→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

servo→get_functionId()

Retourne l'identifiant matériel du servo, sans référence au module.

servo→get_hardwareId()

Retourne l'identifiant matériel unique du servo au format `SERIAL . FUNCTIONID`.

servo→get_logicalName()

Retourne le nom logique du servo.

servo→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

servo→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

servo→get_neutral()

Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.

servo→get_position()

Retourne la position courante du servo.

servo→get_positionAtPowerOn()

Retourne la position du servo au démarrage du module.

servo→get_range()

Retourne la plage d'utilisation du servo.

servo→get_userData()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

servo→isOnline()

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

servo→isOnline_async(callback, context)

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

servo→load(msValidity)

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

servo→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

servo→move(target, ms_duration)

Déclenche un mouvement à vitesse constante vers une position donnée.

servo→nextServo()

Continue l'énumération des servo commencée à l'aide de `yFirstServo()`.

servo→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

servo→set_enabled(newval)

Démarre ou arrête le \$FUNCTION\$.

servo→set_enabledAtPowerOn(newval)

Configure l'état du générateur de signal de commande du servo au démarrage du module.

servo→set_logicalName(newval)

Modifie le nom logique du servo.

servo→set_neutral(newval)

Modifie la durée de l'impulsion correspondant à la position neutre du servo.

servo→set_position(newval)

Modifie immédiatement la consigne de position du servo.

servo→set_positionAtPowerOn(newval)

Configure la position du servo au démarrage du module.

servo→set_range(newval)

Modifie la plage d'utilisation du servo, en pourcents.

servo→set_userData(data)

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

servo→wait_async(callback, context)

3. Reference

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YServo.FindServo()**YServo****yFindServo()**`YServo.FindServo()`

Permet de retrouver un servo d'après un identifiant donné.

`YServo FindServo(String func)`

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le servo soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YServo.isOnline()` pour tester si le servo est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le servo sans ambiguïté

Retourne :

un objet de classe `YServo` qui permet ensuite de contrôler le servo.

YServo.FirstServo()

YServo

yFirstServo()**YServo.FirstServo()**

Commence l'énumération des servo accessibles par la librairie.

YServo FirstServo()

Utiliser la fonction `YServo.nextServo()` pour itérer sur les autres servo.

Retourne :

un pointeur sur un objet `YServo`, correspondant au premier servo accessible en ligne, ou `null` si il n'y a pas de servo disponibles.

servo→describe()`servo.describe()`**YServo**

Retourne un court texte décrivant de manière non-ambigüe l'instance du servo au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String `describe()`

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le servo (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

servo→**get_advertisedValue()**

YServo

servo→**advertisedValue()**

servo.get_advertisedValue()

Retourne la valeur courante du servo (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du servo (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

servo→get_enabled()**YServo****servo→enabled()**`servo.get_enabled()`

Retourne l'état de fonctionnement du \$FUNCTION\$.

int **get_enabled**()

Retourne :

soit Y_ENABLED_FALSE, soit Y_ENABLED_TRUE, selon l'état de fonctionnement du \$FUNCTION\$

En cas d'erreur, déclenche une exception ou retourne Y_ENABLED_INVALID.

servo→get_enabledAtPowerOn()

YServo

servo→enabledAtPowerOn()

servo.get_enabledAtPowerOn()

Retourne l'état du générateur de signal de commande du servo au démarrage du module.

int get_enabledAtPowerOn()

Retourne :

soit Y_ENABLEDATPOWERON_FALSE, soit Y_ENABLEDATPOWERON_TRUE, selon l'état du générateur de signal de commande du servo au démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_ENABLEDATPOWERON_INVALID.

servo→get_errorMessage()**YServo****servo→errorMessage()**`servo.errorMessage()`

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du servo.

`String` **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du servo.

servo→**get_errorType()**

YServo

servo→**errorType()****servo.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du servo.

int **get_errorType()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du servo.

servo→get_friendlyName()**YServo****servo→friendlyName()**`servo.get_friendlyName()`

Retourne un identifiant global du servo au format `NOM_MODULE.NOM_FONCTION`.

String `get_friendlyName()`

Le chaîne retournée utilise soit les noms logiques du module et du servo si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du servo (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le servo en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

servo→**get_functionDescriptor()**

YServo

servo→**functionDescriptor()**

servo.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

servo→**get_functionId()****YServo****servo**→**functionId()****servo.get_functionId()**

Retourne l'identifiant matériel du servo, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le servo (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

servo→**get_hardwareId()**

YServo

servo→**hardwareId()****servo.get_hardwareId()**

Retourne l'identifiant matériel unique du servo au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du servo (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le servo (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

servo→get_logicalName()**YServo****servo→logicalName()**`servo.get_logicalName()`

Retourne le nom logique du servo.

`String` **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du servo.

En cas d'erreur, déclenche une exception ou retourne `Y_LOGICALNAME_INVALID`.

servo→**get_module()**

YServo

servo→**module()**`servo.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

servo→**get_neutral()****YServo****servo**→**neutral()****servo.get_neutral()**

Retourne la durée en microsecondes de l'impulsion correspondant au neutre du servo.

```
int get_neutral( )
```

Retourne :

un entier représentant la durée en microsecondes de l'impulsion correspondant au neutre du servo

En cas d'erreur, déclenche une exception ou retourne Y_NEUTRAL_INVALID.

servo→get_position()

YServo

servo→position()`servo.get_position()`

Retourne la position courante du servo.

`int get_position()`

Retourne :

un entier représentant la position courante du servo

En cas d'erreur, déclenche une exception ou retourne `Y_POSITION_INVALID`.

servo→get_positionAtPowerOn()**YServo****servo→positionAtPowerOn()****servo.get_positionAtPowerOn()**

Retourne la position du servo au démarrage du module.

int **get_positionAtPowerOn()**

Retourne :

un entier représentant la position du servo au démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_POSITIONATPOWERON_INVALID.

servo→**get_range()**

YServo

servo→**range()**`servo.get_range()`

Retourne la plage d'utilisation du servo.

`int get_range()`

Retourne :

un entier représentant la plage d'utilisation du servo

En cas d'erreur, déclenche une exception ou retourne `Y_RANGE_INVALID`.

servo→get_userdata()**YServo****servo→userData()**`servo.getUserData()`

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

Object `get_userdata()`

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

servo→**isOnline()****servo.isOnline()**

YServo

Vérifie si le module hébergeant le servo est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache du servo sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le servo est joignable, `false` sinon

servo→load()`servo.load()`**YServo**

Met en cache les valeurs courantes du servo, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**move()****servo.move()****YServo**

Déclenche un mouvement à vitesse constante vers une position donnée.

```
int move( int target, int ms_duration)
```

Paramètres :

target nouvelle position à la fin du mouvement
ms_duration durée totale du mouvement, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**nextServo()****servo.nextServo()****YServo**

Continue l'énumération des servo commencée à l'aide de `yFirstServo()`.

`YServo` **nextServo()**

Retourne :

un pointeur sur un objet `YServo` accessible en ligne, ou `null` lorsque l'énumération est terminée.

servo→registerValueCallback()**YServo****servo.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

servo→**set_enabled()****YServo****servo**→**setEnabled()****servo.set_enabled()**

Démarre ou arrête le \$FUNCTION\$.

int **set_enabled**(int **newval**)

Paramètres :

newval soit Y_ENABLED_FALSE, soit Y_ENABLED_TRUE

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**set_enabledAtPowerOn()****YServo****servo**→**setEnabledAtPowerOn()****servo.set_enabledAtPowerOn()**

Configure l'état du générateur de signal de commande du servo au démarrage du module.

```
int set_enabledAtPowerOn( int newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval soit `Y_ENABLEDATPOWERON_FALSE`, soit `Y_ENABLEDATPOWERON_TRUE`

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**set_logicalName()**
servo→**setLogicalName()**
servo.set_logicalName()

YServo

Modifie le nom logique du servo.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du servo.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**set_neutral()**

YServo

servo→**setNeutral()****servo.set_neutral()**

Modifie la durée de l'impulsion correspondant à la position neutre du servo.

```
int set_neutral( int newval)
```

La durée est spécifiée en microsecondes, et la valeur standard est 1500 [us]. Ce réglage permet de décaler la plage d'utilisation du servo. Attention, l'utilisation d'une plage supérieure aux caractéristiques du servo risque fortement d'endommager le servo.

Paramètres :

newval un entier représentant la durée de l'impulsion correspondant à la position neutre du servo

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**set_position()****YServo****servo**→**setPosition()****servo.set_position()**

Modifie immédiatement la consigne de position du servo.

```
int set_position( int newval)
```

Paramètres :

newval un entier représentant immédiatement la consigne de position du servo

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→set_positionAtPowerOn()

YServo

servo→setPositionAtPowerOn()

servo.set_positionAtPowerOn()

Configure la position du servo au démarrage du module.

```
int set_positionAtPowerOn( int newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→set_range()**YServo****servo→setRange()**`servo.set_range()`

Modifie la plage d'utilisation du servo, en pourcents.

```
int set_range( int newval)
```

La valeur 100% correspond à un signal de commande standard, variant de 1 [ms] à 2 [ms]. Pour les servos supportent une plage double, de 0.5 [ms] à 2.5 [ms], vous pouvez utiliser une valeur allant jusqu'à 200%. Attention, l'utilisation d'une plage supérieure aux caractéristiques du servo risque fortement d'endommager le servo.

Paramètres :

newval un entier représentant la plage d'utilisation du servo, en pourcents

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

servo→**set_userdata()**

YServo

servo→**setUserData()****servo.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

3.42. Interface de la fonction Temperature

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_temperature.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YTemperature = yoctolib.YTemperature;
php	require_once('yocto_temperature.php');
c++	#include "yocto_temperature.h"
m	#import "yocto_temperature.h"
pas	uses yocto_temperature;
vb	yocto_temperature.vb
cs	yocto_temperature.cs
java	import com.yoctopuce.YoctoAPI.YTemperature;
py	from yocto_temperature import *

Fonction globales

yFindTemperature(func)

Permet de retrouver un capteur de température d'après un identifiant donné.

yFirstTemperature()

Commence l'énumération des capteurs de température accessibles par la librairie.

Méthodes des objets YTemperature

temperature→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

temperature→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de température au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

temperature→get_advertisedValue()

Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

temperature→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), 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_DESCRIPTOR` 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.

YTemperature.FindTemperature()**YTemperature****yFindTemperature()****YTemperature.FindTemperature()**

Permet de retrouver un capteur de température d'après un identifiant donné.

YTemperature **FindTemperature**(String **func**)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de température soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YTemperature.isOnline()` pour tester si le capteur de température est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de température sans ambiguïté

Retourne :

un objet de classe `YTemperature` qui permet ensuite de contrôler le capteur de température.

YTemperature.FirstTemperature()**YTemperature****yFirstTemperature()****YTemperature.FirstTemperature()**

Commence l'énumération des capteurs de température accessibles par la librairie.

YTemperature **FirstTemperature()**

Utiliser la fonction `YTemperature.nextTemperature()` pour itérer sur les autres capteurs de température.

Retourne :

un pointeur sur un objet `YTemperature`, correspondant au premier capteur de température accessible en ligne, ou `null` si il n'y a pas de capteurs de température disponibles.

temperature→calibrateFromPoints()**YTemperature****temperature.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                        ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**describe()****temperature.describe()****YTemperature**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de température au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

String **describe()**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

Retourne :

une chaîne de caractères décrivant le capteur de température (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

temperature→**get_advertisedValue()**

YTemperature

temperature→**advertisedValue()**

temperature.get_advertisedValue()

Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de température (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

temperature→**get_currentRawValue()****YTemperature****temperature**→**currentRawValue()****temperature.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés Celsius, sous forme de nombre à virgule.

double **get_currentRawValue()**

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés Celsius, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

temperature→get_currentValue()

YTemperature

temperature→currentValue()

temperature.get_currentValue()

Retourne la valeur actuelle de la température, en degrés Celsius, sous forme de nombre à virgule.

`double get_currentValue()`

Retourne :

une valeur numérique représentant la valeur actuelle de la température, en degrés Celsius, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

temperature→**get_errorMessage()****YTemperature****temperature**→**errorMessage()****temperature.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de température.

temperature→get_errorType()

YTemperature

temperature→errorType()

temperature.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de température.

temperature→**get_friendlyName()****YTemperature****temperature**→**friendlyName()****temperature.get_friendlyName()**

Retourne un identifiant global du capteur de température au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du capteur de température si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de température (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de température en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

temperature → get_functionDescriptor()	YTemperature
temperature → functionDescriptor()	
temperature.get_functionDescriptor()	

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

temperature→**get_functionId()****YTemperature****temperature**→**functionId()****temperature.get_functionId()**

Retourne l'identifiant matériel du capteur de température, sans référence au module.

String **get_functionId()** ()

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de température (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

temperature→**get_hardwareId()**

YTemperature

temperature→**hardwareId()**

temperature.get_hardwareId()

Retourne l'identifiant matériel unique du capteur de température au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de température (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur de température (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

temperature→**get_highestValue()****YTemperature****temperature**→**highestValue()****temperature.get_highestValue()**

Retourne la valeur maximale observée pour la température depuis le démarrage du module.

double **get_highestValue()**

Retourne :

une valeur numérique représentant la valeur maximale observée pour la température depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

temperature→get_logFrequency()

YTemperature

temperature→logFrequency()

temperature.get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

String get_logFrequency()

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

temperature→**get_logicalName()****YTemperature****temperature**→**logicalName()****temperature.get_logicalName()**

Retourne le nom logique du capteur de température.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du capteur de température.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

temperature→**get_lowestValue()**

YTemperature

temperature→**lowestValue()**

temperature.get_lowestValue()

Retourne la valeur minimale observée pour la température depuis le démarrage du module.

`double get_lowestValue()`

Retourne :

une valeur numérique représentant la valeur minimale observée pour la température depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_LOWESTVALUE_INVALID.

temperature→**get_module()****YTemperature****temperature**→**module()****temperature.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

temperature→**get_recordedData()**

YTemperature

temperature→**recordedData()**

temperature.get_recordedData()

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

YDataSet **get_recordedData**(long **startTime**, long **endTime**)

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

temperature→**get_reportFrequency()****YTemperature****temperature**→**reportFrequency()****temperature.get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String **get_reportFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

temperature→**get_resolution()**

YTemperature

temperature→**resolution()**

temperature.get_resolution()

Retourne la résolution des valeurs mesurées.

double **get_resolution()** ()

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_RESOLUTION_INVALID.

temperature→**get_sensorType()****YTemperature****temperature**→**sensorType()****temperature.get_sensorType()**

Retourne le type de capteur de température utilisé par le module

int **get_sensorType()**

Retourne :

une valeur parmi Y_SENSORTYPE_DIGITAL, Y_SENSORTYPE_TYPE_K, Y_SENSORTYPE_TYPE_E, Y_SENSORTYPE_TYPE_J, Y_SENSORTYPE_TYPE_N, Y_SENSORTYPE_TYPE_R, Y_SENSORTYPE_TYPE_S, Y_SENSORTYPE_TYPE_T, Y_SENSORTYPE_PT100_4WIRES, Y_SENSORTYPE_PT100_3WIRES et Y_SENSORTYPE_PT100_2WIRES représentant le type de capteur de température utilisé par le module

En cas d'erreur, déclenche une exception ou retourne Y_SENSORTYPE_INVALID.

temperature→**get_unit()**

YTemperature

temperature→**unit()**`temperature.get_unit()`

Retourne l'unité dans laquelle la température est exprimée.

String **get_unit()** ()

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la température est exprimée

En cas d'erreur, déclenche une exception ou retourne Y_UNIT_INVALID.

temperature→**get_userdata()****YTemperature****temperature**→**userData()****temperature.userData()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

temperature→**isOnline()****temperature.isOnline()**

YTemperature

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du capteur de température sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur de température est joignable, `false` sinon

temperature→**load()****temperature.load()****YTemperature**

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→loadCalibrationPoints()**YTemperature****temperature.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**nextTemperature()****YTemperature****temperature.nextTemperature()**

Continue l'énumération des capteurs de température commencée à l'aide de `yFirstTemperature()`.

YTemperature **nextTemperature()**

Retourne :

un pointeur sur un objet `YTemperature` accessible en ligne, ou `null` lorsque l'énumération est terminée.

temperature→registerTimedReportCallback()**YTemperature****temperature.registerTimedReportCallback()**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

temperature→**registerValueCallback()****YTemperature****temperature.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

temperature→**set_highestValue()**

YTemperature

temperature→**setHighestValue()**

temperature.set_highestValue()

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**set_logFrequency()****YTemperature****temperature**→**setLogFrequency()****temperature.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**set_logicalName()**

YTemperature

temperature→**setLogicalName()**

temperature.set_logicalName()

Modifie le nom logique du capteur de température.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de température.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**set_lowestValue()****YTemperature****temperature**→**setLowestValue()****temperature.set_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_reportFrequency()

YTemperature

temperature→setReportFrequency()

temperature.set_reportFrequency()

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**set_resolution()**
temperature→**setResolution()**
temperature.set_resolution()

YTemperature

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_sensorType()**YTemperature****temperature→setSensorType()****temperature.set_sensorType()**

Change le type de senseur utilisé par le module.

```
int set_sensorType( int newval)
```

Cette fonction sert à spécifier le type de thermocouple (K,E, etc..) raccordé au module. Cette fonction n'aura pas d'effet si le module utilise un capteur digital. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur parmi Y_SENSORTYPE_DIGITAL, Y_SENSORTYPE_TYPE_K, Y_SENSORTYPE_TYPE_E, Y_SENSORTYPE_TYPE_J, Y_SENSORTYPE_TYPE_N, Y_SENSORTYPE_TYPE_R, Y_SENSORTYPE_TYPE_S, Y_SENSORTYPE_TYPE_T, Y_SENSORTYPE_PT100_4WIRES, Y_SENSORTYPE_PT100_3WIRES et Y_SENSORTYPE_PT100_2WIRES

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→**set_userdata()****YTemperature****temperature**→**setUserData()****temperature.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

3.43. Interface de la fonction Tilt

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_tilt.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YTilt = yoctolib.YTilt;
php	require_once('yocto_tilt.php');
c++	#include "yocto_tilt.h"
m	#import "yocto_tilt.h"
pas	uses yocto_tilt;
vb	yocto_tilt.vb
cs	yocto_tilt.cs
java	import com.yoctopuce.YoctoAPI.YTilt;
py	from yocto_tilt import *

Fonction globales

yFindTilt(func)

Permet de retrouver un inclinomètre d'après un identifiant donné.

yFirstTilt()

Commence l'énumération des inclinomètres accessibles par la librairie.

Méthodes des objets YTilt

tilt→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

tilt→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'inclinomètre au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

tilt→get_advertisedValue()

Retourne la valeur courante de l'inclinomètre (pas plus de 6 caractères).

tilt→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), 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)`

3. Reference

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

tilt→set_logicalName(newval)

Modifie le nom logique de l'inclinomètre.

tilt→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

tilt→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

tilt→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

tilt→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

tilt→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YTilt.FindTilt()**YTilt****yFindTilt()YTilt.FindTilt()**

Permet de retrouver un inclinomètre d'après un identifiant donné.

YTilt FindTilt(String func)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que l'inclinomètre soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YTilt.isOnline()` pour tester si l'inclinomètre est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence l'inclinomètre sans ambiguïté

Retourne :

un objet de classe `YTilt` qui permet ensuite de contrôler l'inclinomètre.

YTilt.FirstTilt()

YTilt

yFirstTilt()`YTilt.FirstTilt()`

Commence l'énumération des inclinomètres accessibles par la librairie.

YTilt FirstTilt()

Utiliser la fonction `YTilt.nextTilt()` pour itérer sur les autres inclinomètres.

Retourne :

un pointeur sur un objet `YTilt`, correspondant au premier inclinomètre accessible en ligne, ou `null` si il n'y a pas de inclinomètres disponibles.

tilt→calibrateFromPoints()**YTilt****tilt.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                        ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→describe()`tilt.describe()`**YTilt**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'inclinomètre au format `TYPE (NAME) = SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant l'inclinomètre (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

tilt→**get_advertisedValue()****YTilt****tilt**→**advertisedValue()****tilt.get_advertisedValue()**

Retourne la valeur courante de l'inclinomètre (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante de l'inclinomètre (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `Y_ADVERTISEDVALUE_INVALID`.

tilt→get_currentRawValue()

YTilt

tilt→currentRawValue()

tilt.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés, sous forme de nombre à virgule.

double get_currentRawValue()

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

tilt→**get_currentValue()****YTilt****tilt**→**currentValue()****tilt.get_currentValue()**

Retourne la valeur actuelle de l'inclinaison, en degrés, sous forme de nombre à virgule.

double **get_currentValue()**

Retourne :

une valeur numérique représentant la valeur actuelle de l'inclinaison, en degrés, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

tilt→**get_errorMessage()**

YTilt

tilt→**errorMessage()****tilt.errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'inclinomètre.

tilt→get_errorType()**YTilt****tilt→errorType()**`tilt.get_errorType()`

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'inclinomètre.

`int get_errorType()`

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'inclinomètre.

tilt→**get_friendlyName()****YTilt****tilt**→**friendlyName()****tilt.get_friendlyName()**

Retourne un identifiant global de l'inclinomètre au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()** ()

Le chaîne retournée utilise soit les noms logiques du module et de l'inclinomètre si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'inclinomètre (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'inclinomètre en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

tilt→**get_functionDescriptor()****YTilt****tilt**→**functionDescriptor()****tilt.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

tilt→**get_functionId()**

YTilt

tilt→**functionId()**`tilt.get_functionId()`

Retourne l'identifiant matériel de l'inclinomètre, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant l'inclinomètre (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

tilt→**get_hardwareId()****YTilt****tilt**→**hardwareId()****tilt.get_hardwareId()**

Retourne l'identifiant matériel unique de l'inclinomètre au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'inclinomètre (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant l'inclinomètre (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

tilt→**get_highestValue()**

YTilt

tilt→**highestValue()**`tilt.get_highestValue()`

Retourne la valeur maximale observée pour l'inclinaison depuis le démarrage du module.

double **get_highestValue()**

Retourne :

une valeur numérique représentant la valeur maximale observée pour l'inclinaison depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_HIGHESTVALUE_INVALID`.

tilt→**get_logFrequency()****YTilt****tilt**→**logFrequency()****tilt.get_logFrequency()**

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

String **get_logFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

tilt→**get_logicalName()**

YTilt

tilt→**logicalName()**`tilt.get_logicalName()`

Retourne le nom logique de l'inclinomètre.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique de l'inclinomètre.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

tilt→**get_lowestValue()****YTilt****tilt**→**lowestValue()****tilt.get_lowestValue()**

Retourne la valeur minimale observée pour l'inclinaison depuis le démarrage du module.

double **get_lowestValue()**

Retourne :

une valeur numérique représentant la valeur minimale observée pour l'inclinaison depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

tilt→**get_module()**

YTilt

tilt→**module()**`tilt.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

tilt→**get_recordedData()****YTilt****tilt**→**recordedData()****tilt.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

YDataSet **get_recordedData**(long **startTime**, long **endTime**)

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intercalé de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

tilt→get_reportFrequency()

YTilt

tilt→reportFrequency()

tilt.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String get_reportFrequency()

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

tilt→**get_resolution()****YTilt****tilt**→**resolution()****tilt.get_resolution()**

Retourne la résolution des valeurs mesurées.

double **get_resolution()** ()

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_RESOLUTION_INVALID.

tilt→**get_unit()**

YTilt

tilt→**unit()**`tilt.get_unit()`

Retourne l'unité dans laquelle l'inclinaison est exprimée.

String **get_unit()** ()

Retourne :

une chaîne de caractères représentant l'unité dans laquelle l'inclinaison est exprimée

En cas d'erreur, déclenche une exception ou retourne Y_UNIT_INVALID.

tilt→**get_userData()****YTilt****tilt**→**userData()****tilt.get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

Object **get_userData()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

tilt→**isOnline()**`tilt.isOnline()`

YTilt

Vérifie si le module hébergeant l'inclinomètre est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache de l'inclinomètre sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'inclinomètre est joignable, `false` sinon

tilt→load()`tilt.load()`**YTilt**

Met en cache les valeurs courantes de l'inclinomètre, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→loadCalibrationPoints()**YTilt****tilt.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→**nextTilt()****tilt.nextTilt()****YTilt**

Continue l'énumération des inclinomètres commencée à l'aide de `yFirstTilt()`.

YTilt **nextTilt()**

Retourne :

un pointeur sur un objet `YTilt` accessible en ligne, ou `null` lorsque l'énumération est terminée.

tilt→registerTimedReportCallback()**YTilt****tilt.registerTimedReportCallback()**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

tilt→registerValueCallback()**YTilt****tilt.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

tilt→**set_highestValue()**

YTilt

tilt→**setHighestValue()**`tilt.set_highestValue()`

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→set_logFrequency()**YTilt****tilt→setLogFrequency()**`tilt.set_logFrequency()`

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→**set_logicalName()****YTilt****tilt**→**setLogicalName()****tilt.set_logicalName()**

Modifie le nom logique de l'inclinomètre.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'inclinomètre.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→**set_lowestValue()****YTilt****tilt**→**setLowestValue()****tilt.set_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→**set_reportFrequency()****tilt**→**setReportFrequency()****tilt.set_reportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→**set_resolution()****YTilt****tilt**→**setResolution()****tilt.set_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

tilt→**set_userdata()**

YTilt

tilt→**setUserData()****tilt.set_userdata()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

3.44. Interface de la fonction Voc

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_voc.js'></script></code>
nodejs	<code>var yoctolib = require('yoctolib'); var YVoc = yoctolib.YVoc;</code>
php	<code>require_once('yocto_voc.php');</code>
c++	<code>#include "yocto_voc.h"</code>
m	<code>#import "yocto_voc.h"</code>
pas	<code>uses yocto_voc;</code>
vb	<code>yocto_voc.vb</code>
cs	<code>yocto_voc.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YVoc;</code>
py	<code>from yocto_voc import *</code>

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.

YVoc.FindVoc()**YVoc****yFindVoc()****YVoc.FindVoc()**

Permet de retrouver un capteur de Composés Organiques Volatils d'après un identifiant donné.

YVoc FindVoc(String func)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de Composés Organiques Volatils soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVoc.isOnline()` pour tester si le capteur de Composés Organiques Volatils est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de Composés Organiques Volatils sans ambiguïté

Retourne :

un objet de classe `YVoc` qui permet ensuite de contrôler le capteur de Composés Organiques Volatils.

YVoc.FirstVoc()**YVoc****yFirstVoc()****YVoc.FirstVoc()**

Commence l'énumération des capteurs de Composés Organiques Volatils accessibles par la librairie.

YVoc FirstVoc()

Utiliser la fonction `YVoc.nextVoc()` pour itérer sur les autres capteurs de Composés Organiques Volatils.

Retourne :

un pointeur sur un objet `YVoc`, correspondant au premier capteur de Composés Organiques Volatils accessible en ligne, ou `null` si il n'y a pas de capteurs de Composés Organiques Volatils disponibles.

voc→calibrateFromPoints()**YVoc****voc.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                        ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→describe()**voc.describe()****YVoc**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de Composés Organiques Volatils au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le capteur de Composés Organiques Volatils (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

voc→**get_advertisedValue()**

YVoc

voc→**advertisedValue()**

voc.get_advertisedValue()

Retourne la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de Composés Organiques Volatils (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

voc→**get_currentRawValue()****YVoc****voc**→**currentRawValue()****voc.get_currentRawValue()**

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en ppm (vol), sous forme de nombre à virgule.

double **get_currentRawValue()****Retourne :**

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en ppm (vol), sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `Y_CURRENTRAWVALUE_INVALID`.

voc→**get_currentValue()**

YVoc

voc→**currentValue()****voc.get_currentValue()**

Retourne la valeur actuelle du taux de VOC estimé, en ppm (vol), sous forme de nombre à virgule.

double **get_currentValue()**

Retourne :

une valeur numérique représentant la valeur actuelle du taux de VOC estimé, en ppm (vol), sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

voc→**get_errorMessage()****YVoc****voc**→**errorMessage()****voc.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de Composés Organiques Volatils.

voc→**get_errorType()**

YVoc

voc→**errorType()****voc.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de Composés Organiques Volatils.

int **get_errorType()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de Composés Organiques Volatils.

voc→**get_friendlyName()****YVoc****voc**→**friendlyName()****voc.get_friendlyName()**

Retourne un identifiant global du capteur de Composés Organiques Volatils au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et du capteur de Composés Organiques Volatils si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de Composés Organiques Volatils (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de Composés Organiques Volatils en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

voc→**get_functionDescriptor()**

YVoc

voc→**functionDescriptor()**

voc.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

voc→**get_functionId()****YVoc****voc**→**functionId()****voc.get_functionId()**

Retourne l'identifiant matériel du capteur de Composés Organiques Volatils, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de Composés Organiques Volatils (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

voc→**get_hardwareId()**

YVoc

voc→**hardwareId()****voc.get_hardwareId()**

Retourne l'identifiant matériel unique du capteur de Composés Organiques Volatils au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de Composés Organiques Volatils (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur de Composés Organiques Volatils (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

voc→**get_highestValue()****YVoc****voc**→**highestValue()****voc.get_highestValue()**

Retourne la valeur maximale observée pour le taux de VOC estimé depuis le démarrage du module.

double **get_highestValue()**

Retourne :

une valeur numérique représentant la valeur maximale observée pour le taux de VOC estimé depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

voc→**get_logFrequency()**

YVoc

voc→**logFrequency()****voc.get_logFrequency()**

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.

String **get_logFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

voc→**get_logicalName()****YVoc****voc**→**logicalName()****voc.get_logicalName()**

Retourne le nom logique du capteur de Composés Organiques Volatils.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du capteur de Composés Organiques Volatils.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

voc→**get_lowestValue()**

YVoc

voc→**lowestValue()**`voc.get_lowestValue()`

Retourne la valeur minimale observée pour le taux de VOC estimé depuis le démarrage du module.

double **get_lowestValue()** ()

Retourne :

une valeur numérique représentant la valeur minimale observée pour le taux de VOC estimé depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_LOWESTVALUE_INVALID.

voc→**get_module()****YVoc****voc**→**module()**`voc.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

voc→**get_recordedData()****YVoc****voc**→**recordedData()****voc.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

YDataSet **get_recordedData(long startTime, long endTime)**

Veillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

voc→get_reportFrequency()**YVoc****voc→reportFrequency()****voc.get_reportFrequency()**

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String **get_reportFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

voc→**get_resolution()**

YVoc

voc→**resolution()**`voc.get_resolution()`

Retourne la résolution des valeurs mesurées.

`double` **get_resolution()**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `Y_RESOLUTION_INVALID`.

voc→**get_unit()****YVoc****voc**→**unit()****voc.get_unit()**

Retourne l'unité dans laquelle le taux de VOC estimé est exprimée.

String **get_unit()**

Retourne :

une chaîne de caractères représentant l'unité dans laquelle le taux de VOC estimé est exprimée

En cas d'erreur, déclenche une exception ou retourne Y_UNIT_INVALID.

voc→**get_userData()**

YVoc

voc→**userData()****voc.userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

voc→isOnline()`voc.isOnline()`**YVoc**

Vérifie si le module hébergeant le capteur de Composés Organiques Volatils est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du capteur de Composés Organiques Volatils sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur de Composés Organiques Volatils est joignable, `false` sinon

voc→**load()****voc.load()****YVoc**

Met en cache les valeurs courantes du capteur de Composés Organiques Volatils, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→loadCalibrationPoints()**YVoc****voc.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→**nextVoc()****voc.nextVoc()**

YVoc

Continue l'énumération des capteurs de Composés Organiques Volatils commencée à l'aide de `yFirstVoc()`.

YVoc nextVoc()

Retourne :

un pointeur sur un objet **YVoc** accessible en ligne, ou `null` lorsque l'énumération est terminée.

voc→registerTimedReportCallback()**YVoc****voc.registerTimedReportCallback()**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

voc→**registerValueCallback()****YVoc****voc.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

voc→**set_highestValue()****YVoc****voc**→**setHighestValue()****voc.set_highestValue()**

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→**set_logFrequency()****YVoc****voc**→**setLogFrequency()****voc.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→**set_logicalName()****YVoc****voc**→**setLogicalName()****voc.set_logicalName()**

Modifie le nom logique du capteur de Composés Organiques Volatils.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de Composés Organiques Volatils.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→**set_lowestValue()**

YVoc

voc→**setLowestValue()**`voc.set_lowestValue()`

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→**set_reportFrequency()**
voc→**setReportFrequency()**
voc.set_reportFrequency()

YVoc

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→**set_resolution()****YVoc****voc**→**setResolution()****voc.set_resolution()**

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voc→**set_userdata()****YVoc****voc**→**setUserData()****voc.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

```
void set_userdata( Object data )
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

3.45. Interface de la fonction Voltage

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrêmes atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_voltage.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YVoltage = yoctolib.YVoltage;
php	require_once('yocto_voltage.php');
c++	#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)

3. Reference

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

voltage→**set_logicalName**(**newval**)

Modifie le nom logique du capteur de tension.

voltage→**set_lowestValue**(**newval**)

Modifie la mémoire de valeur minimale observée.

voltage→**set_reportFrequency**(**newval**)

Modifie la fréquence de notification périodique des valeurs mesurées.

voltage→**set_resolution**(**newval**)

Modifie la résolution des valeurs physique mesurées.

voltage→**set_userData**(**data**)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

voltage→**wait_async**(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YVoltage.FindVoltage()**YVoltage****yFindVoltage()**`YVoltage.FindVoltage()`

Permet de retrouver un capteur de tension d'après un identifiant donné.

`YVoltage` **FindVoltage**(String **func**)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVoltage.isOnline()` pour tester si le capteur de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le capteur de tension sans ambiguïté

Retourne :

un objet de classe `YVoltage` qui permet ensuite de contrôler le capteur de tension.

YVoltage.FirstVoltage()

YVoltage

yFirstVoltage()`YVoltage.FirstVoltage()`

Commence l'énumération des capteurs de tension accessibles par la librairie.

`YVoltage` **FirstVoltage()**

Utiliser la fonction `YVoltage.nextVoltage()` pour itérer sur les autres capteurs de tension.

Retourne :

un pointeur sur un objet `YVoltage`, correspondant au premier capteur de tension accessible en ligne, ou `null` si il n'y a pas de capteurs de tension disponibles.

voltage→calibrateFromPoints()**YVoltage****voltage.calibrateFromPoints()**

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
int calibrateFromPoints( ArrayList<Double> rawValues,  
                        ArrayList<Double> refValues)
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→describe()`voltage.describe()`**YVoltage**

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de tension au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le capteur de tension (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

voltage→**get_advertisedValue()****YVoltage****voltage**→**advertisedValue()****voltage.get_advertisedValue()**

Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de tension (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

voltage→get_currentRawValue()

YVoltage

voltage→currentRawValue()

voltage.get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule.

double **get_currentRawValue()**

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en Volt, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTRAWVALUE_INVALID.

voltage→**get_currentValue()**
voltage→**currentValue()**
voltage.get_currentValue()

YVoltage

Retourne la valeur actuelle de la tension, en Volt, sous forme de nombre à virgule.

double **get_currentValue()**

Retourne :

une valeur numérique représentant la valeur actuelle de la tension, en Volt, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne Y_CURRENTVALUE_INVALID.

voltage→get_errorMessage()

YVoltage

voltage→errorMessage()

voltage.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

String get_errorMessage()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de tension.

voltage→**get_errorType()****YVoltage****voltage**→**errorType()****voltage.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de tension.

int **get_errorType()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de tension.

voltage→get_friendlyName()

YVoltage

voltage→friendlyName()

voltage.get_friendlyName()

Retourne un identifiant global du capteur de tension au format `NOM_MODULE.NOM_FONCTION`.

String get_friendlyName()

Le chaîne retournée utilise soit les noms logiques du module et du capteur de tension si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du capteur de tension (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de tension en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

voltage→**get_functionDescriptor()****YVoltage****voltage**→**functionDescriptor()****voltage.get_functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera
Y_FUNCTIONDESCRIPTOR_INVALID

voltage→**get_functionId()**

YVoltage

voltage→**functionId()****voltage.get_functionId()**

Retourne l'identifiant matériel du capteur de tension, sans référence au module.

String **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le capteur de tension (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

voltage→**get_hardwareId()****YVoltage****voltage**→**hardwareId()****voltage.get_hardwareId()**

Retourne l'identifiant matériel unique du capteur de tension au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de tension (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le capteur de tension (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

voltage→**get_highestValue()**

YVoltage

voltage→**highestValue()**

voltage.get_highestValue()

Retourne la valeur maximale observée pour la tension depuis le démarrage du module.

`double get_highestValue()`

Retourne :

une valeur numérique représentant la valeur maximale observée pour la tension depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_HIGHESTVALUE_INVALID.

voltage→**get_logFrequency()****YVoltage****voltage**→**logFrequency()****voltage.get_logFrequency()**

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.

String **get_logFrequency()**

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne Y_LOGFREQUENCY_INVALID.

voltage→**get_logicalName()**

YVoltage

voltage→**logicalName()**

voltage.get_logicalName()

Retourne le nom logique du capteur de tension.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du capteur de tension.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

voltage→**get_lowestValue()****YVoltage****voltage**→**lowestValue()****voltage.get_lowestValue()**

Retourne la valeur minimale observée pour la tension depuis le démarrage du module.

```
double get_lowestValue()
```

Retourne :

une valeur numérique représentant la valeur minimale observée pour la tension depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne Y_LOWESTVALUE_INVALID.

voltage→**get_module()**

YVoltage

voltage→**module()**`voltage.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

voltage→**get_recordedData()****YVoltage****voltage**→**recordedData()****voltage.get_recordedData()**

Retourne un objet DataSet représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

YDataSet **get_recordedData**(long **startTime**, long **endTime**)

Veuillez vous référer à la documentation de la classe DataSet pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets DataSet ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la fin de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de YDataSet, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

voltage→get_reportFrequency()

YVoltage

voltage→reportFrequency()

voltage.get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

String get_reportFrequency()

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne Y_REPORTFREQUENCY_INVALID.

voltage→**get_resolution()****YVoltage****voltage**→**resolution()****voltage.get_resolution()**

Retourne la résolution des valeurs mesurées.

double **get_resolution()** ()

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_RESOLUTION_INVALID.

voltage→**get_unit()**

YVoltage

voltage→**unit()****voltage.get_unit()**

Retourne l'unité dans laquelle la tension est exprimée.

String **get_unit()**

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la tension est exprimée

En cas d'erreur, déclenche une exception ou retourne Y_UNIT_INVALID.

voltage→**get_userdata()****YVoltage****voltage**→**userData()****voltage.userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

voltage→**isOnline()****voltage.isOnline()**

YVoltage

Vérifie si le module hébergeant le capteur de tension est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du capteur de tension sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le capteur de tension est joignable, `false` sinon

voltage→**load()****voltage.load()****YVoltage**

Met en cache les valeurs courantes du capteur de tension, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→loadCalibrationPoints()**YVoltage****voltage.loadCalibrationPoints()**

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
int loadCalibrationPoints( ArrayList<Double> rawValues,  
                          ArrayList<Double> refValues)
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→**nextVoltage()****voltage.nextVoltage()****YVoltage**

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstVoltage()`.

`YVoltage` **nextVoltage()**

Retourne :

un pointeur sur un objet `YVoltage` accessible en ligne, ou `null` lorsque l'énumération est terminée.

voltage→registerTimedReportCallback()**YVoltage****voltage.registerTimedReportCallback()**

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

```
int registerTimedReportCallback( TimedReportCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

voltage→registerValueCallback()**YVoltage****voltage.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

voltage→**set_highestValue()**

YVoltage

voltage→**setHighestValue()**

voltage.set_highestValue()

Modifie la mémoire de valeur maximale observée.

```
int set_highestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→**set_logFrequency()****YVoltage****voltage**→**setLogFrequency()****voltage.set_logFrequency()**

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
int set_logFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→**set_logicalName()**

YVoltage

voltage→**setLogicalName()**

voltage.set_logicalName()

Modifie le nom logique du capteur de tension.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de tension.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→**set_lowestValue()****YVoltage****voltage**→**setLowestValue()****voltage.set_lowestValue()**

Modifie la mémoire de valeur minimale observée.

```
int set_lowestValue( double newval)
```

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→**set_reportFrequency()**

YVoltage

voltage→**setReportFrequency()**

voltage.set_reportFrequency()

Modifie la fréquence de notification périodique des valeurs mesurées.

```
int set_reportFrequency( String newval)
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF".

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→**set_resolution()**
voltage→**setResolution()**
voltage.set_resolution()

YVoltage

Modifie la résolution des valeurs physique mesurées.

```
int set_resolution( double newval)
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

voltage→**set_userdata()**

YVoltage

voltage→**setUserData()****voltage.set_userdata()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

void **set_userdata**(Object **data**)

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

3.46. Interface de la fonction Source de tension

La librairie de programmation Yoctopuce permet de commander la tension de sortie du module. Vous pouvez affecter une valeur fixe, ou faire des transitions de voltage.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_vsource.js'></script>
php	require_once('yocto_vsource.php');
c++	#include "yocto_vsource.h"
m	#import "yocto_vsource.h"
pas	uses yocto_vsource;
vb	yocto_vsource.vb
cs	yocto_vsource.cs
java	import com.yoctopuce.YoctoAPI.YVSource;
py	from yocto_vsource import *

Fonction globales
yFindVSource(func) Permet de retrouver une source de tension d'après un identifiant donné.
yFirstVSource() Commence l'énumération des sources de tension accessibles par la librairie.
Méthodes des objets YVSource
vsource→describe() Retourne un court texte décrivant la fonction au format TYPE (NAME) = SERIAL . FUNCTIONID.
vsource→get_advertisedValue() Retourne la valeur courante de la source de tension (pas plus de 6 caractères).
vsource→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
vsource→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.
vsource→get_extPowerFailure() Rend TRUE si le voltage de l'alimentation externe est trop bas.
vsource→get_failure() Indique si le module est en condition d'erreur.
vsource→get_friendlyName() Retourne un identifiant global de la fonction au format NOM_MODULE . NOM_FONCTION.
vsource→get_functionDescriptor() Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
vsource→get_functionId() Retourne l'identifiant matériel de la fonction, sans référence au module.
vsource→get_hardwareId() Retourne l'identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.
vsource→get_logicalName() Retourne le nom logique de la source de tension.
vsource→get_module() Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.
vsource→get_module_async(callback, context)

3. Reference

	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
<code>vsourcesource→get_overCurrent()</code>	Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.
<code>vsourcesource→get_overHeat()</code>	Rend TRUE si le module est en surchauffe.
<code>vsourcesource→get_overLoad()</code>	Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.
<code>vsourcesource→get_regulationFailure()</code>	Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.
<code>vsourcesource→get_unit()</code>	Retourne l'unité dans laquelle la tension est exprimée.
<code>vsourcesource→get_userData()</code>	Retourne le contenu de l'attribut <code>userData</code> , précédemment stocké à l'aide de la méthode <code>set_userData</code> .
<code>vsourcesource→get_voltage()</code>	Retourne la valeur de la commande de tension de sortie en mV
<code>vsourcesource→isOnline()</code>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<code>vsourcesource→isOnline_async(callback, context)</code>	Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.
<code>vsourcesource→load(msValidity)</code>	Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.
<code>vsourcesource→load_async(msValidity, callback, context)</code>	Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.
<code>vsourcesource→nextVSource()</code>	Continue l'énumération des sources de tension commencée à l'aide de <code>yFirstVSource()</code> .
<code>vsourcesource→pulse(voltage, ms_duration)</code>	Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.
<code>vsourcesource→registerValueCallback(callback)</code>	Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.
<code>vsourcesource→set_logicalName(newval)</code>	Modifie le nom logique de la source de tension.
<code>vsourcesource→set_userData(data)</code>	Enregistre un contexte libre dans l'attribut <code>userData</code> de la fonction, afin de le retrouver plus tard à l'aide de la méthode <code>get_userData</code> .
<code>vsourcesource→set_voltage(newval)</code>	Règle la tension de sortie du module (en milliVolts).
<code>vsourcesource→voltageMove(target, ms_duration)</code>	Déclenche une variation constante de la sortie vers une valeur donnée.
<code>vsourcesource→wait_async(callback, context)</code>	Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

yFindVSource() —**YVSource****YVSource.FindVSource()****YVSource.FindVSource()**

Permet de retrouver une source de tension d'après un identifiant donné.

YVSource **FindVSource**(String **func**)

yFindVSource() — **YVSource.FindVSource()****YVSource.FindVSource()**

Permet de retrouver une source de tension d'après un identifiant donné.

js	function yFindVSource (func)
php	function yFindVSource (\$func)
cpp	YVSource* yFindVSource (const string& func)
m	YVSource* yFindVSource (NSString* func)
pas	function yFindVSource (func : string): TYVSource
vb	function yFindVSource (ByVal func As String) As YVSource
cs	YVSource FindVSource (string func)
java	YVSource FindVSource (String func)
py	def FindVSource (func)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que la source de tension soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YVSource.isOnline()` pour tester si la source de tension est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence la source de tension sans ambiguïté

Retourne :

un objet de classe `YVSource` qui permet ensuite de contrôler la source de tension.

yFirstVSource() — **YVSource** **YVSource.FirstVSource()** **YVSource.FirstVSource()**

Commence l'énumération des sources de tension accessibles par la librairie.

YVSource FirstVSource()

yFirstVSource() — **YVSource.FirstVSource()****YVSource.FirstVSource()**

Commence l'énumération des sources de tension accessibles par la librairie.

js	function yFirstVSource()
php	function yFirstVSource()
cpp	YVSource* yFirstVSource()
m	YVSource* yFirstVSource()
pas	function yFirstVSource() : TYVSource
vb	function yFirstVSource() As YVSource
cs	YVSource FirstVSource()
java	YVSource FirstVSource()
py	def FirstVSource()

Utiliser la fonction `YVSource.nextVSource()` pour itérer sur les autres sources de tension.

Retourne :

un pointeur sur un objet `YVSource`, correspondant à la première source de tension accessible en ligne, ou `null` si il n'y a pas de sources de tension disponibles.

vsource→**describe()****vsource.describe()****YVSource**

Retourne un court texte décrivant la fonction au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant la fonction (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

vsource→**get_advertisedValue()****YVSource****vsource**→**advertisedValue()****vsource.get_advertisedValue()**

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

String **get_advertisedValue()**

vsource→**get_advertisedValue()****vsource**→**advertisedValue()****vsource.get_advertisedValue()**

Retourne la valeur courante de la source de tension (pas plus de 6 caractères).

js	function get_advertisedValue()
php	function get_advertisedValue()
cpp	string get_advertisedValue()
m	-(NSString*) advertisedValue
pas	function get_advertisedValue() : string
vb	function get_advertisedValue() As String
cs	string get_advertisedValue()
java	String get_advertisedValue()
py	def get_advertisedValue()
cmd	YVSource target get_advertisedValue

Retourne :

une chaîne de caractères représentant la valeur courante de la source de tension (pas plus de 6 caractères)

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

vsource→**get_errorMessage()****YVSource****vsource**→**errorMessage()****vsource.errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

String **get_errorMessage()****vsource**→**get_errorMessage()****vsource**→**errorMessage()****vsource.errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

js	function get_errorMessage()
php	function get_errorMessage()
cpp	string get_errorMessage()
m	-(NSString*) errorMessage
pas	function get_errorMessage() : string
vb	function get_errorMessage() As String
cs	string get_errorMessage()
java	String get_errorMessage()
py	def get_errorMessage()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la fonction

vsSource→**get_errorType()****YVSource****vsSource**→**errorType()****vsSource.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

```
int get_errorType( )
```

vsSource→**get_errorType()****vsSource**→**errorType()****vsSource.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

<code>js</code>	<code>function get_errorType()</code>
<code>php</code>	<code>function get_errorType()</code>
<code>cpp</code>	<code>YRETCODE get_errorType()</code>
<code>pas</code>	<code>function get_errorType(): YRETCODE</code>
<code>vb</code>	<code>function get_errorType() As YRETCODE</code>
<code>cs</code>	<code>YRETCODE get_errorType()</code>
<code>java</code>	<code>int get_errorType()</code>
<code>py</code>	<code>def get_errorType()</code>

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la fonction

vsources→**get_extPowerFailure()****YVSource****vsources**→**extPowerFailure()****vsources.get_extPowerFailure()**

 Rend TRUE si le voltage de l'alimentation externe est trop bas.

```
int get_extPowerFailure( )
```

vsources→**get_extPowerFailure()****vsources**→**extPowerFailure()****vsources.get_extPowerFailure()**

 Rend TRUE si le voltage de l'alimentation externe est trop bas.

<code>js</code>	<code>function get_extPowerFailure()</code>
<code>php</code>	<code>function get_extPowerFailure()</code>
<code>cpp</code>	<code>Y_EXTPOWERFAILURE_enum get_extPowerFailure()</code>
<code>m</code>	<code>-(Y_EXTPOWERFAILURE_enum) extPowerFailure</code>
<code>pas</code>	<code>function get_extPowerFailure(): Integer</code>
<code>vb</code>	<code>function get_extPowerFailure() As Integer</code>
<code>cs</code>	<code>int get_extPowerFailure()</code>
<code>java</code>	<code>int get_extPowerFailure()</code>
<code>py</code>	<code>def get_extPowerFailure()</code>
<code>cmd</code>	<code>YVSource target get_extPowerFailure</code>

Retourne :

soit Y_EXTPOWERFAILURE_FALSE, soit Y_EXTPOWERFAILURE_TRUE

En cas d'erreur, déclenche une exception ou retourne Y_EXTPOWERFAILURE_INVALID.

vsourc→**get_failure()****YVSource****vsourc**→**failure()****vsourc.get_failure()**

Indique si le module est en condition d'erreur.

int **get_failure()****vsourc**→**get_failure()****vsourc**→**failure()****vsourc.get_failure()**

Indique si le module est en condition d'erreur.

js	function get_failure()
php	function get_failure()
cpp	Y_FAILURE_enum get_failure()
m	-(Y_FAILURE_enum) failure
pas	function get_failure() : Integer
vb	function get_failure() As Integer
cs	int get_failure()
java	int get_failure()
py	def get_failure()
cmd	YVSource target get_failure

Il possible de savoir de quelle erreur il s'agit en testant `get_overheat`, `get_overcurrent` etc... Lorsqu'un condition d'erreur est rencontrée, la tension de sortie est mise à zéro est ne peut pas être changée tant la fonction `reset()` n'aura pas appelée.

Retourne :

soit Y_FAILURE_FALSE, soit Y_FAILURE_TRUE

En cas d'erreur, déclenche une exception ou retourne Y_FAILURE_INVALID.

vsource→**get_friendlyName()****YVSource****vsource**→**friendlyName()****vsource.get_friendlyName()**

Retourne un identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

String **get_friendlyName()**

Le chaîne retournée utilise soit les noms logiques du module et de la fonction si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de la fonction (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant la fonction en utilisant les noms logiques (ex: `MyCustomName.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

vsource→**get_functionDescriptor()****vsource**→**functionDescriptor()****vsource.get_vsourceDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()****vsource**→**get_functionDescriptor()****vsource**→**functionDescriptor()****vsource.get_vsourceDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

js	function get_functionDescriptor()
php	function get_functionDescriptor()
cpp	YFUN_DESCR get_functionDescriptor()
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor() : YFUN_DESCR
vb	function get_functionDescriptor() As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor()
java	String get_functionDescriptor()
py	def get_functionDescriptor()

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR. Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

vsource→**get_functionId()****YVSource****vsource**→**functionId()****vsource.get_vsourceId()**

Retourne l'identifiant matériel de la fonction, sans référence au module.

`String` **get_functionId()**

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant la fonction (ex: `relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

vsourcesrc→get_hardwareId()

YVSource

vsourcesrc→hardwareId()`vsourcesrc.get_hardwareId()`

Retourne l'identifiant matériel unique de la fonction au format `SERIAL.FUNCTIONID`.

String `get_hardwareId()` **()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant la fonction (ex: `RELAYLO1-123456.relay1`) En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

vsource→**get_logicalName()**
vsource→**logicalName()**
vsource.get_logicalName()

YVSource

Retourne le nom logique de la source de tension.

String **get_logicalName()**

vsource→**get_logicalName()**
vsource→**logicalName()****vsource.get_logicalName()**

Retourne le nom logique de la source de tension.

js	function get_logicalName()
php	function get_logicalName()
cpp	string get_logicalName()
m	-(NSString*) logicalName
pas	function get_logicalName() : string
vb	function get_logicalName() As String
cs	string get_logicalName()
java	String get_logicalName()
py	def get_logicalName()
cmd	YVSource target get_logicalName

Retourne :

une chaîne de caractères représentant le nom logique de la source de tension

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

vsSource→**get_module()****YVSource****vsSource**→**module()****vsSource.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule get_module()`**vsSource**→**get_module()****vsSource**→**module()****vsSource.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

js	function get_module ()
php	function get_module ()
cpp	<code>YModule *</code> get_module ()
m	<code>-(YModule*)</code> module
pas	function get_module (): <code>TYModule</code>
vb	function get_module () As <code>YModule</code>
cs	<code>YModule</code> get_module ()
java	<code>YModule</code> get_module ()
py	def get_module ()

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

vsSource→**get_overCurrent()****YVSource****vsSource**→**overCurrent()****vsSource.get_overCurrent()**

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

```
int get_overCurrent( )
```

vsSource→**get_overCurrent()****vsSource**→**overCurrent()****vsSource.get_overCurrent()**

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

js	function get_overCurrent()
php	function get_overCurrent()
cpp	Y_OVERCURRENT_enum get_overCurrent()
m	-(Y_OVERCURRENT_enum) overCurrent
pas	function get_overCurrent() : Integer
vb	function get_overCurrent() As Integer
cs	int get_overCurrent()
java	int get_overCurrent()
py	def get_overCurrent()
cmd	YVSource target get_overCurrent

Retourne :

soit Y_OVERCURRENT_FALSE, soit Y_OVERCURRENT_TRUE

En cas d'erreur, déclenche une exception ou retourne Y_OVERCURRENT_INVALID.

vsSource→**get_overHeat()****YVSource****vsSource**→**overHeat()****vsSource.get_overHeat()**

Rend TRUE si le module est en surchauffe.

```
int get_overHeat( )
```

vsSource→**get_overHeat()****vsSource**→**overHeat()****vsSource.get_overHeat()**

Rend TRUE si le module est en surchauffe.

js	function get_overHeat ()
php	function get_overHeat ()
cpp	Y_OVERHEAT_enum get_overHeat ()
m	-(Y_OVERHEAT_enum) overHeat
pas	function get_overHeat (): Integer
vb	function get_overHeat () As Integer
cs	int get_overHeat ()
java	int get_overHeat ()
py	def get_overHeat ()
cmd	YVSource target get_overHeat

Retourne :

soit Y_OVERHEAT_FALSE, soit Y_OVERHEAT_TRUE

En cas d'erreur, déclenche une exception ou retourne Y_OVERHEAT_INVALID.

vsource→**get_overLoad()****YVSource****vsource**→**overLoad()****vsource.get_overLoad()**

 Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

int **get_overLoad()****vsource**→**get_overLoad()****vsource**→**overLoad()****vsource.get_overLoad()**

 Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

js	function get_overLoad()
php	function get_overLoad()
cpp	Y_OVERLOAD_enum get_overLoad()
m	-(Y_OVERLOAD_enum) overLoad
pas	function get_overLoad() : Integer
vb	function get_overLoad() As Integer
cs	int get_overLoad()
java	int get_overLoad()
py	def get_overLoad()
cmd	YVSource target get_overLoad

Retourne :

soit Y_OVERLOAD_FALSE, soit Y_OVERLOAD_TRUE

En cas d'erreur, déclenche une exception ou retourne Y_OVERLOAD_INVALID.

vsourc→**get_regulationFailure()****YVSource****vsourc**→**regulationFailure()****vsourc.get_regulationFailure()**

 Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

```
int get_regulationFailure( )
```

vsourc→**get_regulationFailure()****vsourc**→**regulationFailure()****vsourc.get_regulationFailure()**

 Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

js	function get_regulationFailure ()
php	function get_regulationFailure ()
cpp	Y_REGULATIONFAILURE_enum get_regulationFailure ()
m	-(Y_REGULATIONFAILURE_enum) regulationFailure
pas	function get_regulationFailure (): Integer
vb	function get_regulationFailure () As Integer
cs	int get_regulationFailure ()
java	int get_regulationFailure ()
py	def get_regulationFailure ()
cmd	YVSource target get_regulationFailure

Retourne :

soit Y_REGULATIONFAILURE_FALSE, soit Y_REGULATIONFAILURE_TRUE

En cas d'erreur, déclenche une exception ou retourne Y_REGULATIONFAILURE_INVALID.

vsource→**get_unit()****YVSource****vsource**→**unit()****vsource.get_unit()**

 Retourne l'unité dans laquelle la tension est exprimée.

String **get_unit()****vsource**→**get_unit()****vsource**→**unit()****vsource.get_unit()**

 Retourne l'unité dans laquelle la tension est exprimée.

js	function get_unit()
php	function get_unit()
cpp	string get_unit()
m	-(NSString*) unit
pas	function get_unit() : string
vb	function get_unit() As String
cs	string get_unit()
java	String get_unit()
py	def get_unit()
cmd	YVSource target get_unit

Retourne :

une chaîne de caractères représentant l'unité dans laquelle la tension est exprimée

En cas d'erreur, déclenche une exception ou retourne Y_UNIT_INVALID.

vsource→**get_userData()****YVSource****vsource**→**userData()****vsource.get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

Object **get_userData()****vsource**→**get_userData()****vsource**→**userData()****vsource.get_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.

js	function get_userData()
php	function get_userData()
cpp	void * get_userData()
m	-(void*) userData
pas	function get_userData() : Tobject
vb	function get_userData() As Object
cs	object get_userData()
java	Object get_userData()
py	def get_userData()

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

vsource→**get_voltage()****YVSource****vsource**→**voltage()****vsource.get_voltage()**

Retourne la valeur de la commande de tension de sortie en mV

int **get_voltage()****vsource**→**get_voltage()****vsource**→**voltage()****vsource.get_voltage()**

Retourne la valeur de la commande de tension de sortie en mV

js	function get_voltage()
php	function get_voltage()
cpp	int get_voltage()
m	-(int) voltage
pas	function get_voltage() : LongInt
vb	function get_voltage() As Integer
cs	int get_voltage()
java	int get_voltage()
py	def get_voltage()

Retourne :

un entier représentant la valeur de la commande de tension de sortie en mV

En cas d'erreur, déclenche une exception ou retourne Y_VOLTAGE_INVALID.

vssource→**isOnline()****vs**source.**isOnline()**

YVSource

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

boolean **isOnline()**

vssource→**isOnline()****vs**source.**isOnline()**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

js	function isOnline()
php	function isOnline()
cpp	bool isOnline()
m	-(BOOL) isOnline
pas	function isOnline() : boolean
vb	function isOnline() As Boolean
cs	bool isOnline()
java	boolean isOnline()
py	def isOnline()

Si les valeurs des attributs en cache de la fonction sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

true si la fonction est joignable, false sinon

vsource→**load()****vsource.load()****YVSource**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

vsource→**load()****vsource.load()**

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

js	function load(msValidity)
php	function load(\$msValidity)
cpp	YRETCODE load(int msValidity)
m	-(YRETCODE) load : (int) msValidity
pas	function load(msValidity: integer): YRETCODE
vb	function load(ByVal msValidity As Integer) As YRETCODE
cs	YRETCODE load(int msValidity)
java	int load(long msValidity)
py	def load(msValidity)

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

vsSource→**nextVSource()****vsSource.nextVSource()****YVSource**

Continue l'énumération des sources de tension commencée à l'aide de `yFirstVSource()`.

`YVSource` **nextVSource()**

vsSource→**nextVSource()****vsSource.nextVSource()**

Continue l'énumération des sources de tension commencée à l'aide de `yFirstVSource()`.

js	function nextVSource()
php	function nextVSource()
cpp	<code>YVSource *</code> nextVSource()
m	-(<code>YVSource*</code>) nextVSource
pas	function nextVSource() : <code>TYVSource</code>
vb	function nextVSource() As <code>YVSource</code>
cs	<code>YVSource</code> nextVSource()
java	<code>YVSource</code> nextVSource()
py	def nextVSource()

Retourne :

un pointeur sur un objet `YVSource` accessible en ligne, ou `null` lorsque l'énumération est terminée.

vsource→**pulse()**`vsource.pulse()`**YVSource**

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

```
int pulse( int voltage, int ms_duration)
```

vsource→**pulse()**`vsource.pulse()`

Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

js	function pulse (voltage , ms_duration)
php	function pulse (\$voltage , \$ms_duration)
cpp	int pulse (int voltage , int ms_duration)
m	-(int) pulse : (int) voltage : (int) ms_duration
pas	function pulse (voltage : integer, ms_duration : integer): integer
vb	function pulse (ByVal voltage As Integer, ByVal ms_duration As Integer) As Integer
cs	int pulse (int voltage , int ms_duration)
java	int pulse (int voltage , int ms_duration)
py	def pulse (voltage , ms_duration)
cmd	YVSource target pulse voltage ms_duration

Paramètres :

voltage tension demandée, en millivolts
ms_duration durée de l'impulsion, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

vsources.registerValueCallback()**YVSource****vsources.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
void registerValueCallback( UpdateCallback callback)
```

vsources.registerValueCallback()vsources.registerValueCallback()

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

js	function registerValueCallback(callback)
php	function registerValueCallback(\$callback)
cpp	void registerValueCallback(YDisplayUpdateCallback callback)
pas	procedure registerValueCallback(callback: TGenericUpdateCallback)
vb	procedure registerValueCallback(ByVal callback As GenericUpdateCallback)
cs	void registerValueCallback(UpdateCallback callback)
java	void registerValueCallback(UpdateCallback callback)
py	def registerValueCallback(callback)
m	-(void) registerValueCallback : (YFunctionUpdateCallback) callback

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

vsource→**set_logicalName()**
vsource→**setLogicalName()**
vsource.set_logicalName()

YVSource

Modifie le nom logique de la source de tension.

```
int set_logicalName( String newval)
```

vsource→**set_logicalName()**
vsource→**setLogicalName()****vsource.set_logicalName()**

Modifie le nom logique de la source de tension.

js	function set_logicalName (newval)
php	function set_logicalName (\$newval)
cpp	int set_logicalName (const string& newval)
m	-(int) setLogicalName : (NSString*) newval
pas	function set_logicalName (newval : string): integer
vb	function set_logicalName (ByVal newval As String) As Integer
cs	int set_logicalName (string newval)
java	int set_logicalName (String newval)
py	def set_logicalName (newval)
cmd	YVSource target set_logicalName newval

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de la source de tension

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

vsSource→**set_userData()****YVSource****vsSource**→**setUserData()****vsSource.set_userData()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userData.

```
void set_userData( Object data)
```

vsSource→**set_userData()****vsSource**→**setUserData()****vsSource.set_userData()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userData.

js	function set_userData(data)
php	function set_userData(\$data)
cpp	void set_userData(void* data)
m	-(void) setUserData : (void*) data
pas	procedure set_userData(data: Tobject)
vb	procedure set_userData(ByVal data As Object)
cs	void set_userData(object data)
java	void set_userData(Object data)
py	def set_userData(data)

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

vsource→**set_voltage()****YVSource****vsource**→**setVoltage()****vsource.set_voltage()**

 Règle la tension de sortie du module (en milliVolts).

```
int set_voltage( int newval)
```

vsource→**set_voltage()****vsource**→**setVoltage()****vsource.set_voltage()**

 Règle la tension de sortie du module (en milliVolts).

js	function set_voltage(newval)
php	function set_voltage(\$newval)
cpp	int set_voltage(int newval)
m	-(int) setVoltage : (int) newval
pas	function set_voltage(newval: LongInt): integer
vb	function set_voltage(ByVal newval As Integer) As Integer
cs	int set_voltage(int newval)
java	int set_voltage(int newval)
py	def set_voltage(newval)
cmd	YVSource target set_voltage newval

Paramètres :**newval** un entier**Retourne :**

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

vsourcesource→voltageMove()vsourcesource.voltageMove()**YVSource**

Déclenche une variation constante de la sortie vers une valeur donnée.

```
int voltageMove( int target, int ms_duration)
```

vsourcesource→voltageMove()vsourcesource.voltageMove()

Déclenche une variation constante de la sortie vers une valeur donnée.

js	function voltageMove(target, ms_duration)
php	function voltageMove(\$target, \$ms_duration)
cpp	int voltageMove(int target, int ms_duration)
m	-(int) voltageMove : (int) target : (int) ms_duration
pas	function voltageMove(target: integer, ms_duration: integer): integer
vb	function voltageMove(ByVal target As Integer, ByVal ms_duration As Integer) As Integer
cs	int voltageMove(int target, int ms_duration)
java	int voltageMove(int target, int ms_duration)
py	def voltageMove(target, ms_duration)
cmd	YVSource target voltageMove target ms_duration

Paramètres :

target nouvelle valeur de sortie à la fin de la transition, en milliVolts.
ms_duration durée de la transition, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.47. Interface de la fonction WakeUpMonitor

La fonction WakeUpMonitor prend en charge le contrôle global de toutes les sources de réveil possibles ainsi que les mises en sommeil automatiques.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_wakeupmonitor.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YWakeUpMonitor = yoctolib.YWakeUpMonitor;
php	require_once('yocto_wakeupmonitor.php');
c++	#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.

YWakeUpMonitor.FindWakeUpMonitor()**YWakeUpMonitor****yFindWakeUpMonitor()****YWakeUpMonitor.FindWakeUpMonitor()**

Permet de retrouver un moniteur d'après un identifiant donné.

YWakeUpMonitor **FindWakeUpMonitor**(String **func**)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le moniteur soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWakeUpMonitor.isOnline()` pour tester si le moniteur est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le moniteur sans ambiguïté

Retourne :

un objet de classe `YWakeUpMonitor` qui permet ensuite de contrôler le moniteur.

YWakeUpMonitor.FirstWakeUpMonitor()**YWakeupMonitor****yFirstWakeUpMonitor()****YWakeupMonitor.FirstWakeUpMonitor()**

Commence l'énumération des Moniteurs accessibles par la librairie.

`YWakeupMonitor` [FirstWakeUpMonitor\(\)](#)

Utiliser la fonction `YWakeupMonitor.nextWakeUpMonitor()` pour itérer sur les autres Moniteurs.

Retourne :

un pointeur sur un objet `YWakeupMonitor`, correspondant au premier moniteur accessible en ligne, ou `null` si il n'y a pas de Moniteurs disponibles.

wakeupmonitor→describe()**YWakeUpMonitor****wakeupmonitor.describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du moniteur au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String describe()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le moniteur (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

wakeupmonitor→get_advertisedValue()**YWakeUpMonitor****wakeupmonitor→advertisedValue()****wakeupmonitor.get_advertisedValue()**

Retourne la valeur courante du moniteur (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du moniteur (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

wakeupmonitor→get_errorMessage()

YWakeUpMonitor

wakeupmonitor→errorMessage()

wakeupmonitor.get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

String get_errorMessage()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du moniteur.

wakeupmonitor→get_errorType()**YWakeUpMonitor****wakeupmonitor→errorType()****wakeupmonitor.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du moniteur.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du moniteur.

wakeupmonitor→get_friendlyName()

YWakeUpMonitor

wakeupmonitor→friendlyName()

wakeupmonitor.get_friendlyName()

Retourne un identifiant global du moniteur au format `NOM_MODULE.NOM_FONCTION`.

String get_friendlyName()

Le chaîne retournée utilise soit les noms logiques du module et du moniteur si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du moniteur (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le moniteur en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

wakeupmonitor→**get_functionDescriptor()****YWakeUpMonitor****wakeupmonitor**→**functionDescriptor()****wakeupmonitor.get_functionDescriptor()**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de `YFunction` référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type `YFUN_DESCR`.

Si la fonction n'a jamais été contactée, la valeur retournée sera `Y_FUNCTIONDESCRIPTOR_INVALID`

wakeupmonitor→get_functionId()

YWakeUpMonitor

wakeupmonitor→functionId()

wakeupmonitor.get_functionId()

Retourne l'identifiant matériel du moniteur, sans référence au module.

String **get_functionId()** ()

Par exemple relay1.

Retourne :

une chaîne de caractères identifiant le moniteur (ex: relay1)

En cas d'erreur, déclenche une exception ou retourne Y_FUNCTIONID_INVALID.

wakeupmonitor→get_hardwareId()**YWakeUpMonitor****wakeupmonitor→hardwareId()****wakeupmonitor.get_hardwareId()**

Retourne l'identifiant matériel unique du moniteur au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du moniteur (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le moniteur (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

wakeupmonitor→get_logicalName()

YWakeUpMonitor

wakeupmonitor→logicalName()

wakeupmonitor.get_logicalName()

Retourne le nom logique du moniteur.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du moniteur.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

wakeupmonitor→**get_module()****YWakeUpMonitor****wakeupmonitor**→**module()****wakeupmonitor.get_module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

wakeupmonitor→**get_nextWakeUp()**

YWakeUpMonitor

wakeupmonitor→**nextWakeUp()**

wakeupmonitor.get_nextWakeUp()

Retourne la prochaine date/heure de réveil agendée (format UNIX)

long **get_nextWakeUp()**

Retourne :

un entier représentant la prochaine date/heure de réveil agendée (format UNIX)

En cas d'erreur, déclenche une exception ou retourne Y_NEXTWAKEUP_INVALID.

wakeupmonitor→get_powerDuration()**YWakeUpMonitor****wakeupmonitor→powerDuration()****wakeupmonitor.get_powerDuration()**

Retourne le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

int **get_powerDuration()** ()

Retourne :

un entier représentant le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement

En cas d'erreur, déclenche une exception ou retourne Y_POWERDURATION_INVALID.

wakeupmonitor→get_sleepCountdown()

YWakeUpMonitor

wakeupmonitor→sleepCountdown()

wakeupmonitor.get_sleepCountdown()

Retourne le temps avant le prochain sommeil.

`int get_sleepCountdown()`

Retourne :

un entier représentant le temps avant le prochain sommeil

En cas d'erreur, déclenche une exception ou retourne Y_SLEEP_COUNTDOWN_INVALID.

wakeupmonitor→get_userdata()**YWakeUpMonitor****wakeupmonitor→userData()****wakeupmonitor.get_userdata()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.

Object **get_userdata()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

wakeupmonitor→**get_wakeUpReason()**

YWakeUpMonitor

wakeupmonitor→**wakeUpReason()**

wakeupmonitor.get_wakeUpReason()

Renvoie la raison du dernier réveil.

int **get_wakeUpReason()**

Retourne :

une valeur parmi Y_WAKEUPREASON_USBPOWER, Y_WAKEUPREASON_EXTPOWER,
Y_WAKEUPREASON_ENDOFSLEEP, Y_WAKEUPREASON_EXTSIG1,
Y_WAKEUPREASON_SCHEDULE1 et Y_WAKEUPREASON_SCHEDULE2

En cas d'erreur, déclenche une exception ou retourne Y_WAKEUPREASON_INVALID.

wakeupmonitor→get_wakeUpState()**YWakeUpMonitor****wakeupmonitor→wakeUpState()****wakeupmonitor.get_wakeUpState()**

Revoie l'état actuel du moniteur

int **get_wakeUpState()**

Retourne :

soit Y_WAKEUPSTATE_SLEEPING, soit Y_WAKEUPSTATE_AWAKE

En cas d'erreur, déclenche une exception ou retourne Y_WAKEUPSTATE_INVALID.

wakeupmonitor→isOnline()

YWakeUpMonitor

wakeupmonitor.isOnline()

Vérifie si le module hébergeant le moniteur est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache du moniteur sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

true si le moniteur est joignable, false sinon

wakeupmonitor→**load()**`wakeupmonitor.load()`**YWakeUpMonitor**

Met en cache les valeurs courantes du moniteur, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**nextWakeUpMonitor()**

YWakeUpMonitor

wakeupmonitor.nextWakeUpMonitor()

Continue l'énumération des Moniteurs commencée à l'aide de `yFirstWakeUpMonitor()`.

`YWakeUpMonitor` **nextWakeUpMonitor()**

Retourne :

un pointeur sur un objet `YWakeUpMonitor` accessible en ligne, ou `null` lorsque l'énumération est terminée.

wakeupmonitor→registerValueCallback()**YWakeUpMonitor****wakeupmonitor.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

wakeupmonitor→resetSleepCountDown()

YWakeUpMonitor

wakeupmonitor.resetSleepCountDown()

Réinitialise le compteur de mise en sommeil.

int resetSleepCountDown()

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**set_logicalName()**
wakeupmonitor→**setLogicalName()**
wakeupmonitor.set_logicalName()

YWakeUpMonitor

Modifie le nom logique du moniteur.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du moniteur.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**set_nextWakeUp()**

YWakeUpMonitor

wakeupmonitor→**setNextWakeUp()**

wakeupmonitor.set_nextWakeUp()

Modifie les jours de la semaine où un réveil doit avoir lieu.

```
int set_nextWakeUp( long newval)
```

Paramètres :

newval un entier représentant les jours de la semaine où un réveil doit avoir lieu

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**set_powerDuration()****YWakeUpMonitor****wakeupmonitor**→**setPowerDuration()****wakeupmonitor.set_powerDuration()**

Modifie le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

int **set_powerDuration**(int **newval**)

Paramètres :

newval un entier représentant le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**set_sleepCountdown()**
wakeupmonitor→**setSleepCountdown()**
wakeupmonitor.set_sleepCountdown()

YWakeUpMonitor

Modifie le temps avant le prochain sommeil .

```
int set_sleepCountdown( int newval)
```

Paramètres :

newval un entier représentant le temps avant le prochain sommeil

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→set_userdata()**YWakeUpMonitor****wakeupmonitor→setUserData()****wakeupmonitor.set_userdata()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

```
void set_userdata( Object data)
```

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

wakeupmonitor→**sleep()**`wakeupmonitor.sleep()`

YWakeUpMonitor

Déclenche une mise en sommeil jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

```
int sleep( int secBeforeSleep)
```

Paramètres :

secBeforeSleep nombre de seconde avant la mise en sommeil

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→sleepFor()**YWakeUpMonitor****wakeupmonitor.sleepFor()**

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.

```
int sleepFor( int secUntilWakeUp, int secBeforeSleep)
```

Le compte à rebours avant la mise en sommeil peut être annulé grâce à resetSleepCountDown.

Paramètres :

secUntilWakeUp nombre de secondes avant le prochain réveil

secBeforeSleep nombre de secondes avant la mise en sommeil

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→sleepUntil()

YWakeUpMonitor

wakeupmonitor.sleepUntil()

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.

```
int sleepUntil( int wakeUpTime, int secBeforeSleep)
```

Le compte à rebours avant la mise en sommeil peut être annulé grâce à resetSleepCountDown.

Paramètres :

wakeUpTime date/heure du réveil (format UNIX)

secBeforeSleep nombre de secondes avant la mise en sommeil

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupmonitor→**wakeUp()**
wakeupmonitor.wakeUp()

YWakeUpMonitor

Force un réveil.

```
int wakeUp( )
```

3.48. Interface de la fonction WakeUpSchedule

La fonction WakeUpSchedule implémente une condition de réveil. Le réveil est spécifiée par un ensemble de mois et/ou jours et/ou heures et/ou minutes où il doit se produire.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_wakeupschedule.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YWakeUpSchedule = yoctolib.YWakeUpSchedule;
php	require_once('yocto_wakeupschedule.php');
c++	#include "yocto_wakeupschedule.h"
m	#import "yocto_wakeupschedule.h"
pas	uses yocto_wakeupschedule;
vb	yocto_wakeupschedule.vb
cs	yocto_wakeupschedule.cs
java	import com.yoctopuce.YoctoAPI.YWakeUpSchedule;
py	from yocto_wakeupschedule import *

Fonction globales
yFindWakeUpSchedule(func) Permet de retrouver un réveil agendé d'après un identifiant donné.
yFirstWakeUpSchedule() Commence l'énumération des réveils agendés accessibles par la librairie.
Méthodes des objets YWakeUpSchedule
wakeupschedule→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance du réveil agendé au format TYPE (NAME) = SERIAL . FUNCTIONID.
wakeupschedule→get_advertisedValue() Retourne la valeur courante du réveil agendé (pas plus de 6 caractères).
wakeupschedule→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.
wakeupschedule→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.
wakeupschedule→get_friendlyName() Retourne un identifiant global du réveil agendé au format NOM_MODULE . NOM_FONCTION.
wakeupschedule→get_functionDescriptor() Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
wakeupschedule→get_functionId() Retourne l'identifiant matériel du réveil agendé, sans référence au module.
wakeupschedule→get_hardwareId() Retourne l'identifiant matériel unique du réveil agendé au format SERIAL . FUNCTIONID.
wakeupschedule→get_hours() Retourne les heures où le réveil est actif..
wakeupschedule→get_logicalName() Retourne le nom logique du réveil agendé.
wakeupschedule→get_minutes() Retourne toutes les minutes de chaque heure où le réveil est actif.
wakeupschedule→get_minutesA()

Retourne les minutes de l'intervall 00-29 de chaque heures où le réveil est actif.

wakeupschedule→get_minutesB()

Retourne les minutes de l'intervall 30-59 de chaque heure où le réveil est actif.

wakeupschedule→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

wakeupschedule→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

wakeupschedule→get_monthDays()

Retourne les jours du mois où le réveil est actif..

wakeupschedule→get_months()

Retourne les mois où le réveil est actif..

wakeupschedule→get_nextOccurence()

Retourne la date/heure de la prochaine occurrence de réveil

wakeupschedule→get_userData()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

wakeupschedule→get_weekDays()

Retourne les jours de la semaine où le réveil est actif..

wakeupschedule→isOnline()

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

wakeupschedule→isOnline_async(callback, context)

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

wakeupschedule→load(msValidity)

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

wakeupschedule→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

wakeupschedule→nextWakeUpSchedule()

Continue l'énumération des réveils agendés commencée à l'aide de `yFirstWakeUpSchedule()`.

wakeupschedule→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

wakeupschedule→set_hours(newval, newval)

Modifie les heures où un réveil doit avoir lieu

wakeupschedule→set_logicalName(newval)

Modifie le nom logique du réveil agendé.

wakeupschedule→set_minutes(bitmap)

Modifie toutes les minutes où un réveil doit avoir lieu

wakeupschedule→set_minutesA(newval, newval)

Modifie les minutes de l'intervall 00-29 où un réveil doit avoir lieu

wakeupschedule→set_minutesB(newval)

Modifie les minutes de l'intervall 30-59 où un réveil doit avoir lieu.

wakeupschedule→set_monthDays(newval, newval)

Modifie les jours du mois où un réveil doit avoir lieu

wakeupschedule→set_months(newval, newval)

Modifie les mois où un réveil doit avoir lieu

wakeupschedule→set_userData(data)

3. Reference

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

wakeupschedule→**set_weekDays**(**newval**, **newval**)

Modifie les jours de la semaine où un réveil doit avoir lieu

wakeupschedule→**wait_async**(**callback**, **context**)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YWakeUpSchedule.FindWakeUpSchedule() yFindWakeUpSchedule()

YWakeupSchedule

YWakeupSchedule.FindWakeUpSchedule()

Permet de retrouver un réveil agendé d'après un identifiant donné.

```
YWakeupSchedule FindWakeUpSchedule( String func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le réveil agendé soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWakeupSchedule.isOnline()` pour tester si le réveil agendé est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le réveil agendé sans ambiguïté

Retourne :

un objet de classe `YWakeupSchedule` qui permet ensuite de contrôler le réveil agendé.

YWakeUpSchedule.FirstWakeUpSchedule()
yFirstWakeUpSchedule()
YWakeUpSchedule.FirstWakeUpSchedule()

YWakeUpSchedule

Commence l'énumération des réveils agendés accessibles par la librairie.

[YWakeUpSchedule](#) **FirstWakeUpSchedule()**

Utiliser la fonction `YWakeUpSchedule.nextWakeUpSchedule()` pour itérer sur les autres réveils agendés.

Retourne :

un pointeur sur un objet `YWakeUpSchedule`, correspondant au premier réveil agendé accessible en ligne, ou `null` si il n'y a pas de réveils agendés disponibles.

wakeupschedule→describe()**YWakeUpSchedule****wakeupschedule.describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du réveil agendé au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String **describe()**

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le réveil agendé (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

wakeupschedule→**get_advertisedValue()**

YWakeUpSchedule

wakeupschedule→**advertisedValue()**

wakeupschedule.get_advertisedValue()

Retourne la valeur courante du réveil agendé (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du réveil agendé (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

wakeupschedule→get_errorMessage()**YWakeUpSchedule****wakeupschedule→errorMessage()****wakeupschedule.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du réveil agendé.

wakeupschedule→get_errorType()

YWakeUpSchedule

wakeupschedule→errorType()

wakeupschedule.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du réveil agendé.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du réveil agendé.

wakeupschedule→get_friendlyName()**YWakeUpSchedule****wakeupschedule→friendlyName()****wakeupschedule.get_friendlyName()**

Retourne un identifiant global du réveil agendé au format `NOM_MODULE.NOM_FONCTION`.

String `get_friendlyName()`

Le chaîne retournée utilise soit les noms logiques du module et du réveil agendé si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du réveil agendé (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le réveil agendé en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

wakeupschedule→**get_functionDescriptor()**

YWakeUpSchedule

wakeupschedule→**functionDescriptor()**

wakeupschedule.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

wakeupschedule→get_functionId()**YWakeUpSchedule****wakeupschedule→functionId()****wakeupschedule.get_functionId()**

Retourne l'identifiant matériel du réveil agendé, sans référence au module.

String **get_functionId()** ()

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le réveil agendé (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

wakeupschedule→**get_hardwareId()**

YWakeUpSchedule

wakeupschedule→**hardwareId()**

wakeupschedule.get_hardwareId()

Retourne l'identifiant matériel unique du réveil agendé au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du réveil agendé (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le réveil agendé (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

wakeupschedule→get_hours()**YWakeUpSchedule****wakeupschedule→hours()****wakeupschedule.get_hours()**

Retourne les heures où le réveil est actif..

int **get_hours()**

Retourne :

un entier représentant les heures où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y_HOURS_INVALID.

wakeupschedule→get_logicalName()

YWakeUpSchedule

wakeupschedule→logicalName()

wakeupschedule.get_logicalName()

Retourne le nom logique du réveil agendé.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du réveil agendé.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

wakeupschedule→get_minutes()**YWakeUpSchedule****wakeupschedule→minutes()****wakeupschedule.get_minutes()**

Retourne toutes les minutes de chaque heure où le réveil est actif.

```
long get_minutes( )
```

wakeupschedule→get_minutesA()

YWakeUpSchedule

wakeupschedule→minutesA()

wakeupschedule.get_minutesA()

Retourne les minutes de l'interval 00-29 de chaque heures où le réveil est actif.

int get_minutesA()

Retourne :

un entier représentant les minutes de l'interval 00-29 de chaque heures où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y_MINUTESA_INVALID.

wakeupschedule→get_minutesB()**YWakeUpSchedule****wakeupschedule→minutesB()****wakeupschedule.get_minutesB()**

Retourne les minutes de l'interval 30-59 de chaque heure où le réveil est actif.

int **get_minutesB()**

Retourne :

un entier représentant les minutes de l'interval 30-59 de chaque heure où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y_MINUTESB_INVALID.

wakeupschedule→**get_module()**

YWakeUpSchedule

wakeupschedule→**module()**

wakeupschedule.get_module()

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

YModule **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :

une instance de YModule

wakeupschedule→get_monthDays()**YWakeUpSchedule****wakeupschedule→monthDays()****wakeupschedule.get_monthDays()**

Retourne les jours du mois où le réveil est actif..

int **get_monthDays()**

Retourne :

un entier représentant les jours du mois où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y_MONTHDAYS_INVALID.

wakeupschedule→**get_months()**

YWakeUpSchedule

wakeupschedule→**months()**

wakeupschedule.get_months()

Retourne les mois où le réveil est actif..

```
int get_months( )
```

Retourne :

un entier représentant les mois où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y_MONTHS_INVALID.

wakeupschedule→get_nextOccurence()**YWakeUpSchedule****wakeupschedule→nextOccurence()****wakeupschedule.get_nextOccurence()**

Retourne la date/heure de la prochaine occurrence de réveil

`long get_nextOccurence()`

Retourne :

un entier représentant la date/heure de la prochaine occurrence de réveil

En cas d'erreur, déclenche une exception ou retourne Y_NEXTOCCURENCE_INVALID.

wakeupschedule→get_userdata()

YWakeUpSchedule

wakeupschedule→userdata()

wakeupschedule.get_userdata()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

wakeupschedule→get_weekDays()**YWakeUpSchedule****wakeupschedule→weekDays()****wakeupschedule.get_weekDays ()**

Retourne les jours de la semaine où le réveil est actif..

int **get_weekDays**()

Retourne :

un entier représentant les jours de la semaine où le réveil est actif

En cas d'erreur, déclenche une exception ou retourne Y_WEEKDAYS_INVALID.

wakeupschedule→**isOnline()**

YWakeUpSchedule

wakeupschedule.isOnline()

Vérifie si le module hébergeant le réveil agendé est joignable, sans déclencher d'erreur.

boolean **isOnline()**

Si les valeurs des attributs en cache du réveil agendé sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

true si le réveil agendé est joignable, false sinon

wakeupschedule→load()`wakeupschedule.load()`**YWakeUpSchedule**

Met en cache les valeurs courantes du réveil agendé, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→**nextWakeUpSchedule()**

YWakeUpSchedule

wakeupschedule.nextWakeUpSchedule()

Continue l'énumération des réveils agendés commencée à l'aide de `yFirstWakeUpSchedule()`.

`YWakeUpSchedule` **nextWakeUpSchedule()**

Retourne :

un pointeur sur un objet `YWakeUpSchedule` accessible en ligne, ou `null` lorsque l'énumération est terminée.

wakeupschedule→registerValueCallback()**YWakeUpSchedule****wakeupschedule.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

wakeupschedule→**set_hours()**

YWakeUpSchedule

wakeupschedule→**setHours()**

wakeupschedule.set_hours()

Modifie les heures où un réveil doit avoir lieu

```
int set_hours( int newval)
```

Paramètres :

newval un entier représentant les heures où un réveil doit avoir lieu

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set_logicalName()**YWakeUpSchedule****wakeupschedule→setLogicalName()****wakeupschedule.set_logicalName()**

Modifie le nom logique du réveil agendé.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du réveil agendé.

Retourne :

`YAPI_SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→**set_minutes()**

YWakeUpSchedule

wakeupschedule→**setMinutes()**

wakeupschedule.set_minutes()

Modifie toutes les minutes où un réveil doit avoir lieu

```
int set_minutes( long bitmap)
```

Paramètres :

bitmap Minutes 00-59 de chaque heure où le réveil est actif.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set_minutesA()**YWakeUpSchedule****wakeupschedule→setMinutesA()****wakeupschedule.set_minutesA()**

Modifie les minutes de l'interval 00-29 où un réveil doit avoir lieu

```
int set_minutesA( int newval)
```

Paramètres :

newval un entier représentant les minutes de l'interval 00-29 où un réveil doit avoir lieu

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set_minutesB()

YWakeUpSchedule

wakeupschedule→setMinutesB()

wakeupschedule.set_minutesB()

Modifie les minutes de l'interval 30-59 où un réveil doit avoir lieu.

```
int set_minutesB( int newval)
```

Paramètres :

newval un entier représentant les minutes de l'interval 30-59 où un réveil doit avoir lieu

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set_monthDays()**YWakeUpSchedule****wakeupschedule→setMonthDays()****wakeupschedule.set_monthDays ()**

Modifie les jours du mois où un réveil doit avoir lieu

```
int set_monthDays( int newval)
```

Paramètres :

newval un entier représentant les jours du mois où un réveil doit avoir lieu

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→**set_months()**

YWakeUpSchedule

wakeupschedule→**setMonths()**

wakeupschedule.set_months()

Modifie les mois où un réveil doit avoir lieu

```
int set_months( int newval)
```

Paramètres :

newval un entier représentant les mois où un réveil doit avoir lieu

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wakeupschedule→set_userdata()**YWakeUpSchedule****wakeupschedule→setUserData()****wakeupschedule.set_userdata()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

```
void set_userdata( Object data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

wakeupschedule→**set_weekDays()**

YWakeUpSchedule

wakeupschedule→**setWeekDays()**

wakeupschedule.set_weekDays()

Modifie les jours de la semaine où un réveil doit avoir lieu

```
int set_weekDays( int newval)
```

Paramètres :

newval un entier représentant les jours de la semaine où un réveil doit avoir lieu

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

3.49. Interface de la fonction Watchdog

La fonction WatchDog est gérée comme un relais qui couperait brièvement l'alimentation d'un appareil après un d'attente temps donné afin de provoquer une réinitialisation complète de cet appareil. Il suffit d'appeler le watchdog à intervalle régulier pour l'empêcher de provoquer la réinitialisation. Le watchdog peut aussi être piloté directement à l'aide des méthode *pulse* et *delayedpulse* pour éteindre un appareil pendant un temps donné.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_watchdog.js'></script>
nodejs	var yoctolib = require('yoctolib'); var YWatchdog = yoctolib.YWatchdog;
php	require_once('yocto_watchdog.php');
c++	#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 watchdog, en millisecondes.

watchdog→get_triggerDuration()

Retourne la durée d'un reset généré par le watchdog, en millisecondes.

watchdog→get_userData()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

watchdog→isOnline()

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

watchdog→isOnline_async(callback, context)

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

watchdog→load(msValidity)

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

watchdog→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

watchdog→nextWatchdog()

Continue l'énumération des watchdog commencée à l'aide de `yFirstWatchdog()`.

watchdog→pulse(ms_duration)

Commute le relais à l'état B (actif) pour une durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

watchdog→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

watchdog→resetWatchdog()

Réinitialise le WatchDog.

watchdog→set_autoStart(newval)

Modifie l'état du watching au démarrage du module.

watchdog→set_logicalName(newval)

Modifie le nom logique du watchdog.

watchdog→set_maxTimeOnStateA(newval)

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

watchdog→set_maxTimeOnStateB(newval)

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

watchdog→set_output(newval)

Modifie l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

watchdog→set_running(newval)

Modifie manuellement l'état de fonctionnement du watchdog.

watchdog→set_state(newval)

Modifie l'état du watchdog (A pour la position de repos, B pour l'état actif).

watchdog→set_stateAtPowerOn(newval)

Pré-programme l'état du watchdog au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

watchdog→set_triggerDelay(newval)

Modifie le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes.

watchdog→set_triggerDuration(newval)

Modifie la durée des resets générés par le watchdog, en millisecondes.

watchdog→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

watchdog→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YWatchdog.FindWatchdog()**YWatchdog****yFindWatchdog()YWatchdog.FindWatchdog()**

Permet de retrouver un watchdog d'après un identifiant donné.

YWatchdog FindWatchdog(String func)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le watchdog soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWatchdog.isOnline()` pour tester si le watchdog est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence le watchdog sans ambiguïté

Retourne :

un objet de classe `YWatchdog` qui permet ensuite de contrôler le watchdog.

YWatchdog.FirstWatchdog()**YWatchdog****yFirstWatchdog()****YWatchdog.FirstWatchdog()**

Commence l'énumération des watchdog accessibles par la librairie.

YWatchdog **FirstWatchdog()**

Utiliser la fonction `YWatchdog.nextWatchdog()` pour itérer sur les autres watchdog.

Retourne :

un pointeur sur un objet `YWatchdog`, correspondant au premier watchdog accessible en ligne, ou `null` si il n'y a pas de watchdog disponibles.

watchdog→delayedPulse()

YWatchdog

watchdog.delayedPulse()

Pré-programme une impulsion

```
int delayedPulse( int ms_delay, int ms_duration)
```

Paramètres :

ms_delay delai d'attente avant l'impulsion, en millisecondes

ms_duration durée de l'impulsion, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→describe()`watchdog.describe()`**YWatchdog**

Retourne un court texte décrivant de manière non-ambigüe l'instance du watchdog au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String `describe()`

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un debugueur.

Retourne :

une chaîne de caractères décrivant le watchdog (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

watchdog→**get_advertisedValue()**

YWatchdog

watchdog→**advertisedValue()**

watchdog.get_advertisedValue()

Retourne la valeur courante du watchdog (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante du watchdog (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

watchdog→get_autoStart()**YWatchdog****watchdog→autoStart()**`watchdog.get_autoStart()`

Retourne l'état du watchdog à la mise sous tension du module.

`int get_autoStart()`

Retourne :

soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon l'état du watchdog à la mise sous tension du module

En cas d'erreur, déclenche une exception ou retourne `Y_AUTOSTART_INVALID`.

watchdog→get_countdown()

YWatchdog

watchdog→countdown()

watchdog.get_countdown()

Retourne le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à `delayedPulse()`.

`long get_countdown()`

Si aucune impulsion n'est programmée, retourne zéro.

Retourne :

un entier représentant le nombre de millisecondes restantes avant le déclenchement d'une impulsion préprogrammée par un appel à `delayedPulse()`

En cas d'erreur, déclenche une exception ou retourne `Y_COUNTDOWN_INVALID`.

watchdog→get_errorMessage()**YWatchdog****watchdog→errorMessage()****watchdog.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du watchdog.

watchdog→get_errorType()

YWatchdog

watchdog→errorType()

watchdog.get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du watchdog.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du watchdog.

watchdog→get_friendlyName()**YWatchdog****watchdog→friendlyName()****watchdog.get_friendlyName()**

Retourne un identifiant global du watchdog au format `NOM_MODULE.NOM_FONCTION`.

String `get_friendlyName()`

Le chaîne retournée utilise soit les noms logiques du module et du watchdog si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du watchdog (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le watchdog en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

watchdog→**get_functionDescriptor()**

YWatchdog

watchdog→**functionDescriptor()**

watchdog.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

watchdog→get_functionId()**YWatchdog****watchdog→functionId()****watchdog.get_functionId()**

Retourne l'identifiant matériel du watchdog, sans référence au module.

String **get_functionId()** ()

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le watchdog (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

watchdog→**get_hardwareId()**

YWatchdog

watchdog→**hardwareId()**

watchdog.get_hardwareId()

Retourne l'identifiant matériel unique du watchdog au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()** ()

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du watchdog (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le watchdog (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

watchdog→get_logicalName()**YWatchdog****watchdog→logicalName()****watchdog.get_logicalName()**

Retourne le nom logique du watchdog.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique du watchdog.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

watchdog→**get_maxTimeOnStateA()**

YWatchdog

watchdog→**maxTimeOnStateA()**

watchdog.get_maxTimeOnStateA()

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

long get_maxTimeOnStateA()

Zéro signifie qu'il n'y a pas de limitation

Retourne :

un entier représentant le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B

En cas d'erreur, déclenche une exception ou retourne Y_MAXTIMEONSTATEA_INVALID.

watchdog→get_maxTimeOnStateB()**YWatchdog****watchdog→maxTimeOnStateB()****watchdog.get_maxTimeOnStateB()**

Retourne le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
long get_maxTimeOnStateB( )
```

Zéro signifie qu'il n'y a pas de limitation

Retourne :

un entier représentant le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A

En cas d'erreur, déclenche une exception ou retourne Y_MAXTIMEONSTATEB_INVALID.

watchdog→**get_module()**

YWatchdog

watchdog→**module()**`watchdog.get_module()`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

`YModule` **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

watchdog→get_output()**YWatchdog****watchdog→output()**`watchdog.get_output()`

Retourne l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

int **get_output()**

Retourne :

soit Y_OUTPUT_OFF, soit Y_OUTPUT_ON, selon l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur

En cas d'erreur, déclenche une exception ou retourne Y_OUTPUT_INVALID.

watchdog→get_pulseTimer()

YWatchdog

watchdog→pulseTimer()

watchdog.get_pulseTimer()

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

long get_pulseTimer()

Si aucune impulsion n'est en cours, retourne zéro.

Retourne :

un entier représentant le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée

En cas d'erreur, déclenche une exception ou retourne Y_PULSETIMER_INVALID.

watchdog→get_running()**YWatchdog****watchdog→running()**`watchdog.get_running()`

Retourne l'état du watchdog.

`int get_running()`

Retourne :

soit `Y_RUNNING_OFF`, soit `Y_RUNNING_ON`, selon l'état du watchdog

En cas d'erreur, déclenche une exception ou retourne `Y_RUNNING_INVALID`.

watchdog→**get_state()**

YWatchdog

watchdog→**state()**`watchdog.get_state()`

Retourne l'état du watchdog (A pour la position de repos, B pour l'état actif).

```
int get_state( )
```

Retourne :

soit Y_STATE_A, soit Y_STATE_B, selon l'état du watchdog (A pour la position de repos, B pour l'état actif)

En cas d'erreur, déclenche une exception ou retourne Y_STATE_INVALID.

watchdog→get_stateAtPowerOn()**YWatchdog****watchdog→stateAtPowerOn()****watchdog.get_stateAtPowerOn()**

Retourne l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
int get_stateAtPowerOn( )
```

Retourne :

une valeur parmi Y_STATEATPOWERON_UNCHANGED, Y_STATEATPOWERON_A et Y_STATEATPOWERON_B représentant l'état du watchdog au démarrage du module (A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement)

En cas d'erreur, déclenche une exception ou retourne Y_STATEATPOWERON_INVALID.

watchdog→get_triggerDelay()

YWatchdog

watchdog→triggerDelay()

watchdog.get_triggerDelay()

Retourne le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchdog, en millisecondes.

long get_triggerDelay()

Retourne :

un entier représentant le délai d'attente avant qu'un reset ne soit automatiquement généré par le watchdog, en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y_TRIGGERDELAY_INVALID.

watchdog→get_triggerDuration()**YWatchdog****watchdog→triggerDuration()****watchdog.get_triggerDuration()**

Retourne la durée d'un reset généré par le watchdog, en millisecondes.

```
long get_triggerDuration( )
```

Retourne :

un entier représentant la durée d'un reset généré par le watchdog, en millisecondes

En cas d'erreur, déclenche une exception ou retourne Y_TRIGGERDURATION_INVALID.

watchdog→**get_userdata()**

YWatchdog

watchdog→**userData()****watchdog.get_userdata()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

watchdog→isOnline()`watchdog.isOnline()`**YWatchdog**

Vérifie si le module hébergeant le watchdog est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache du watchdog sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le watchdog est joignable, `false` sinon

watchdog → **load()** `watchdog.load()`**YWatchdog**

Met en cache les valeurs courantes du watchdog, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→**nextWatchdog()****YWatchdog****watchdog.nextWatchdog()**

Continue l'énumération des watchdog commencée à l'aide de `yFirstWatchdog()`.

`YWatchdog` **nextWatchdog()**

Retourne :

un pointeur sur un objet `YWatchdog` accessible en ligne, ou `null` lorsque l'énumération est terminée.

watchdog→**pulse()****watchdog.pulse()**

YWatchdog

Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

```
int pulse( int ms_duration)
```

Paramètres :

ms_duration durée de l'impulsion, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→registerValueCallback()**YWatchdog****watchdog.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

watchdog→**resetWatchdog()**

YWatchdog

watchdog.resetWatchdog()

Réinitialise le WatchDog.

```
int resetWatchdog( )
```

Quand le watchdog est en fonctionnement cette fonction doit être appelée à interval régulier, pour empêcher que le watdog ne se déclenche

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→set_autoStart()**YWatchdog****watchdog→setAutoStart()****watchdog.set_autoStart()**

Modifie l'état du watching au démarrage du module.

```
int set_autoStart( int newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

newval soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon l'état du watching au démarrage du module

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→**set_logicalName()**

YWatchdog

watchdog→**setLogicalName()**

watchdog.set_logicalName()

Modifie le nom logique du watchdog.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du watchdog.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→set_maxTimeOnStateA()**YWatchdog****watchdog→setMaxTimeOnStateA()****watchdog.set_maxTimeOnStateA()**

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état A avant de basculer automatiquement dans l'état B.

```
int set_maxTimeOnStateA( long newval)
```

Zéro signifie qu'il n'y a pas de limitation

Paramètres :

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→**set_maxTimeOnStateB()**

YWatchdog

watchdog→**setMaxTimeOnStateB()**

watchdog.set_maxTimeOnStateB()

Règle le temps maximal (en ms) pendant lequel le watchdog peut rester dans l'état B avant de basculer automatiquement dans l'état A.

```
int set_maxTimeOnStateB( long newval)
```

Zéro signifie qu'il n'y a pas de limitation

Paramètres :

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→set_output()**YWatchdog****watchdog→setOutput()**`watchdog.set_output ()`

Modifie l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur.

```
int set_output( int newval)
```

Paramètres :

newval soit Y_OUTPUT_OFF, soit Y_OUTPUT_ON, selon l'état de la sortie du watchdog, lorsqu'il est utilisé comme un simple interrupteur

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→**set_running()**

YWatchdog

watchdog→**setRunning()**`watchdog.set_running()`

Modifie manuellement l'état de fonctionnement du watchdog.

```
int set_running( int newval)
```

Paramètres :

newval soit Y_RUNNING_OFF, soit Y_RUNNING_ON, selon manuellement l'état de fonctionnement du watchdog

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→set_state()**YWatchdog****watchdog→setState()**`watchdog.set_state()`

Modifie l'état du watchdog (A pour la position de repos, B pour l'état actif).

```
int set_state( int newval)
```

Paramètres :

newval soit Y_STATE_A, soit Y_STATE_B, selon l'état du watchdog (A pour la position de repos, B pour l'état actif)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→set_stateAtPowerOn()

YWatchdog

watchdog→setStateAtPowerOn()

watchdog.set_stateAtPowerOn()

Pré-programme l'état du watchdog au démarrage du module(A pour la position de repos, B pour l'état actif, UNCHANGED pour aucun changement).

```
int set_stateAtPowerOn( int newval)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module sinon la modification n'aura aucun effet.

Paramètres :

newval une valeur parmi Y_STATEATPOWERON_UNCHANGED, Y_STATEATPOWERON_A et Y_STATEATPOWERON_B

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→set_triggerDelay()**YWatchdog****watchdog→setTriggerDelay()****watchdog.set_triggerDelay()**

Modifie le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes.

```
int set_triggerDelay( long newval)
```

Paramètres :

newval un entier représentant le délai d'attente avant qu'un reset ne soit généré par le watchdog, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→set_triggerDuration()

YWatchdog

watchdog→setTriggerDuration()

watchdog.set_triggerDuration()

Modifie la durée des resets générés par le watchdog, en millisecondes.

```
int set_triggerDuration( long newval)
```

Paramètres :

newval un entier représentant la durée des resets générés par le watchdog, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

watchdog→set_userdata()**YWatchdog****watchdog→setUserData()****watchdog.set_userdata()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

```
void set_userdata( Object data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

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');
c++	#include "yocto_wireless.h"
m	#import "yocto_wireless.h"
pas	uses yocto_wireless;
vb	yocto_wireless.vb
cs	yocto_wireless.cs
java	import com.yoctopuce.YoctoAPI.YWireless;
py	from yocto_wireless import *

Fonction globales
yFindWireless(func) Permet de retrouver une interface réseau sans fil d'après un identifiant donné.
yFirstWireless() Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.
Méthodes des objets YWireless
wireless→adhocNetwork(ssid, securityKey) Modifie la configuration de l'interface réseau sans fil pour créer un réseau sans fil sans point d'accès, en mode "ad-hoc".
wireless→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau sans fil au format TYPE (NAME) = SERIAL . FUNCTIONID.
wireless→get_advertisedValue() Retourne la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).
wireless→get_channel() Retourne le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.
wireless→get_detectedWlans() Retourne une liste d'objets objet YFileRecord qui décrivent les réseaux sans fils détectés.
wireless→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.
wireless→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.
wireless→get_friendlyName() Retourne un identifiant global de l'interface réseau sans fil au format NOM_MODULE . NOM_FONCTION.
wireless→get_functionDescriptor() Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.
wireless→get_functionId() Retourne l'identifiant matériel de l'interface réseau sans fil, sans référence au module.
wireless→get_hardwareId()

Retourne l'identifiant matériel unique de l'interface réseau sans fil au format `SERIAL.FUNCTIONID`.

wireless→get_linkQuality()

Retourne la qualité de la connection, exprimée en pourcents.

wireless→get_logicalName()

Retourne le nom logique de l'interface réseau sans fil.

wireless→get_message()

Retourne le dernier message de diagnostic de l'interface au réseau sans fil.

wireless→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

wireless→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

wireless→get_security()

Retourne l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné.

wireless→get_ssid()

Retourne le nom (SSID) du réseau sans-fil sélectionné.

wireless→get_userData()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

wireless→isOnline()

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

wireless→isOnline_async(callback, context)

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

wireless→joinNetwork(ssid, securityKey)

Modifie la configuration de l'interface réseau sans fil pour se connecter à un point d'accès sans fil existant (mode "infrastructure").

wireless→load(msValidity)

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

wireless→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

wireless→nextWireless()

Continue l'énumération des interfaces réseau sans fil commencée à l'aide de `yFirstWireless()`.

wireless→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

wireless→set_logicalName(newval)

Modifie le nom logique de l'interface réseau sans fil.

wireless→set_userData(data)

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

wireless→softAPNetwork(ssid, securityKey)

Modifie la configuration de l'interface réseau sans fil pour créer un pseudo point d'accès sans fil ("Soft AP").

wireless→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YWireless.FindWireless()**YWireless****yFindWireless()**`YWireless.FindWireless()`

Permet de retrouver une interface réseau sans fil d'après un identifiant donné.

`YWireless` **FindWireless**(`String func`)

L'identifiant peut être spécifié sous plusieurs formes:

- `NomLogiqueFonction`
- `NoSerieModule.IdentifiantFonction`
- `NoSerieModule.NomLogiqueFonction`
- `NomLogiqueModule.IdentifiantMatériel`
- `NomLogiqueModule.NomLogiqueFonction`

Cette fonction n'exige pas que l'interface réseau sans fil soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWireless.isOnline()` pour tester si l'interface réseau sans fil est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

func une chaîne de caractères qui référence l'interface réseau sans fil sans ambiguïté

Retourne :

un objet de classe `YWireless` qui permet ensuite de contrôler l'interface réseau sans fil.

YWireless.FirstWireless()**YWireless****yFirstWireless()**`ywireless.FirstWireless()`

Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.

`YWireless` **FirstWireless()**

Utiliser la fonction `YWireless.nextWireless()` pour itérer sur les autres interfaces réseau sans fil.

Retourne :

un pointeur sur un objet `YWireless`, correspondant à la première interface réseau sans fil accessible en ligne, ou `null` si il n'y a pas de interfaces réseau sans fil disponibles.

wireless→adhocNetwork()**YWireless****wireless.adhocNetwork()**

Modifie la configuration de l'interface réseau sans fil pour créer un réseau sans fil sans point d'accès, en mode "ad-hoc".

```
int adhocNetwork( String ssid, String securityKey)
```

Sur le YoctoHub-Wireless-g, il est recommandé d'utiliser de préférence la fonction `softAPNetwork()` qui crée un pseudo point d'accès, plus efficace et mieux supporté qu'un réseau ad-hoc.

Si une clef d'accès est configurée pour un réseau ad-hoc, le réseau sera protégé par une sécurité WEP40 (5 caractères ou 10 chiffres hexadécimaux) ou WEP128 (13 caractères ou 26 chiffres hexadécimaux). Pour réduire les risques d'intrusion, il est recommandé d'utiliser une clé WEP128 basée sur 26 chiffres hexadécimaux provenant d'une bonne source aléatoire.

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

ssid nom du réseau sans fil à créer
securityKey clé d'accès de réseau, sous forme de chaîne de caractères

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wireless→describe()`wireless.describe()`**YWireless**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'interface réseau sans fil au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

String `describe()`

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant l'interface réseau sans fil (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

wireless→**get_advertisedValue()**

YWireless

wireless→**advertisedValue()**

wireless.get_advertisedValue()

Retourne la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).

String **get_advertisedValue()**

Retourne :

une chaîne de caractères représentant la valeur courante de l'interface réseau sans fil (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne Y_ADVERTISEDVALUE_INVALID.

wireless→**get_channel()****YWireless****wireless**→**channel()****wireless.get_channel()**

Retourne le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.

int **get_channel()**

Retourne :

un entier représentant le numéro du canal 802.11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé

En cas d'erreur, déclenche une exception ou retourne `Y_CHANNEL_INVALID`.

wireless→get_detectedWlans()

YWireless

wireless→detectedWlans()

wireless.get_detectedWlans()

Retourne une liste d'objets objet YFileRecord qui décrivent les réseaux sans fils détectés.

`ArrayList<YWlanRecord> get_detectedWlans()`

La liste n'est pas mise à jour quand le module est déjà connecté à un accès sans fil (mode "infrastructure"). Pour forcer la détection des réseaux sans fil, il faut appeler `adhocNetwork()` pour se déconnecter du réseau actuel. L'appelant est responsable de la désallocation de la liste retournée.

Retourne :

une liste d'objets `YWlanRecord`, contenant le SSID, le canal, la qualité du signal, et l'algorithme de sécurité utilisé par le réseau sans-fil

En cas d'erreur, déclenche une exception ou retourne une liste vide.

wireless→**get_errorMessage()****YWireless****wireless**→**errorMessage()****wireless.get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

String **get_errorMessage()**

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau sans fil.

wireless→**get_errorType()**

YWireless

wireless→**errorType()****wireless.get_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'interface réseau sans fil.

```
int get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'interface réseau sans fil.

wireless→get_friendlyName()**YWireless****wireless→friendlyName()****wireless.get_friendlyName()**

Retourne un identifiant global de l'interface réseau sans fil au format `NOM_MODULE.NOM_FONCTION`.

String get_friendlyName()

Le chaîne retournée utilise soit les noms logiques du module et de l'interface réseau sans fil si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'interface réseau sans fil (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'interface réseau sans fil en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FRIENDLYNAME_INVALID`.

wireless→**get_functionDescriptor()**

YWireless

wireless→**functionDescriptor()**

wireless.get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

String **get_functionDescriptor()**

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

wireless→**get_functionId()****YWireless****wireless**→**functionId()****wireless.get_functionId()**

Retourne l'identifiant matériel de l'interface réseau sans fil, sans référence au module.

`String` **get_functionId()** ()

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant l'interface réseau sans fil (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_FUNCTIONID_INVALID`.

wireless→**get_hardwareId()**

YWireless

wireless→**hardwareId()**

wireless.get_hardwareId()

Retourne l'identifiant matériel unique de l'interface réseau sans fil au format `SERIAL.FUNCTIONID`.

String **get_hardwareId()**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'interface réseau sans fil (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant l'interface réseau sans fil (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

wireless→get_linkQuality()**YWireless****wireless→linkQuality()****wireless.get_linkQuality()**

Retourne la qualité de la connection, exprimée en pourcents.

int **get_linkQuality()** ()

Retourne :

un entier représentant la qualité de la connection, exprimée en pourcents

En cas d'erreur, déclenche une exception ou retourne Y_LINKQUALITY_INVALID.

wireless→**get_logicalName()**

YWireless

wireless→**logicalName()**

wireless.get_logicalName()

Retourne le nom logique de l'interface réseau sans fil.

String **get_logicalName()**

Retourne :

une chaîne de caractères représentant le nom logique de l'interface réseau sans fil.

En cas d'erreur, déclenche une exception ou retourne Y_LOGICALNAME_INVALID.

wireless→**get_message()****YWireless****wireless**→**message()**`wireless.get_message()`

Retourne le dernier message de diagnostic de l'interface au réseau sans fil.

String **get_message()**

Retourne :

une chaîne de caractères représentant le dernier message de diagnostic de l'interface au réseau sans fil

En cas d'erreur, déclenche une exception ou retourne Y_MESSAGE_INVALID.

wireless→**get_module()**

YWireless

wireless→**module()****wireless.get_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

YModule **get_module()**

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :

une instance de YModule

wireless→**get_security()****YWireless****wireless**→**security()****wireless.get_security()**

Retourne l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné.

```
int get_security( )
```

Retourne :

une valeur parmi Y_SECURITY_UNKNOWN, Y_SECURITY_OPEN, Y_SECURITY_WEP, Y_SECURITY_WPA et Y_SECURITY_WPA2 représentant l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné

En cas d'erreur, déclenche une exception ou retourne Y_SECURITY_INVALID.

wireless→**get_ssid()**

YWireless

wireless→**ssid()****wireless.get_ssid()**

Retourne le nom (SSID) du réseau sans-fil sélectionné.

String **get_ssid()**

Retourne :

une chaîne de caractères représentant le nom (SSID) du réseau sans-fil sélectionné

En cas d'erreur, déclenche une exception ou retourne Y_SSID_INVALID.

wireless→**get_userdata()****YWireless****wireless**→**userData()****wireless.userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userdata`.

Object **get_userdata()**

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

wireless→**isOnline()****wireless.isOnline()**

YWireless

Vérifie si le module hébergeant l'interface réseau sans fil est joignable, sans déclencher d'erreur.

boolean **isOnline**()

Si les valeurs des attributs en cache de l'interface réseau sans fil sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'interface réseau sans fil est joignable, `false` sinon

wireless→**joinNetwork()****wireless.joinNetwork()****YWireless**

Modifie la configuration de l'interface réseau sans fil pour se connecter à un point d'accès sans fil existant (mode "infrastructure").

```
int joinNetwork( String ssid, String securityKey)
```

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

ssid nom du réseau sans fil à utiliser

securityKey clé d'accès au réseau, sous forme de chaîne de caractères

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wireless→load()`wireless.load()`**YWireless**

Met en cache les valeurs courantes de l'interface réseau sans fil, avec une durée de validité spécifiée.

```
int load( long msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wireless→**nextWireless()****YWireless****wireless.nextWireless()**

Continue l'énumération des interfaces réseau sans fil commencée à l'aide de `yFirstWireless()`.

[YWireless](#) **nextWireless()****Retourne :**

un pointeur sur un objet `YWireless` accessible en ligne, ou `null` lorsque l'énumération est terminée.

wireless→registerValueCallback()**YWireless****wireless.registerValueCallback()**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
int registerValueCallback( UpdateCallback callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

wireless→**set_logicalName()**
wireless→**setLogicalName()**
wireless.set_logicalName()

YWireless

Modifie le nom logique de l'interface réseau sans fil.

```
int set_logicalName( String newval)
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'interface réseau sans fil.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

wireless→**set_userdata()**

YWireless

wireless→**setUserData()****wireless.set_userdata()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userdata`.

`void set_userdata(Object data)`

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

wireless→softAPNetwork()**YWireless****wireless.softAPNetwork()**

Modifie la configuration de l'interface réseau sans fil pour créer un pseudo point d'accès sans fil ("Soft AP").

```
int softAPNetwork( String ssid, String securityKey)
```

Cette fonction ne fonctionne que sur le YoctoHub-Wireless-g.

Si une clef d'accès est configurée pour un réseau SoftAP, le réseau sera protégé par une sécurité WEP40 (5 caractères ou 10 chiffres hexadécimaux) ou WEP128 (13 caractères ou 26 chiffres hexadécimaux). Pour réduire les risques d'intrusion, il est recommandé d'utiliser une clé WEP128 basée sur 26 chiffres hexadécimaux provenant d'une bonne source aléatoire.

N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

ssid nom du réseau sans fil à créer

securityKey clé d'accès de réseau, sous forme de chaîne de caractères

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

Index

A

Accelerometer 36
Accès 3
Activer 4
ad hoc Network, YWireless 1763
Alimentation 499
Altitude 78
AnButton 120
Android 3, 4

B

Blueprint 16
brakingForceMove, YMotor 876
Brute 361

C

calibrate, YLightSensor 746
calibrateFromPoints, YAccelerometer 40
calibrateFromPoints, YAltitude 82
calibrateFromPoints, YCarbonDioxide 162
calibrateFromPoints, YCompass 230
calibrateFromPoints, YCurrent 270
calibrateFromPoints, YGenericSensor 555
calibrateFromPoints, YGyro 604
calibrateFromPoints, YHumidity 680
calibrateFromPoints, YLightSensor 747
calibrateFromPoints, YMagnetometer 788
calibrateFromPoints, YPower 997
calibrateFromPoints, YPressure 1040
calibrateFromPoints, YPwmInput 1079
calibrateFromPoints, YQt 1188
calibrateFromPoints, YSensor 1326
calibrateFromPoints, YTemperature 1457
calibrateFromPoints, YTilt 1498
calibrateFromPoints, YVoc 1537
calibrateFromPoints, YVoltage 1576
callbackLogin, YNetwork 918
cancel3DCalibration, YRefFrame 1254
CarbonDioxide 158
checkFirmware, YModule 836
CheckLogicalName, YAPI 18
clear, YDisplayLayer 468
clearConsole, YDisplayLayer 469
ColorLed 197
Compass 226
Compatibilité 3
Configuration 1250
consoleOut, YDisplayLayer 470
Contrôle 6, 8, 499, 832, 970
copyLayerContent, YDisplay 424
Current 266

D

DataLogger 305
delayedPulse, YDigitalIO 380
delayedPulse, YRelay 1290
delayedPulse, YWatchdog 1719
describe, YAccelerometer 41
describe, YAltitude 83
describe, YAnButton 124
describe, YCarbonDioxide 163
describe, YColorLed 200
describe, YCompass 231
describe, YCurrent 271
describe, YDataLogger 309
describe, YDigitalIO 381
describe, YDisplay 425
describe, YDualPower 502
describe, YFiles 527
describe, YGenericSensor 556
describe, YGyro 605
describe, YHubPort 654
describe, YHumidity 681
describe, YLed 718
describe, YLightSensor 748
describe, YMagnetometer 789
describe, YModule 837
describe, YMotor 877
describe, YNetwork 919
describe, YOsControl 973
describe, YPower 998
describe, YPressure 1041
describe, YPwmInput 1080
describe, YPwmOutput 1127
describe, YPwmPowerSource 1164
describe, YQt 1189
describe, YRealTimeClock 1226
describe, YRefFrame 1255
describe, YRelay 1291
describe, YSensor 1327
describe, YSerialPort 1366
describe, YServo 1422
describe, YTemperature 1458
describe, YTilt 1499
describe, YVoc 1538
describe, YVoltage 1577
describe, YVSource 1614
describe, YWakeUpMonitor 1647
describe, YWakeUpSchedule 1682
describe, YWatchdog 1720
describe, YWireless 1764
DigitalIO 376
Display 420
DisplayLayer 467
Données 339, 349, 361

drawBar, YDisplayLayer 471
drawBitmap, YDisplayLayer 472
drawCircle, YDisplayLayer 473
drawDisc, YDisplayLayer 474
drawImage, YDisplayLayer 475
drawPixel, YDisplayLayer 476
drawRect, YDisplayLayer 477
drawText, YDisplayLayer 478
drivingForceMove, YMotor 878
dutyCycleMove, YPwmOutput 1128

E

EnableUSBHost, YAPI 19
Enregistrées 349, 361
Erreurs 13

F

fade, YDisplay 426
Files 524
FindAccelerometer, YAccelerometer 38
FindAltitude, YAltitude 80
FindAnButton, YAnButton 122
FindCarbonDioxide, YCarbonDioxide 160
FindColorLed, YColorLed 198
FindCompass, YCompass 228
FindCurrent, YCurrent 268
FindDataLogger, YDataLogger 307
FindDigitalIO, YDigitalIO 378
FindDisplay, YDisplay 422
FindDualPower, YDualPower 500
FindFiles, YFiles 525
FindGenericSensor, YGenericSensor 553
FindGyro, YGyro 602
FindHubPort, YHubPort 652
FindHumidity, YHumidity 678
FindLed, YLed 716
FindLightSensor, YLightSensor 744
FindMagnetometer, YMagnetometer 786
FindModule, YModule 834
FindMotor, YMotor 874
FindNetwork, YNetwork 916
FindOsControl, YOsControl 971
FindPower, YPower 995
FindPressure, YPressure 1038
FindPwmInput, YPwmInput 1077
FindPwmOutput, YPwmOutput 1125
FindPwmPowerSource, YPwmPowerSource 1162
FindQt, YQt 1186
FindRealTimeClock, YRealTimeClock 1224
FindRefFrame, YRefFrame 1252
FindRelay, YRelay 1288
FindSensor, YSensor 1324
FindSerialPort, YSerialPort 1364
FindServo, YServo 1420
FindTemperature, YTemperature 1455
FindTilt, YTilt 1496
FindVoc, YVoc 1535

FindVoltage, YVoltage 1574
FindVSource, YVSource 1612
FindWakeUpMonitor, YWakeUpMonitor 1645
FindWakeUpSchedule, YWakeUpSchedule 1680
FindWatchdog, YWatchdog 1717
FindWireless, YWireless 1761
FirstAccelerometer, YAccelerometer 39
FirstAltitude, YAltitude 81
FirstAnButton, YAnButton 123
FirstCarbonDioxide, YCarbonDioxide 161
FirstColorLed, YColorLed 199
FirstCompass, YCompass 229
FirstCurrent, YCurrent 269
FirstDataLogger, YDataLogger 308
FirstDigitalIO, YDigitalIO 379
FirstDisplay, YDisplay 423
FirstDualPower, YDualPower 501
FirstFiles, YFiles 526
FirstGenericSensor, YGenericSensor 554
FirstGyro, YGyro 603
FirstHubPort, YHubPort 653
FirstHumidity, YHumidity 679
FirstLed, YLed 717
FirstLightSensor, YLightSensor 745
FirstMagnetometer, YMagnetometer 787
FirstModule, YModule 835
FirstMotor, YMotor 875
FirstNetwork, YNetwork 917
FirstOsControl, YOsControl 972
FirstPower, YPower 996
FirstPressure, YPressure 1039
FirstPwmInput, YPwmInput 1078
FirstPwmOutput, YPwmOutput 1126
FirstPwmPowerSource, YPwmPowerSource 1163
FirstQt, YQt 1187
FirstRealTimeClock, YRealTimeClock 1225
FirstRefFrame, YRefFrame 1253
FirstRelay, YRelay 1289
FirstSensor, YSensor 1325
FirstSerialPort, YSerialPort 1365
FirstServo, YServo 1421
FirstTemperature, YTemperature 1456
FirstTilt, YTilt 1497
FirstVoc, YVoc 1536
FirstVoltage, YVoltage 1575
FirstVSource, YVSource 1613
FirstWakeUpMonitor, YWakeUpMonitor 1646
FirstWakeUpSchedule, YWakeUpSchedule 1681
FirstWatchdog, YWatchdog 1718
FirstWireless, YWireless 1762
Fonctions 17, 1322
forgetAllDataStreams, YDataLogger 310
format_fs, YFiles 528
Forme 339
FreeAPI, YAPI 20

G

GenericSensor 551
get_3DCalibrationHint, YRefFrame 1256
get_3DCalibrationLogMsg, YRefFrame 1257
get_3DCalibrationProgress, YRefFrame 1258
get_3DCalibrationStage, YRefFrame 1259
get_3DCalibrationStageProgress, YRefFrame 1260
get_adminPassword, YNetwork 920
get_advertisedValue, YAccelerometer 42
get_advertisedValue, YAltitude 84
get_advertisedValue, YAnButton 125
get_advertisedValue, YCarbonDioxide 164
get_advertisedValue, YColorLed 201
get_advertisedValue, YCompass 232
get_advertisedValue, YCurrent 272
get_advertisedValue, YDataLogger 311
get_advertisedValue, YDigitalIO 382
get_advertisedValue, YDisplay 427
get_advertisedValue, YDualPower 503
get_advertisedValue, YFiles 529
get_advertisedValue, YGenericSensor 557
get_advertisedValue, YGyro 606
get_advertisedValue, YHubPort 655
get_advertisedValue, YHumidity 682
get_advertisedValue, YLed 719
get_advertisedValue, YLightSensor 749
get_advertisedValue, YMagnetometer 790
get_advertisedValue, YMotor 879
get_advertisedValue, YNetwork 921
get_advertisedValue, YOsControl 974
get_advertisedValue, YPower 999
get_advertisedValue, YPressure 1042
get_advertisedValue, YPwmInput 1081
get_advertisedValue, YPwmOutput 1129
get_advertisedValue, YPwmPowerSource 1165
get_advertisedValue, YQt 1190
get_advertisedValue, YRealTimeClock 1227
get_advertisedValue, YRefFrame 1261
get_advertisedValue, YRelay 1292
get_advertisedValue, YSensor 1328
get_advertisedValue, YSerialPort 1368
get_advertisedValue, YServo 1423
get_advertisedValue, YTemperature 1459
get_advertisedValue, YTilt 1500
get_advertisedValue, YVoc 1539
get_advertisedValue, YVoltage 1578
get_advertisedValue, YVSource 1615
get_advertisedValue, YWakeUpMonitor 1648
get_advertisedValue, YWakeUpSchedule 1683
get_advertisedValue, YWatchdog 1721
get_advertisedValue, YWireless 1765
get_analogCalibration, YAnButton 126
get_autoStart, YDataLogger 312
get_autoStart, YWatchdog 1722
get_averageValue, YDataRun 339
get_averageValue, YDataStream 362
get_averageValue, YMeasure 826
get_baudRate, YHubPort 656
get_beacon, YModule 838
get_beaconDriven, YDataLogger 313
get_bearing, YRefFrame 1262
get_bitDirection, YDigitalIO 383
get_bitOpenDrain, YDigitalIO 384
get_bitPolarity, YDigitalIO 385
get_bitState, YDigitalIO 386
get_blinking, YLed 720
get_brakingForce, YMotor 880
get_brightness, YDisplay 428
get_calibratedValue, YAnButton 127
get_calibrationMax, YAnButton 128
get_calibrationMin, YAnButton 129
get_callbackCredentials, YNetwork 922
get_callbackEncoding, YNetwork 923
get_callbackMaxDelay, YNetwork 924
get_callbackMethod, YNetwork 925
get_callbackMinDelay, YNetwork 926
get_callbackUrl, YNetwork 927
get_channel, YWireless 1766
get_columnCount, YDataStream 363
get_columnNames, YDataStream 364
get_cosPhi, YPower 1000
get_countdown, YRelay 1293
get_countdown, YWatchdog 1723
get_CTS, YSerialPort 1367
get_currentRawValue, YAccelerometer 43
get_currentRawValue, YAltitude 85
get_currentRawValue, YCarbonDioxide 165
get_currentRawValue, YCompass 233
get_currentRawValue, YCurrent 273
get_currentRawValue, YGenericSensor 558
get_currentRawValue, YGyro 607
get_currentRawValue, YHumidity 683
get_currentRawValue, YLightSensor 750
get_currentRawValue, YMagnetometer 791
get_currentRawValue, YPower 1001
get_currentRawValue, YPressure 1043
get_currentRawValue, YPwmInput 1082
get_currentRawValue, YQt 1191
get_currentRawValue, YSensor 1329
get_currentRawValue, YTemperature 1460
get_currentRawValue, YTilt 1501
get_currentRawValue, YVoc 1540
get_currentRawValue, YVoltage 1579
get_currentRunIndex, YDataLogger 314
get_currentValue, YAccelerometer 44
get_currentValue, YAltitude 86
get_currentValue, YCarbonDioxide 166
get_currentValue, YCompass 234
get_currentValue, YCurrent 274
get_currentValue, YGenericSensor 559
get_currentValue, YGyro 608
get_currentValue, YHumidity 684
get_currentValue, YLightSensor 751
get_currentValue, YMagnetometer 792
get_currentValue, YPower 1002
get_currentValue, YPressure 1044

get_currentValue, YPwmInput 1083
get_currentValue, YQt 1192
get_currentValue, YSensor 1330
get_currentValue, YTemperature 1461
get_currentValue, YTilt 1502
get_currentValue, YVoc 1541
get_currentValue, YVoltage 1580
get_cutOffVoltage, YMotor 881
get_data, YDataStream 365
get_dataRows, YDataStream 366
get_dataSamplesIntervalMs, YDataStream 367
get_dataSets, YDataLogger 315
get_dataStreams, YDataLogger 316
get_dateTime, YRealTimeClock 1228
get_detectedWlans, YWireless 1767
get_discoverable, YNetwork 928
get_display, YDisplayLayer 479
get_displayHeight, YDisplay 429
get_displayHeight, YDisplayLayer 480
get_displayLayer, YDisplay 430
get_displayType, YDisplay 431
get_displayWidth, YDisplay 432
get_displayWidth, YDisplayLayer 481
get_drivingForce, YMotor 882
get_duration, YDataRun 340
get_duration, YDataStream 368
get_dutyCycle, YPwmInput 1084
get_dutyCycle, YPwmOutput 1130
get_dutyCycleAtPowerOn, YPwmOutput 1131
get_enabled, YDisplay 433
get_enabled, YHubPort 657
get_enabled, YPwmOutput 1132
get_enabled, YServo 1424
get_enabledAtPowerOn, YPwmOutput 1133
get_enabledAtPowerOn, YServo 1425
get_endTimeUTC, YDataSet 350
get_endTimeUTC, YMeasure 827
get_errCount, YSerialPort 1369
get_errorMessage, YAccelerometer 45
get_errorMessage, YAltitude 87
get_errorMessage, YAnButton 130
get_errorMessage, YCarbonDioxide 167
get_errorMessage, YColorLed 202
get_errorMessage, YCompass 235
get_errorMessage, YCurrent 275
get_errorMessage, YDataLogger 317
get_errorMessage, YDigitalIO 387
get_errorMessage, YDisplay 434
get_errorMessage, YDualPower 504
get_errorMessage, YFiles 530
get_errorMessage, YGenericSensor 560
get_errorMessage, YGyro 609
get_errorMessage, YHubPort 658
get_errorMessage, YHumidity 685
get_errorMessage, YLed 721
get_errorMessage, YLightSensor 752
get_errorMessage, YMagnetometer 793
get_errorMessage, YModule 839
get_errorMessage, YMotor 883

get_errorMessage, YNetwork 929
get_errorMessage, YOsControl 975
get_errorMessage, YPower 1003
get_errorMessage, YPressure 1045
get_errorMessage, YPwmInput 1085
get_errorMessage, YPwmOutput 1134
get_errorMessage, YPwmPowerSource 1166
get_errorMessage, YQt 1193
get_errorMessage, YRealTimeClock 1229
get_errorMessage, YRefFrame 1263
get_errorMessage, YRelay 1294
get_errorMessage, YSensor 1331
get_errorMessage, YSerialPort 1370
get_errorMessage, YServo 1426
get_errorMessage, YTemperature 1462
get_errorMessage, YTilt 1503
get_errorMessage, YVoc 1542
get_errorMessage, YVoltage 1581
get_errorMessage, YVSource 1616
get_errorMessage, YWakeUpMonitor 1649
get_errorMessage, YWakeUpSchedule 1684
get_errorMessage, YWatchdog 1724
get_errorMessage, YWireless 1768
get_errorType, YAccelerometer 46
get_errorType, YAltitude 88
get_errorType, YAnButton 131
get_errorType, YCarbonDioxide 168
get_errorType, YColorLed 203
get_errorType, YCompass 236
get_errorType, YCurrent 276
get_errorType, YDataLogger 318
get_errorType, YDigitalIO 388
get_errorType, YDisplay 435
get_errorType, YDualPower 505
get_errorType, YFiles 531
get_errorType, YGenericSensor 561
get_errorType, YGyro 610
get_errorType, YHubPort 659
get_errorType, YHumidity 686
get_errorType, YLed 722
get_errorType, YLightSensor 753
get_errorType, YMagnetometer 794
get_errorType, YModule 840
get_errorType, YMotor 884
get_errorType, YNetwork 930
get_errorType, YOsControl 976
get_errorType, YPower 1004
get_errorType, YPressure 1046
get_errorType, YPwmInput 1086
get_errorType, YPwmOutput 1135
get_errorType, YPwmPowerSource 1167
get_errorType, YQt 1194
get_errorType, YRealTimeClock 1230
get_errorType, YRefFrame 1264
get_errorType, YRelay 1295
get_errorType, YSensor 1332
get_errorType, YSerialPort 1371
get_errorType, YServo 1427
get_errorType, YTemperature 1463

get_errorType, YTilt 1504
get_errorType, YVoc 1543
get_errorType, YVoltage 1582
get_errorType, YVSource 1617
get_errorType, YWakeUpMonitor 1650
get_errorType, YWakeUpSchedule 1685
get_errorType, YWatchdog 1725
get_errorType, YWireless 1769
get_extPowerFailure, YVSource 1618
get_extVoltage, YDualPower 506
get_failSafeTimeout, YMotor 885
get_failure, YVSource 1619
get_filesCount, YFiles 532
get_firmwareRelease, YModule 841
get_freeSpace, YFiles 533
get_frequency, YMotor 886
get_frequency, YPwmInput 1087
get_frequency, YPwmOutput 1136
get_friendlyName, YAccelerometer 47
get_friendlyName, YAltitude 89
get_friendlyName, YAnButton 132
get_friendlyName, YCarbonDioxide 169
get_friendlyName, YColorLed 204
get_friendlyName, YCompass 237
get_friendlyName, YCurrent 277
get_friendlyName, YDataLogger 319
get_friendlyName, YDigitalIO 389
get_friendlyName, YDisplay 436
get_friendlyName, YDualPower 507
get_friendlyName, YFiles 534
get_friendlyName, YGenericSensor 562
get_friendlyName, YGyro 611
get_friendlyName, YHubPort 660
get_friendlyName, YHumidity 687
get_friendlyName, YLed 723
get_friendlyName, YLightSensor 754
get_friendlyName, YMagnetometer 795
get_friendlyName, YMotor 887
get_friendlyName, YNetwork 931
get_friendlyName, YOsControl 977
get_friendlyName, YPower 1005
get_friendlyName, YPressure 1047
get_friendlyName, YPwmInput 1088
get_friendlyName, YPwmOutput 1137
get_friendlyName, YPwmPowerSource 1168
get_friendlyName, YQt 1195
get_friendlyName, YRealTimeClock 1231
get_friendlyName, YRefFrame 1265
get_friendlyName, YRelay 1296
get_friendlyName, YSensor 1333
get_friendlyName, YSerialPort 1372
get_friendlyName, YServo 1428
get_friendlyName, YTemperature 1464
get_friendlyName, YTilt 1505
get_friendlyName, YVoc 1544
get_friendlyName, YVoltage 1583
get_friendlyName, YVSource 1620
get_friendlyName, YWakeUpMonitor 1651
get_friendlyName, YWakeUpSchedule 1686

get_friendlyName, YWatchdog 1726
get_friendlyName, YWireless 1770
get_functionDescriptor, YAccelerometer 48
get_functionDescriptor, YAltitude 90
get_functionDescriptor, YAnButton 133
get_functionDescriptor, YCarbonDioxide 170
get_functionDescriptor, YColorLed 205
get_functionDescriptor, YCompass 238
get_functionDescriptor, YCurrent 278
get_functionDescriptor, YDataLogger 320
get_functionDescriptor, YDigitalIO 390
get_functionDescriptor, YDisplay 437
get_functionDescriptor, YDualPower 508
get_functionDescriptor, YFiles 535
get_functionDescriptor, YGenericSensor 563
get_functionDescriptor, YGyro 612
get_functionDescriptor, YHubPort 661
get_functionDescriptor, YHumidity 688
get_functionDescriptor, YLed 724
get_functionDescriptor, YLightSensor 755
get_functionDescriptor, YMagnetometer 796
get_functionDescriptor, YMotor 888
get_functionDescriptor, YNetwork 932
get_functionDescriptor, YOsControl 978
get_functionDescriptor, YPower 1006
get_functionDescriptor, YPressure 1048
get_functionDescriptor, YPwmInput 1089
get_functionDescriptor, YPwmOutput 1138
get_functionDescriptor, YPwmPowerSource 1169
get_functionDescriptor, YQt 1196
get_functionDescriptor, YRealTimeClock 1232
get_functionDescriptor, YRefFrame 1266
get_functionDescriptor, YRelay 1297
get_functionDescriptor, YSensor 1334
get_functionDescriptor, YSerialPort 1373
get_functionDescriptor, YServo 1429
get_functionDescriptor, YTemperature 1465
get_functionDescriptor, YTilt 1506
get_functionDescriptor, YVoc 1545
get_functionDescriptor, YVoltage 1584
get_functionDescriptor, YVSource 1621
get_functionDescriptor, YWakeUpMonitor 1652
get_functionDescriptor, YWakeUpSchedule 1687
get_functionDescriptor, YWatchdog 1727
get_functionDescriptor, YWireless 1771
get_functionId, YAccelerometer 49
get_functionId, YAltitude 91
get_functionId, YAnButton 134
get_functionId, YCarbonDioxide 171
get_functionId, YColorLed 206
get_functionId, YCompass 239
get_functionId, YCurrent 279
get_functionId, YDataLogger 321
get_functionId, YDataSet 351
get_functionId, YDigitalIO 391
get_functionId, YDisplay 438
get_functionId, YDualPower 509
get_functionId, YFiles 536
get_functionId, YGenericSensor 564

get_functionId, YGyro 613
get_functionId, YHubPort 662
get_functionId, YHumidity 689
get_functionId, YLed 725
get_functionId, YLightSensor 756
get_functionId, YMagnetometer 797
get_functionId, YMotor 889
get_functionId, YNetwork 933
get_functionId, YOsControl 979
get_functionId, YPower 1007
get_functionId, YPressure 1049
get_functionId, YPwmInput 1090
get_functionId, YPwmOutput 1139
get_functionId, YPwmPowerSource 1170
get_functionId, YQt 1197
get_functionId, YRealTimeClock 1233
get_functionId, YRefFrame 1267
get_functionId, YRelay 1298
get_functionId, YSensor 1335
get_functionId, YSerialPort 1374
get_functionId, YServo 1430
get_functionId, YTemperature 1466
get_functionId, YTilt 1507
get_functionId, YVoc 1546
get_functionId, YVoltage 1585
get_functionId, YVSource 1622
get_functionId, YWakeUpMonitor 1653
get_functionId, YWakeUpSchedule 1688
get_functionId, YWatchdog 1728
get_functionId, YWireless 1772
get_hardwareId, YAccelerometer 50
get_hardwareId, YAltitude 92
get_hardwareId, YAnButton 135
get_hardwareId, YCarbonDioxide 172
get_hardwareId, YColorLed 207
get_hardwareId, YCompass 240
get_hardwareId, YCurrent 280
get_hardwareId, YDataLogger 322
get_hardwareId, YDataSet 352
get_hardwareId, YDigitalIO 392
get_hardwareId, YDisplay 439
get_hardwareId, YDualPower 510
get_hardwareId, YFiles 537
get_hardwareId, YGenericSensor 565
get_hardwareId, YGyro 614
get_hardwareId, YHubPort 663
get_hardwareId, YHumidity 690
get_hardwareId, YLed 726
get_hardwareId, YLightSensor 757
get_hardwareId, YMagnetometer 798
get_hardwareId, YModule 842
get_hardwareId, YMotor 890
get_hardwareId, YNetwork 934
get_hardwareId, YOsControl 980
get_hardwareId, YPower 1008
get_hardwareId, YPressure 1050
get_hardwareId, YPwmInput 1091
get_hardwareId, YPwmOutput 1140
get_hardwareId, YPwmPowerSource 1171

get_hardwareId, YQt 1198
get_hardwareId, YRealTimeClock 1234
get_hardwareId, YRefFrame 1268
get_hardwareId, YRelay 1299
get_hardwareId, YSensor 1336
get_hardwareId, YSerialPort 1375
get_hardwareId, YServo 1431
get_hardwareId, YTemperature 1467
get_hardwareId, YTilt 1508
get_hardwareId, YVoc 1547
get_hardwareId, YVoltage 1586
get_hardwareId, YVSource 1623
get_hardwareId, YWakeUpMonitor 1654
get_hardwareId, YWakeUpSchedule 1689
get_hardwareId, YWatchdog 1729
get_hardwareId, YWireless 1773
get_heading, YGyro 615
get_highestValue, YAccelerometer 51
get_highestValue, YAltitude 93
get_highestValue, YCarbonDioxide 173
get_highestValue, YCompass 241
get_highestValue, YCurrent 281
get_highestValue, YGenericSensor 566
get_highestValue, YGyro 616
get_highestValue, YHumidity 691
get_highestValue, YLightSensor 758
get_highestValue, YMagnetometer 799
get_highestValue, YPower 1009
get_highestValue, YPressure 1051
get_highestValue, YPwmInput 1092
get_highestValue, YQt 1199
get_highestValue, YSensor 1337
get_highestValue, YTemperature 1468
get_highestValue, YTilt 1509
get_highestValue, YVoc 1548
get_highestValue, YVoltage 1587
get_hours, YWakeUpSchedule 1690
get_hslColor, YColorLed 208
get_ipAddress, YNetwork 935
get_isPressed, YAnButton 136
get_lastLogs, YModule 843
get_lastMsg, YSerialPort 1376
get_lastTimePressed, YAnButton 137
get_lastTimeReleased, YAnButton 138
get_layerCount, YDisplay 440
get_layerHeight, YDisplay 441
get_layerHeight, YDisplayLayer 482
get_layerWidth, YDisplay 442
get_layerWidth, YDisplayLayer 483
get_linkQuality, YWireless 1774
get_list, YFiles 538
get_logFrequency, YAccelerometer 52
get_logFrequency, YAltitude 94
get_logFrequency, YCarbonDioxide 174
get_logFrequency, YCompass 242
get_logFrequency, YCurrent 282
get_logFrequency, YGenericSensor 567
get_logFrequency, YGyro 617
get_logFrequency, YHumidity 692

get_logFrequency, YLightSensor 759
get_logFrequency, YMagnetometer 800
get_logFrequency, YPower 1010
get_logFrequency, YPressure 1052
get_logFrequency, YPwmInput 1093
get_logFrequency, YQt 1200
get_logFrequency, YSensor 1338
get_logFrequency, YTemperature 1469
get_logFrequency, YTilt 1510
get_logFrequency, YVoc 1549
get_logFrequency, YVoltage 1588
get_logicalName, YAccelerometer 53
get_logicalName, YAltitude 95
get_logicalName, YAnButton 139
get_logicalName, YCarbonDioxide 175
get_logicalName, YColorLed 209
get_logicalName, YCompass 243
get_logicalName, YCurrent 283
get_logicalName, YDataLogger 323
get_logicalName, YDigitalIO 393
get_logicalName, YDisplay 443
get_logicalName, YDualPower 511
get_logicalName, YFiles 539
get_logicalName, YGenericSensor 568
get_logicalName, YGyro 618
get_logicalName, YHubPort 664
get_logicalName, YHumidity 693
get_logicalName, YLed 727
get_logicalName, YLightSensor 760
get_logicalName, YMagnetometer 801
get_logicalName, YModule 844
get_logicalName, YMotor 891
get_logicalName, YNetwork 936
get_logicalName, YOsControl 981
get_logicalName, YPower 1011
get_logicalName, YPressure 1053
get_logicalName, YPwmInput 1094
get_logicalName, YPwmOutput 1141
get_logicalName, YPwmPowerSource 1172
get_logicalName, YQt 1201
get_logicalName, YRealTimeClock 1235
get_logicalName, YRefFrame 1269
get_logicalName, YRelay 1300
get_logicalName, YSensor 1339
get_logicalName, YSerialPort 1377
get_logicalName, YServo 1432
get_logicalName, YTemperature 1470
get_logicalName, YTilt 1511
get_logicalName, YVoc 1550
get_logicalName, YVoltage 1589
get_logicalName, YVSource 1624
get_logicalName, YWakeUpMonitor 1655
get_logicalName, YWakeUpSchedule 1691
get_logicalName, YWatchdog 1730
get_logicalName, YWireless 1775
get_lowestValue, YAccelerometer 54
get_lowestValue, YAltitude 96
get_lowestValue, YCarbonDioxide 176
get_lowestValue, YCompass 244

get_lowestValue, YCurrent 284
get_lowestValue, YGenericSensor 569
get_lowestValue, YGyro 619
get_lowestValue, YHumidity 694
get_lowestValue, YLightSensor 761
get_lowestValue, YMagnetometer 802
get_lowestValue, YPower 1012
get_lowestValue, YPressure 1054
get_lowestValue, YPwmInput 1095
get_lowestValue, YQt 1202
get_lowestValue, YSensor 1340
get_lowestValue, YTemperature 1471
get_lowestValue, YTilt 1512
get_lowestValue, YVoc 1551
get_lowestValue, YVoltage 1590
get_luminosity, YLed 728
get_luminosity, YModule 845
get_macAddress, YNetwork 937
get_magneticHeading, YCompass 245
get_maxTimeOnStateA, YRelay 1301
get_maxTimeOnStateA, YWatchdog 1731
get_maxTimeOnStateB, YRelay 1302
get_maxTimeOnStateB, YWatchdog 1732
get_maxValue, YDataRun 341
get_maxValue, YDataStream 369
get_maxValue, YMeasure 828
get_measureNames, YDataRun 342
get_measures, YDataSet 353
get_measureType, YLightSensor 762
get_message, YWireless 1776
get_meter, YPower 1013
get_meterTimer, YPower 1014
get_minutes, YWakeUpSchedule 1692
get_minutesA, YWakeUpSchedule 1693
get_minutesB, YWakeUpSchedule 1694
get_minValue, YDataRun 343
get_minValue, YDataStream 370
get_minValue, YMeasure 829
get_module, YAccelerometer 55
get_module, YAltitude 97
get_module, YAnButton 140
get_module, YCarbonDioxide 177
get_module, YColorLed 210
get_module, YCompass 246
get_module, YCurrent 285
get_module, YDataLogger 324
get_module, YDigitalIO 394
get_module, YDisplay 444
get_module, YDualPower 512
get_module, YFiles 540
get_module, YGenericSensor 570
get_module, YGyro 620
get_module, YHubPort 665
get_module, YHumidity 695
get_module, YLed 729
get_module, YLightSensor 763
get_module, YMagnetometer 803
get_module, YMotor 892
get_module, YNetwork 938

get_module, YOsControl	982	get_productId, YModule	847
get_module, YPower	1015	get_productName, YModule	848
get_module, YPressure	1055	get_productRelease, YModule	849
get_module, YPwmInput	1096	get_progress, YDataSet	355
get_module, YPwmOutput	1142	get_protocol, YSerialPort	1380
get_module, YPwmPowerSource	1173	get_pulseCounter, YAnButton	141
get_module, YQt	1203	get_pulseCounter, YPwmInput	1098
get_module, YRealTimeClock	1236	get_pulseDuration, YPwmInput	1099
get_module, YRefFrame	1270	get_pulseDuration, YPwmOutput	1144
get_module, YRelay	1303	get_pulseTimer, YAnButton	142
get_module, YSensor	1341	get_pulseTimer, YPwmInput	1100
get_module, YSerialPort	1378	get_pulseTimer, YRelay	1305
get_module, YServo	1433	get_pulseTimer, YWatchdog	1735
get_module, YTemperature	1472	get_pwmReportMode, YPwmInput	1101
get_module, YTilt	1513	get_qnh, YAltitude	98
get_module, YVoc	1552	get_quaternionW, YGyro	622
get_module, YVoltage	1591	get_quaternionX, YGyro	623
get_module, YVSource	1625	get_quaternionY, YGyro	624
get_module, YWakeUpMonitor	1656	get_quaternionZ, YGyro	625
get_module, YWakeUpSchedule	1695	get_range, YServo	1437
get_module, YWatchdog	1733	get_rawValue, YAnButton	143
get_module, YWireless	1777	get_readiness, YNetwork	941
get_monthDays, YWakeUpSchedule	1696	get_rebootCountdown, YModule	850
get_months, YWakeUpSchedule	1697	get_recordedData, YAccelerometer	56
get_motorStatus, YMotor	893	get_recordedData, YAltitude	99
get_mountOrientation, YRefFrame	1271	get_recordedData, YCarbonDioxide	178
get_mountPosition, YRefFrame	1272	get_recordedData, YCompass	247
get_msgCount, YSerialPort	1379	get_recordedData, YCurrent	286
get_neutral, YServo	1434	get_recordedData, YGenericSensor	571
get_nextOccurence, YWakeUpSchedule	1698	get_recordedData, YGyro	626
get_nextWakeUp, YWakeUpMonitor	1657	get_recordedData, YHumidity	696
get_orientation, YDisplay	445	get_recordedData, YLightSensor	764
get_output, YRelay	1304	get_recordedData, YMagnetometer	804
get_output, YWatchdog	1734	get_recordedData, YPower	1016
get_outputVoltage, YDigitalIO	395	get_recordedData, YPressure	1056
get_overCurrent, YVSource	1626	get_recordedData, YPwmInput	1102
get_overCurrentLimit, YMotor	894	get_recordedData, YQt	1204
get_overHeat, YVSource	1627	get_recordedData, YSensor	1342
get_overLoad, YVSource	1628	get_recordedData, YTemperature	1473
get_period, YPwmInput	1097	get_recordedData, YTilt	1514
get_period, YPwmOutput	1143	get_recordedData, YVoc	1553
get_persistentSettings, YModule	846	get_recordedData, YVoltage	1592
get_pitch, YGyro	621	get_recording, YDataLogger	325
get_poeCurrent, YNetwork	939	get_regulationFailure, YVSource	1629
get_portDirection, YDigitalIO	396	get_reportFrequency, YAccelerometer	57
get_portOpenDrain, YDigitalIO	397	get_reportFrequency, YAltitude	100
get_portPolarity, YDigitalIO	398	get_reportFrequency, YCarbonDioxide	179
get_portSize, YDigitalIO	399	get_reportFrequency, YCompass	248
get_portState, YDigitalIO	400	get_reportFrequency, YCurrent	287
get_portState, YHubPort	666	get_reportFrequency, YGenericSensor	572
get_position, YServo	1435	get_reportFrequency, YGyro	627
get_positionAtPowerOn, YServo	1436	get_reportFrequency, YHumidity	697
get_power, YLed	730	get_reportFrequency, YLightSensor	765
get_powerControl, YDualPower	513	get_reportFrequency, YMagnetometer	805
get_powerDuration, YWakeUpMonitor	1658	get_reportFrequency, YPower	1017
get_powerMode, YPwmPowerSource	1174	get_reportFrequency, YPressure	1057
get_powerState, YDualPower	514	get_reportFrequency, YPwmInput	1103
get_preview, YDataSet	354	get_reportFrequency, YQt	1205
get_primaryDNS, YNetwork	940	get_reportFrequency, YSensor	1343

get_reportFrequency, YTemperature 1474
get_reportFrequency, YTilt 1515
get_reportFrequency, YVoc 1554
get_reportFrequency, YVoltage 1593
get_resolution, YAccelerometer 58
get_resolution, YAltitude 101
get_resolution, YCarbonDioxide 180
get_resolution, YCompass 249
get_resolution, YCurrent 288
get_resolution, YGenericSensor 573
get_resolution, YGyro 628
get_resolution, YHumidity 698
get_resolution, YLightSensor 766
get_resolution, YMagnetometer 806
get_resolution, YPower 1018
get_resolution, YPressure 1058
get_resolution, YPwmInput 1104
get_resolution, YQt 1206
get_resolution, YSensor 1344
get_resolution, YTemperature 1475
get_resolution, YTilt 1516
get_resolution, YVoc 1555
get_resolution, YVoltage 1594
get_rgbColor, YColorLed 211
get_rgbColorAtPowerOn, YColorLed 212
get_roll, YGyro 629
get_router, YNetwork 942
get_rowCount, YDataStream 371
get_runIndex, YDataStream 372
get_running, YWatchdog 1736
get_rxCount, YSerialPort 1381
get_secondaryDNS, YNetwork 943
get_security, YWireless 1778
get_sensitivity, YAnButton 144
get_sensorType, YTemperature 1476
get_serialMode, YSerialPort 1382
get_serialNumber, YModule 851
get_shutdownCountdown, YOsControl 983
get_signalBias, YGenericSensor 574
get_signalRange, YGenericSensor 575
get_signalUnit, YGenericSensor 576
get_signalValue, YGenericSensor 577
get_sleepCountdown, YWakeUpMonitor 1659
get_ssid, YWireless 1779
get_starterTime, YMotor 895
get_startTime, YDataStream 373
get_startTimeUTC, YDataRun 344
get_startTimeUTC, YDataSet 356
get_startTimeUTC, YDataStream 374
get_startTimeUTC, YMeasure 830
get_startupSeq, YDisplay 446
get_state, YRelay 1306
get_state, YWatchdog 1737
get_stateAtPowerOn, YRelay 1307
get_stateAtPowerOn, YWatchdog 1738
get_subnetMask, YNetwork 944
get_summary, YDataSet 357
get_timeSet, YRealTimeClock 1237
get_timeUTC, YDataLogger 326

get_triggerDelay, YWatchdog 1739
get_triggerDuration, YWatchdog 1740
get_txCount, YSerialPort 1383
get_unit, YAccelerometer 59
get_unit, YAltitude 102
get_unit, YCarbonDioxide 181
get_unit, YCompass 250
get_unit, YCurrent 289
get_unit, YDataSet 358
get_unit, YGenericSensor 578
get_unit, YGyro 630
get_unit, YHumidity 699
get_unit, YLightSensor 767
get_unit, YMagnetometer 807
get_unit, YPower 1019
get_unit, YPressure 1059
get_unit, YPwmInput 1105
get_unit, YQt 1207
get_unit, YSensor 1345
get_unit, YTemperature 1477
get_unit, YTilt 1517
get_unit, YVoc 1556
get_unit, YVoltage 1595
get_unit, YVSource 1630
get_unixTime, YRealTimeClock 1238
get_upTime, YModule 852
get_usbCurrent, YModule 853
get_userData, YAccelerometer 60
get_userData, YAltitude 103
get_userData, YAnButton 145
get_userData, YCarbonDioxide 182
get_userData, YColorLed 213
get_userData, YCompass 251
get_userData, YCurrent 290
get_userData, YDataLogger 327
get_userData, YDigitalIO 401
get_userData, YDisplay 447
get_userData, YDualPower 515
get_userData, YFiles 541
get_userData, YGenericSensor 579
get_userData, YGyro 631
get_userData, YHubPort 667
get_userData, YHumidity 700
get_userData, YLed 731
get_userData, YLightSensor 768
get_userData, YMagnetometer 808
get_userData, YModule 854
get_userData, YMotor 896
get_userData, YNetwork 945
get_userData, YOsControl 984
get_userData, YPower 1020
get_userData, YPressure 1060
get_userData, YPwmInput 1106
get_userData, YPwmOutput 1145
get_userData, YPwmPowerSource 1175
get_userData, YQt 1208
get_userData, YRealTimeClock 1239
get_userData, YRefFrame 1273
get_userData, YRelay 1308

get_userdata, YSensor 1346
get_userdata, YSerialPort 1384
get_userdata, YServo 1438
get_userdata, YTemperature 1478
get_userdata, YTilt 1518
get_userdata, YVoc 1557
get_userdata, YVoltage 1596
get_userdata, YVSource 1631
get_userdata, YWakeUpMonitor 1660
get_userdata, YWakeUpSchedule 1699
get_userdata, YWatchdog 1741
get_userdata, YWireless 1780
get_userPassword, YNetwork 946
get_userVar, YModule 855
get_utcOffset, YRealTimeClock 1240
get_valueCount, YDataRun 345
get_valueInterval, YDataRun 346
get_valueRange, YGenericSensor 580
get_voltage, YVSource 1632
get_wakeUpReason, YWakeUpMonitor 1661
get_wakeUpState, YWakeUpMonitor 1662
get_weekDays, YWakeUpSchedule 1700
get_wwwWatchdogDelay, YNetwork 947
get_xValue, YAccelerometer 61
get_xValue, YGyro 632
get_xValue, YMagnetometer 809
get_yValue, YAccelerometer 62
get_yValue, YGyro 633
get_yValue, YMagnetometer 810
get_zValue, YAccelerometer 63
get_zValue, YGyro 634
get_zValue, YMagnetometer 811
GetAPIVersion, YAPI 21
GetTickCount, YAPI 22
Gyro 600

H

HandleEvents, YAPI 23
hide, YDisplayLayer 484
Horloge 1223
hslMove, YColorLed 214
Hub 3
Humidity 676

I

InitAPI, YAPI 24
Interface 36, 78, 120, 158, 197, 226, 266, 305,
376, 420, 467, 499, 524, 551, 600, 651, 676,
715, 742, 784, 832, 872, 913, 993, 1036, 1075,
1123, 1161, 1184, 1223, 1286, 1322, 1361,
1418, 1453, 1494, 1533, 1572, 1611, 1643,
1678, 1715, 1760
Introduction 1
isOnline, YAccelerometer 64
isOnline, YAltitude 104
isOnline, YAnButton 146
isOnline, YCarbonDioxide 183
isOnline, YColorLed 215

isOnline, YCompass 252
isOnline, YCurrent 291
isOnline, YDataLogger 328
isOnline, YDigitalIO 402
isOnline, YDisplay 448
isOnline, YDualPower 516
isOnline, YFiles 542
isOnline, YGenericSensor 581
isOnline, YGyro 635
isOnline, YHubPort 668
isOnline, YHumidity 701
isOnline, YLed 732
isOnline, YLightSensor 769
isOnline, YMagnetometer 812
isOnline, YModule 856
isOnline, YMotor 897
isOnline, YNetwork 948
isOnline, YOsControl 985
isOnline, YPower 1021
isOnline, YPressure 1061
isOnline, YPwmInput 1107
isOnline, YPwmOutput 1146
isOnline, YPwmPowerSource 1176
isOnline, YQt 1209
isOnline, YRealTimeClock 1241
isOnline, YRefFrame 1274
isOnline, YRelay 1309
isOnline, YSensor 1347
isOnline, YSerialPort 1385
isOnline, YServo 1439
isOnline, YTemperature 1479
isOnline, YTilt 1519
isOnline, YVoc 1558
isOnline, YVoltage 1597
isOnline, YVSource 1633
isOnline, YWakeUpMonitor 1663
isOnline, YWakeUpSchedule 1701
isOnline, YWatchdog 1742
isOnline, YWireless 1781

J

joinNetwork, YWireless 1782

K

keepALive, YMotor 898

L

LightSensor 742
lineTo, YDisplayLayer 485
load, YAccelerometer 65
load, YAltitude 105
load, YAnButton 147
load, YCarbonDioxide 184
load, YColorLed 216
load, YCompass 253
load, YCurrent 292
load, YDataLogger 329

- load, YDigitalIO 403
- load, YDisplay 449
- load, YDualPower 517
- load, YFiles 543
- load, YGenericSensor 582
- load, YGyro 636
- load, YHubPort 669
- load, YHumidity 702
- load, YLed 733
- load, YLightSensor 770
- load, YMagnetometer 813
- load, YModule 857
- load, YMotor 899
- load, YNetwork 949
- load, YOsControl 986
- load, YPower 1022
- load, YPressure 1062
- load, YPwmInput 1108
- load, YPwmOutput 1147
- load, YPwmPowerSource 1177
- load, YQt 1210
- load, YRealTimeClock 1242
- load, YRefFrame 1275
- load, YRelay 1310
- load, YSensor 1348
- load, YSerialPort 1386
- load, YServo 1440
- load, YTemperature 1480
- load, YTilt 1520
- load, YVoc 1559
- load, YVoltage 1598
- load, YVSource 1634
- load, YWakeUpMonitor 1664
- load, YWakeUpSchedule 1702
- load, YWatchdog 1743
- load, YWireless 1783
- loadCalibrationPoints, YAccelerometer 66
- loadCalibrationPoints, YAltitude 106
- loadCalibrationPoints, YCarbonDioxide 185
- loadCalibrationPoints, YCompass 254
- loadCalibrationPoints, YCurrent 293
- loadCalibrationPoints, YGenericSensor 583
- loadCalibrationPoints, YGyro 637
- loadCalibrationPoints, YHumidity 703
- loadCalibrationPoints, YLightSensor 771
- loadCalibrationPoints, YMagnetometer 814
- loadCalibrationPoints, YPower 1023
- loadCalibrationPoints, YPressure 1063
- loadCalibrationPoints, YPwmInput 1109
- loadCalibrationPoints, YQt 1211
- loadCalibrationPoints, YSensor 1349
- loadCalibrationPoints, YTemperature 1481
- loadCalibrationPoints, YTilt 1521
- loadCalibrationPoints, YVoc 1560
- loadCalibrationPoints, YVoltage 1599
- loadMore, YDataSet 359

M

- Magnetometer 784
- Mesurée 826
- Mise 339
- modbusReadBits, YSerialPort 1387
- modbusReadInputBits, YSerialPort 1388
- modbusReadInputRegisters, YSerialPort 1389
- modbusReadRegisters, YSerialPort 1390
- modbusWriteAndReadRegisters, YSerialPort 1391
- modbusWriteBit, YSerialPort 1392
- modbusWriteBits, YSerialPort 1393
- modbusWriteRegister, YSerialPort 1394
- modbusWriteRegisters, YSerialPort 1395
- Module 8, 832
- more3DCalibration, YRefFrame 1276
- Motor 872
- move, YServo 1441
- moveTo, YDisplayLayer 486

N

- Natif 3
- Network 913
- newSequence, YDisplay 450
- nextAccelerometer, YAccelerometer 67
- nextAltitude, YAltitude 107
- nextAnButton, YAnButton 148
- nextCarbonDioxide, YCarbonDioxide 186
- nextColorLed, YColorLed 217
- nextCompass, YCompass 255
- nextCurrent, YCurrent 294
- nextDataLogger, YDataLogger 330
- nextDigitalIO, YDigitalIO 404
- nextDisplay, YDisplay 451
- nextDualPower, YDualPower 518
- nextFiles, YFiles 544
- nextGenericSensor, YGenericSensor 584
- nextGyro, YGyro 638
- nextHubPort, YHubPort 670
- nextHumidity, YHumidity 704
- nextLed, YLed 734
- nextLightSensor, YLightSensor 772
- nextMagnetometer, YMagnetometer 815
- nextModule, YModule 858
- nextMotor, YMotor 900
- nextNetwork, YNetwork 950
- nextOsControl, YOsControl 987
- nextPower, YPower 1024
- nextPressure, YPressure 1064
- nextPwmInput, YPwmInput 1110
- nextPwmOutput, YPwmOutput 1148
- nextPwmPowerSource, YPwmPowerSource 1178
- nextQt, YQt 1212
- nextRealTimeClock, YRealTimeClock 1243
- nextRefFrame, YRefFrame 1277
- nextRelay, YRelay 1311

nextSensor, YSensor 1350
nextSerialPort, YSerialPort 1396
nextServo, YServo 1442
nextTemperature, YTemperature 1482
nextTilt, YTilt 1522
nextVoc, YVoc 1561
nextVoltage, YVoltage 1600
nextVSource, YVSource 1635
nextWakeUpMonitor, YWakeUpMonitor 1665
nextWakeUpSchedule, YWakeUpSchedule 1703
nextWatchdog, YWatchdog 1744
nextWireless, YWireless 1784

O

Objets 467

P

pauseSequence, YDisplay 452
ping, YNetwork 951
playSequence, YDisplay 453
Port 4, 651
Power 993
Préparation 3
PreregisterHub, YAPI 25
Pressure 1036
pulse, YDigitalIO 405
pulse, YRelay 1312
pulse, YVSource 1636
pulse, YWatchdog 1745
pulseDurationMove, YPwmOutput 1149
PwmInput 1075
PwmPowerSource 1161

Q

Quaternion 1184
queryLine, YSerialPort 1397
queryMODBUS, YSerialPort 1398

R

read_seek, YSerialPort 1403
readHex, YSerialPort 1399
readLine, YSerialPort 1400
readMessages, YSerialPort 1401
readStr, YSerialPort 1402
Real 1223
reboot, YModule 859
Reference 16
Référentiel 1250
registerAnglesCallback, YGyro 639
RegisterDeviceArrivalCallback, YAPI 26
RegisterDeviceRemovalCallback, YAPI 27
RegisterHub, YAPI 28
RegisterHubDiscoveryCallback, YAPI 29
registerLogCallback, YModule 860
RegisterLogFunction, YAPI 30
registerQuaternionCallback, YGyro 640
registerTimedReportCallback, YAccelerometer

68
registerTimedReportCallback, YAltitude 108
registerTimedReportCallback, YCarbonDioxide 187
registerTimedReportCallback, YCompass 256
registerTimedReportCallback, YCurrent 295
registerTimedReportCallback, YGenericSensor 585
registerTimedReportCallback, YGyro 641
registerTimedReportCallback, YHumidity 705
registerTimedReportCallback, YLightSensor 773
registerTimedReportCallback, YMagnetometer 816
registerTimedReportCallback, YPower 1025
registerTimedReportCallback, YPressure 1065
registerTimedReportCallback, YPwmInput 1111
registerTimedReportCallback, YQt 1213
registerTimedReportCallback, YSensor 1351
registerTimedReportCallback, YTemperature 1483
registerTimedReportCallback, YTilt 1523
registerTimedReportCallback, YVoc 1562
registerTimedReportCallback, YVoltage 1601
registerValueCallback, YAccelerometer 69
registerValueCallback, YAltitude 109
registerValueCallback, YAnButton 149
registerValueCallback, YCarbonDioxide 188
registerValueCallback, YColorLed 218
registerValueCallback, YCompass 257
registerValueCallback, YCurrent 296
registerValueCallback, YDataLogger 331
registerValueCallback, YDigitalIO 406
registerValueCallback, YDisplay 454
registerValueCallback, YDualPower 519
registerValueCallback, YFiles 545
registerValueCallback, YGenericSensor 586
registerValueCallback, YGyro 642
registerValueCallback, YHubPort 671
registerValueCallback, YHumidity 706
registerValueCallback, YLed 735
registerValueCallback, YLightSensor 774
registerValueCallback, YMagnetometer 817
registerValueCallback, YMotor 901
registerValueCallback, YNetwork 952
registerValueCallback, YOsControl 988
registerValueCallback, YPower 1026
registerValueCallback, YPressure 1066
registerValueCallback, YPwmInput 1112
registerValueCallback, YPwmOutput 1150
registerValueCallback, YPwmPowerSource 1179
registerValueCallback, YQt 1214
registerValueCallback, YRealTimeClock 1244
registerValueCallback, YRefFrame 1278
registerValueCallback, YRelay 1313
registerValueCallback, YSensor 1352
registerValueCallback, YSerialPort 1404
registerValueCallback, YServo 1443
registerValueCallback, YTemperature 1484
registerValueCallback, YTilt 1524

- registerValueCallback, YVoc 1563
- registerValueCallback, YVoltage 1602
- registerValueCallback, YVSource 1637
- registerValueCallback, YWakeUpMonitor 1666
- registerValueCallback, YWakeUpSchedule 1704
- registerValueCallback, YWatchdog 1746
- registerValueCallback, YWireless 1785
- Relay 1286
- remove, YFiles 546
- reset, YDisplayLayer 487
- reset, YPower 1027
- reset, YSerialPort 1405
- resetAll, YDisplay 455
- resetCounter, YAnButton 150
- resetCounter, YPwmInput 1113
- resetSleepCountDown, YWakeUpMonitor 1667
- resetStatus, YMotor 902
- resetWatchdog, YWatchdog 1747
- revertFromFlash, YModule 861
- rgbMove, YColorLed 219

S

- save3DCalibration, YRefFrame 1279
- saveSequence, YDisplay 456
- saveToFlash, YModule 862
- selectColorPen, YDisplayLayer 488
- selectEraser, YDisplayLayer 489
- selectFont, YDisplayLayer 490
- selectGrayPen, YDisplayLayer 491
- Senseur 1322
- Séquence 339, 349, 361
- SerialPort 1361
- Servo 1418
- set_adminPassword, YNetwork 953
- set_allSettings, YModule 863
- set_analogCalibration, YAnButton 151
- set_autoStart, YDataLogger 332
- set_autoStart, YWatchdog 1748
- set_beacon, YModule 864
- set_beaconDriven, YDataLogger 333
- set_bearing, YRefFrame 1280
- set_bitDirection, YDigitalIO 407
- set_bitOpenDrain, YDigitalIO 408
- set_bitPolarity, YDigitalIO 409
- set_bitState, YDigitalIO 410
- set_blinking, YLed 736
- set_brakingForce, YMotor 903
- set_brightness, YDisplay 457
- set_calibrationMax, YAnButton 152
- set_calibrationMin, YAnButton 153
- set_callbackCredentials, YNetwork 954
- set_callbackEncoding, YNetwork 955
- set_callbackMaxDelay, YNetwork 956
- set_callbackMethod, YNetwork 957
- set_callbackMinDelay, YNetwork 958
- set_callbackUrl, YNetwork 959
- set_currentValue, YAltitude 110
- set_cutOffVoltage, YMotor 904
- set_discoverable, YNetwork 960
- set_drivingForce, YMotor 905
- set_dutyCycle, YPwmOutput 1151
- set_dutyCycleAtPowerOn, YPwmOutput 1152
- set_enabled, YDisplay 458
- set_enabled, YHubPort 672
- set_enabled, YPwmOutput 1153
- set_enabled, YServo 1444
- set_enabledAtPowerOn, YPwmOutput 1154
- set_enabledAtPowerOn, YServo 1445
- set_failSafeTimeout, YMotor 906
- set_frequency, YMotor 907
- set_frequency, YPwmOutput 1155
- set_highestValue, YAccelerometer 70
- set_highestValue, YAltitude 111
- set_highestValue, YCarbonDioxide 189
- set_highestValue, YCompass 258
- set_highestValue, YCurrent 297
- set_highestValue, YGenericSensor 587
- set_highestValue, YGyro 643
- set_highestValue, YHumidity 707
- set_highestValue, YLightSensor 775
- set_highestValue, YMagnetometer 818
- set_highestValue, YPower 1028
- set_highestValue, YPressure 1067
- set_highestValue, YPwmInput 1114
- set_highestValue, YQt 1215
- set_highestValue, YSensor 1353
- set_highestValue, YTemperature 1485
- set_highestValue, YTilt 1525
- set_highestValue, YVoc 1564
- set_highestValue, YVoltage 1603
- set_hours, YWakeUpSchedule 1705
- set_hslColor, YColorLed 220
- set_logFrequency, YAccelerometer 71
- set_logFrequency, YAltitude 112
- set_logFrequency, YCarbonDioxide 190
- set_logFrequency, YCompass 259
- set_logFrequency, YCurrent 298
- set_logFrequency, YGenericSensor 588
- set_logFrequency, YGyro 644
- set_logFrequency, YHumidity 708
- set_logFrequency, YLightSensor 776
- set_logFrequency, YMagnetometer 819
- set_logFrequency, YPower 1029
- set_logFrequency, YPressure 1068
- set_logFrequency, YPwmInput 1115
- set_logFrequency, YQt 1216
- set_logFrequency, YSensor 1354
- set_logFrequency, YTemperature 1486
- set_logFrequency, YTilt 1526
- set_logFrequency, YVoc 1565
- set_logFrequency, YVoltage 1604
- set_logicalName, YAccelerometer 72
- set_logicalName, YAltitude 113
- set_logicalName, YAnButton 154
- set_logicalName, YCarbonDioxide 191
- set_logicalName, YColorLed 221
- set_logicalName, YCompass 260
- set_logicalName, YCurrent 299

set_logicalName, YDataLogger	334	set_maxTimeOnStateA, YWatchdog	1750
set_logicalName, YDigitalIO	411	set_maxTimeOnStateB, YRelay	1316
set_logicalName, YDisplay	459	set_maxTimeOnStateB, YWatchdog	1751
set_logicalName, YDualPower	520	set_measureType, YLightSensor	779
set_logicalName, YFiles	547	set_minutes, YWakeUpSchedule	1707
set_logicalName, YGenericSensor	589	set_minutesA, YWakeUpSchedule	1708
set_logicalName, YGyro	645	set_minutesB, YWakeUpSchedule	1709
set_logicalName, YHubPort	673	set_monthDays, YWakeUpSchedule	1710
set_logicalName, YHumidity	709	set_months, YWakeUpSchedule	1711
set_logicalName, YLed	737	set_mountPosition, YRefFrame	1282
set_logicalName, YLightSensor	777	set_neutral, YServo	1447
set_logicalName, YMagnetometer	820	set_nextWakeUp, YWakeUpMonitor	1669
set_logicalName, YModule	865	set_orientation, YDisplay	460
set_logicalName, YMotor	908	set_output, YRelay	1317
set_logicalName, YNetwork	961	set_output, YWatchdog	1752
set_logicalName, YOsControl	989	set_outputVoltage, YDigitalIO	412
set_logicalName, YPower	1030	set_overCurrentLimit, YMotor	909
set_logicalName, YPressure	1069	set_period, YPwmOutput	1157
set_logicalName, YPwmInput	1116	set_portDirection, YDigitalIO	413
set_logicalName, YPwmOutput	1156	set_portOpenDrain, YDigitalIO	414
set_logicalName, YPwmPowerSource	1180	set_portPolarity, YDigitalIO	415
set_logicalName, YQt	1217	set_portState, YDigitalIO	416
set_logicalName, YRealTimeClock	1245	set_position, YServo	1448
set_logicalName, YRefFrame	1281	set_positionAtPowerOn, YServo	1449
set_logicalName, YRelay	1314	set_power, YLed	739
set_logicalName, YSensor	1355	set_powerControl, YDualPower	521
set_logicalName, YSerialPort	1407	set_powerDuration, YWakeUpMonitor	1670
set_logicalName, YServo	1446	set_powerMode, YPwmPowerSource	1181
set_logicalName, YTemperature	1487	set_primaryDNS, YNetwork	962
set_logicalName, YTilt	1527	set_protocol, YSerialPort	1408
set_logicalName, YVoc	1566	set_pulseDuration, YPwmOutput	1158
set_logicalName, YVoltage	1605	set_pwmReportMode, YPwmInput	1118
set_logicalName, YVSource	1638	set_qnh, YAltitude	115
set_logicalName, YWakeUpMonitor	1668	set_range, YServo	1450
set_logicalName, YWakeUpSchedule	1706	set_recording, YDataLogger	335
set_logicalName, YWatchdog	1749	set_reportFrequency, YAccelerometer	74
set_logicalName, YWireless	1786	set_reportFrequency, YAltitude	116
set_lowestValue, YAccelerometer	73	set_reportFrequency, YCarbonDioxide	193
set_lowestValue, YAltitude	114	set_reportFrequency, YCompass	262
set_lowestValue, YCarbonDioxide	192	set_reportFrequency, YCurrent	301
set_lowestValue, YCompass	261	set_reportFrequency, YGenericSensor	591
set_lowestValue, YCurrent	300	set_reportFrequency, YGyro	647
set_lowestValue, YGenericSensor	590	set_reportFrequency, YHumidity	711
set_lowestValue, YGyro	646	set_reportFrequency, YLightSensor	780
set_lowestValue, YHumidity	710	set_reportFrequency, YMagnetometer	822
set_lowestValue, YLightSensor	778	set_reportFrequency, YPower	1032
set_lowestValue, YMagnetometer	821	set_reportFrequency, YPressure	1071
set_lowestValue, YPower	1031	set_reportFrequency, YPwmInput	1119
set_lowestValue, YPressure	1070	set_reportFrequency, YQt	1219
set_lowestValue, YPwmInput	1117	set_reportFrequency, YSensor	1357
set_lowestValue, YQt	1218	set_reportFrequency, YTemperature	1489
set_lowestValue, YSensor	1356	set_reportFrequency, YTilt	1529
set_lowestValue, YTemperature	1488	set_reportFrequency, YVoc	1568
set_lowestValue, YTilt	1528	set_reportFrequency, YVoltage	1607
set_lowestValue, YVoc	1567	set_resolution, YAccelerometer	75
set_lowestValue, YVoltage	1606	set_resolution, YAltitude	117
set_luminosity, YLed	738	set_resolution, YCarbonDioxide	194
set_luminosity, YModule	866	set_resolution, YCompass	263
set_maxTimeOnStateA, YRelay	1315	set_resolution, YCurrent	302

set_resolution, YGenericSensor 592
set_resolution, YGyro 648
set_resolution, YHumidity 712
set_resolution, YLightSensor 781
set_resolution, YMagnetometer 823
set_resolution, YPower 1033
set_resolution, YPressure 1072
set_resolution, YPwmInput 1120
set_resolution, YQt 1220
set_resolution, YSensor 1358
set_resolution, YTemperature 1490
set_resolution, YTilt 1530
set_resolution, YVoc 1569
set_resolution, YVoltage 1608
set_rgbColor, YColorLed 222
set_rgbColorAtPowerOn, YColorLed 223
set_RTS, YSerialPort 1406
set_running, YWatchdog 1753
set_secondaryDNS, YNetwork 963
set_sensitivity, YAnButton 155
set_sensorType, YTemperature 1491
set_serialMode, YSerialPort 1409
set_signalBias, YGenericSensor 593
set_signalRange, YGenericSensor 594
set_sleepCountdown, YWakeUpMonitor 1671
set_starterTime, YMotor 910
set_startupSeq, YDisplay 461
set_state, YRelay 1318
set_state, YWatchdog 1754
set_stateAtPowerOn, YRelay 1319
set_stateAtPowerOn, YWatchdog 1755
set_timeUTC, YDataLogger 336
set_triggerDelay, YWatchdog 1756
set_triggerDuration, YWatchdog 1757
set_unit, YGenericSensor 595
set_unixTime, YRealTimeClock 1246
set_userData, YAccelerometer 76
set_userData, YAltitude 118
set_userData, YAnButton 156
set_userData, YCarbonDioxide 195
set_userData, YColorLed 224
set_userData, YCompass 264
set_userData, YCurrent 303
set_userData, YDataLogger 337
set_userData, YDigitalIO 417
set_userData, YDisplay 462
set_userData, YDualPower 522
set_userData, YFiles 548
set_userData, YGenericSensor 596
set_userData, YGyro 649
set_userData, YHubPort 674
set_userData, YHumidity 713
set_userData, YLed 740
set_userData, YLightSensor 782
set_userData, YMagnetometer 824
set_userData, YModule 867
set_userData, YMotor 911
set_userData, YNetwork 964
set_userData, YOsControl 990

set_userData, YPower 1034
set_userData, YPressure 1073
set_userData, YPwmInput 1121
set_userData, YPwmOutput 1159
set_userData, YPwmPowerSource 1182
set_userData, YQt 1221
set_userData, YRealTimeClock 1247
set_userData, YRefFrame 1283
set_userData, YRelay 1320
set_userData, YSensor 1359
set_userData, YSerialPort 1410
set_userData, YServo 1451
set_userData, YTemperature 1492
set_userData, YTilt 1531
set_userData, YVoc 1570
set_userData, YVoltage 1609
set_userData, YVSource 1639
set_userData, YWakeUpMonitor 1672
set_userData, YWakeUpSchedule 1712
set_userData, YWatchdog 1758
set_userData, YWireless 1787
set_userPassword, YNetwork 965
set_userVar, YModule 868
set_utcOffset, YRealTimeClock 1248
set_valueInterval, YDataRun 347
set_valueRange, YGenericSensor 597
set_voltage, YVSource 1640
set_weekDays, YWakeUpSchedule 1713
set_wwwWatchdogDelay, YNetwork 966
setAntialiasingMode, YDisplayLayer 492
setConsoleBackground, YDisplayLayer 493
setConsoleMargins, YDisplayLayer 494
setConsoleWordWrap, YDisplayLayer 495
setLayerPosition, YDisplayLayer 496
shutdown, YOsControl 991
Sleep, YAPI 31
sleep, YWakeUpMonitor 1673
sleepFor, YWakeUpMonitor 1674
sleepUntil, YWakeUpMonitor 1675
softAPNetwork, YWireless 1788
Source 1611
start3DCalibration, YRefFrame 1284
stopSequence, YDisplay 463
swapLayerContent, YDisplay 464

T

Temperature 1453
Temps 1223
Tension 1611
Tilt 1494
toggle_bitState, YDigitalIO 418
triggerFirmwareUpdate, YModule 869
TriggerHubDiscovery, YAPI 32
Type 1322

U

unhide, YDisplayLayer 497
UnregisterHub, YAPI 33

UpdateDeviceList, YAPI 34
updateFirmware, YModule 870
upload, YDisplay 465
upload, YFiles 549
useDHCP, YNetwork 967
useStaticIP, YNetwork 968

V

Valeur 826
Virtual 3
Voltage 1572
voltageMove, YVSource 1641

W

wakeUp, YWakeUpMonitor 1676
WakeUpMonitor 1643
WakeUpSchedule 1678
Watchdog 1715
Wireless 1760
writeArray, YSerialPort 1411
writeBin, YSerialPort 1412
writeHex, YSerialPort 1413
writeLine, YSerialPort 1414
writeMODBUS, YSerialPort 1415
writeStr, YSerialPort 1416

Y

YAccelerometer 38-76
YAltitude 80-118
YAnButton 122-156
YAPI 18-34
YCarbonDioxide 160-195
yCheckLogicalName 18
YColorLed 198-224
YCompass 228-264
YCurrent 268-303
YDataLogger 307-337
YDataRun 339-347
YDataSet 350-359
YDataStream 362-374
YDigitalIO 378-418
YDisplay 422-465
YDisplayLayer 468-497
YDualPower 500-522
yEnableUSBHost 19
YFiles 525-549
yFindAccelerometer 38
yFindAltitude 80
yFindAnButton 122
yFindCarbonDioxide 160
yFindColorLed 198
yFindCompass 228
yFindCurrent 268
yFindDataLogger 307
yFindDigitalIO 378
yFindDisplay 422
yFindDualPower 500

yFindFiles 525
yFindGenericSensor 553
yFindGyro 602
yFindHubPort 652
yFindHumidity 678
yFindLed 716
yFindLightSensor 744
yFindMagnetometer 786
yFindModule 834
yFindMotor 874
yFindNetwork 916
yFindOsControl 971
yFindPower 995
yFindPressure 1038
yFindPwmInput 1077
yFindPwmOutput 1125
yFindPwmPowerSource 1162
yFindQt 1186
yFindRealTimeClock 1224
yFindRefFrame 1252
yFindRelay 1288
yFindSensor 1324
yFindSerialPort 1364
yFindServo 1420
yFindTemperature 1455
yFindTilt 1496
yFindVoc 1535
yFindVoltage 1574
yFindVSource 1612
yFindWakeUpMonitor 1645
yFindWakeUpSchedule 1680
yFindWatchdog 1717
yFindWireless 1761
yFirstAccelerometer 39
yFirstAltitude 81
yFirstAnButton 123
yFirstCarbonDioxide 161
yFirstColorLed 199
yFirstCompass 229
yFirstCurrent 269
yFirstDataLogger 308
yFirstDigitalIO 379
yFirstDisplay 423
yFirstDualPower 501
yFirstFiles 526
yFirstGenericSensor 554
yFirstGyro 603
yFirstHubPort 653
yFirstHumidity 679
yFirstLed 717
yFirstLightSensor 745
yFirstMagnetometer 787
yFirstModule 835
yFirstMotor 875
yFirstNetwork 917
yFirstOsControl 972
yFirstPower 996
yFirstPressure 1039
yFirstPwmInput 1078

- yFirstPwmOutput 1126
- yFirstPwmPowerSource 1163
- yFirstQt 1187
- yFirstRealTimeClock 1225
- yFirstRefFrame 1253
- yFirstRelay 1289
- yFirstSensor 1325
- yFirstSerialPort 1365
- yFirstServo 1421
- yFirstTemperature 1456
- yFirstTilt 1497
- yFirstVoc 1536
- yFirstVoltage 1575
- yFirstVSource 1613
- yFirstWakeUpMonitor 1646
- yFirstWakeUpSchedule 1681
- yFirstWatchdog 1718
- yFirstWireless 1762
- yFreeAPI 20
- YGenericSensor 553-598
- yGetAPIVersion 21
- yGetTickCount 22
- YGyro 602-649
- yHandleEvents 23
- YHubPort 652-674
- YHumidity 678-713
- yInitAPI 24
- YLed 716-740
- YLightSensor 744-782
- YMagnetometer 786-824
- YMeasure 826-830
- YModule 834-870
- YMotor 874-911
- YNetwork 916-968
- Yocto-Demo 3
- Yocto-hub 651

- YOsControl 971-991
- YPower 995-1034
- yPreregisterHub 25
- YPressure 1038-1073
- YPwmInput 1077-1121
- YPwmOutput 1125-1159
- YPwmPowerSource 1162-1182
- YQt 1186-1221
- YRealTimeClock 1224-1248
- YRefFrame 1252-1284
- yRegisterDeviceArrivalCallback 26
- yRegisterDeviceRemovalCallback 27
- yRegisterHub 28
- yRegisterHubDiscoveryCallback 29
- yRegisterLogFunction 30
- YRelay 1288-1320
- YSensor 1324-1359
- YSerialPort 1364-1416
- YServo 1420-1451
- ySleep 31
- YTemperature 1455-1492
- YTilt 1496-1531
- yTriggerHubDiscovery 32
- yUnregisterHub 33
- yUpdateDeviceList 34
- YVoc 1535-1570
- YVoltage 1574-1609
- YVSource 1612-1641
- YWakeUpMonitor 1645-1676
- YWakeUpSchedule 1680-1713
- YWatchdog 1717-1758
- YWireless 1761-1788

Z

- zeroAdjust, YGenericSensor 598