Référence de l'API C++

Table des matières

1. Introduction

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

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

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

2. Utilisation du Yocto–Demo en C++

Le C++ n'est pas le langage le plus simple à maîtriser. Pourtant, si on prend soin à se limiter aux fonctionnalités essentielles, c'est un langage tout à fait utilisable pour des petits programmes vite faits, et qui a l'avantage d'être très portable d'un système d'exploitation à l'autre. Sous Windows, tous les exemples et les modèles de projet sont testés avec Microsoft Visual Studio 2010 Express, disponible gratuitement sur le site de Microsoft [1]. Sous Mac OS X, tous les exemples et les modèles de projet sont testés avec XCode 4, disponible sur l'App Store. Par ailleurs, aussi bien sous Mac OS X que sous Linux, vous pouvez compiler les exemples en ligne de commande avec GCC en utilisant le `GNUmakefile` fourni. De même, sous Windows, un `Makefile` pour permet de compiler les exemples en ligne de commande, et en pleine connaissance des arguments de compilation et link.

Les librairies Yoctopuce[2] pour C++ vous sont fournies au format source dans leur intégralité. Une partie de la librairie de bas-niveau est écrite en C pur sucre, mais vous n'aurez à priori pas besoin d'interagir directement avec elle: tout a été fait pour que l'interaction soit le plus simple possible depuis le C++. La librairie vous est fournie bien entendu aussi sous forme binaire, de sorte à pouvoir la linker directement si vous le préférez.

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

Afin des les garder simples, tous les exemples fournis dans cette documentation sont des applications consoles. Il va de soit que que les fonctionnement des librairies est strictement identiques si vous les intégrez dans une application dotée d'une interface graphique. Vous trouverez dans la dernière section de ce chapitre toutes les informations nécessaires à la création d'un projet à neuf linké avec les librairies Yoctopuce.

2.1. 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 C++ qui utilise la fonction Led.

```
#include "yocto_api.h"
#include "yocto_led.h"

[...]
String  errmsg;
YLed *led;

// On récupère l'objet représentant le module (ici connecté en local sur USB)
yRegisterHub("usb", errmsg);
led = yFindLed("YCTOPOC1-123456.led");
```

[1] http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express
[2] www.yoctopuce.com/FR/libraries.php

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.h et yocto_led.h

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

yRegisterHub

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

yFindLed

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

```
YLed *led = yFindLed("YCTOPOC1-123456.led");
YLed *led = yFindLed("YCTOPOC1-123456.MaFonction");
YLed *led = yFindLed("MonModule.led");
YLed *led = yFindLed("MonModule.MaFonction");
YLed *led = yFindLed("MaFonction");
```

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

isOnline

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

set_power

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

Un exemple réel

Lancez votre environnement C++ et ouvrez le projet exemple correspondant, fourni dans le répertoire **Examples/Doc-GettingStarted-Yocto-Demo** de la librairie Yoctopuce. Si vous préférez travailler avec votre éditeur de texte préféré, ouvrez le fichier `main.cpp`, vous taperez simplement `make` dans le répertoire de l'exemple pour le compiler.

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.

```
#include "yocto_api.h"
#include "yocto_led.h"
#include <iostream>
#include <stdlib.h>

using namespace std;

static void usage(void)
{
    cout << "usage: demo <serial_number>  [ on | off ]" << endl;
```

```
    cout << "       demo <logical_name> [ on | off ]" << endl;
    cout << "       demo any [ on | off ]                (use any discovered device)"
<< endl;
    exit(1);
}

int main(int argc, const char * argv[])
{
    string errmsg;
    string target;
    YLed   *led;
    string on_off;

    if(argc < 3) {
        usage();
    }
    target    = (string) argv[1];
    on_off = (string) argv[2];

    // Setup the API to use local USB devices
    if(yRegisterHub("usb", errmsg) != YAPI_SUCCESS) {
        cerr << "RegisterHub error: " << errmsg << endl;
        return 1;
    }

    if(target == "any"){
        led =  yFirstLed();
    }else{
        led =  yFindLed(target + ".led");
    }
    if (led && led->isOnline()) {
        led->set_power(on_off == "on" ? Y_POWER_ON : Y_POWER_OFF);
    } else {
        cout << "Module not connected (check identification and USB cable)" << endl;
    }

    return 0;
}
```

2.2. Contrôle de la partie module

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

```
#include <iostream>
#include <stdlib.h>

#include "yocto_api.h"

using namespace std;

static void usage(const char *exe)
{
    cout << "usage: " << exe << " <serial or logical name> [ON/OFF]" << endl;
    exit(1);
}


int main(int argc, const char * argv[])
{
    string      errmsg;

    // Setup the API to use local USB devices
    if(yRegisterHub("usb", errmsg) != YAPI_SUCCESS) {
        cerr << "RegisterHub error: " << errmsg << endl;
        return 1;
    }

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

    YModule *module = yFindModule(argv[1]);  // use serial or logical name

    if (module->isOnline()) {
        if (argc > 2) {
            if (string(argv[2]) == "ON")
                module->set_beacon(Y_BEACON_ON);
            else
```

```
                module->set_beacon(Y_BEACON_OFF);
        }
        cout << "serial:       " << module->get_serialNumber() << endl;
        cout << "logical name: " << module->get_logicalName() << endl;
        cout << "luminosity:   " << module->get_luminosity() << endl;
        cout << "beacon:       ";
        if (module->get_beacon()==Y_BEACON_ON)
           cout << "ON" << endl;
        else
           cout << "OFF" << endl;
        cout << "upTime:       " << module->get_upTime()/1000 << " sec" << endl;
        cout << "USB current:  " << module->get_usbCurrent() << " mA" << endl;
    } else {
        cout << argv[1] << " not connected (check identification and USB cable)"
             << endl;
    }
    return 0;
}
```

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

Modifications des réglages du module

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

```
#include <iostream>
#include <stdlib.h>

#include "yocto_api.h"

using namespace std;

static void usage(const char *exe)
{
    cerr << "usage: " << exe << " <serial> <newLogicalName>" << endl;
    exit(1);
}

int main(int argc, const char * argv[])
{
    string      errmsg;

    // Setup the API to use local USB devices
    if(yRegisterHub("usb", errmsg) != YAPI_SUCCESS) {
        cerr << "RegisterHub error: " << errmsg << endl;
        return 1;
    }

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

    YModule *module = yFindModule(argv[1]);  // use serial or logical name

    if (module->isOnline()) {
        if (argc >= 3){
            string newname =  argv[2];
            if (!yCheckLogicalName(newname)){
                cerr << "Invalid name (" << newname << ")" << endl;
                usage(argv[0]);
            }
            module->set_logicalName(newname);
            module->saveToFlash();
        }
        cout << "Current name: " << module->get_logicalName() << endl;
    } else {
        cout << argv[1] << " not connected (check identification and USB cable)"
             << endl;
    }
    return 0;
```

```
}
```

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

Enumeration des modules

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

```
#include <iostream>

#include "yocto_api.h"

using namespace std;

int main(int argc, const char * argv[])
{
    string      errmsg;

    // Setup the API to use local USB devices
    if(yRegisterHub("usb", errmsg) != YAPI_SUCCESS) {
        cerr << "RegisterHub error: " << errmsg << endl;
        return 1;
    }

    cout << "Device list: " << endl;

    YModule *module = yFirstModule();
    while (module != NULL) {
        cout << module->get_serialNumber() << " ";
        cout << module->get_productName()  << endl;
        module = module->nextModule();
    }
    return 0;
}
```

2.3. Gestion des erreurs

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

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

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

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

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

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

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

2.4. Intégration de la librairie Yoctopuce en C++

Selon vos besoins et vos préférences, vous pouvez être mené à intégrer de différentes manières la librairie à vos projets. Cette section explique comment implémenter les différentes options.

Intégration au format source

L'intégration de toutes les sources de la librairie dans vos projets a plusieurs avantages:

- Elle garanti le respect des conventions de compilation de votre projet (32/64 bits, inclusion des symboles de debug, caractères unicode ou ASCII, etc.);
- Elle facilite le déboggage si vous cherchez la cause d'un problème lié à la librairie Yoctopuce
- Elle réduit les dépendances sur des composants tiers, par exemple pour parer au cas où vous pourriez être mené à recompiler ce projet pour une architecture différente dans de nombreuses années.
- Elle ne requiert pas l'installation d'une librairie dynamique spécifique à Yoctopuce sur le système final, tout est dans l'exécutable.

Pour intégrer le code source, le plus simple est d'inclure simplement le répertoire `Sources` de la librairie Yoctopuce à votre **IncludePath**, et d'ajouter tous les fichiers de ce répertoire (y compris le sous-répertoire `yapi`) à votre projet.

Pour que votre projet se construise ensuite correctement, il faudra linker avec votre projet les librairies systèmes requises, à savoir:

- Pour Windows: les librairies sont mises automatiquement
- Pour Mac OS X: **IOKit.framework** et **CoreFoundation.framework**
- Pour Linux: **libm**, **libpthread**, **libusb1.0** et **libstdc++**

Intégration en librairie statique

L'intégration de de la librairie Yoctopuce sous forme de librairie statique est une manière plus simple de construire un petit exécutable utilisant des modules Yoctopuce. Elle permet une compilation rapide du programme en une seule commande. Elle ne requiert pas non plus l'installation d'une librairie dynamique spécifique à Yoctopuce sur le système final, tout est dans l'exécutable.

Pour intégrer la librairie statique Yoctopuce à votre projet, vous devez inclure le répertoire `Sources` de la librairie Yoctopuce à votre **IncludePath**, et ajouter le sous-répertoire de `Binaries/...` correspondant à votre système d'exploitation à votre **LibPath**.

Ensuite, pour que votre projet se construise ensuite correctement, il faudra linker avec votre projet la librairie Yoctopuce et les librairies systèmes requises:

- Pour Windows: **yocto-static.lib**
- Pour Mac OS X: **libyocto-static.a**. **IOKit.framework** et **CoreFoundation.framework**
- Pour Linux: **libyocto-static.a**, **libm**, **libpthread**, **libusb1.0** et **libstdc++**.

Attention, sous Linux, si vous voulez compiler en ligne de commande avec GCC, il est en général souhaitable de linker les librairies systèmes en dynamique et non en statique. Pour mélanger sur la même ligne de commande des librairies statiques et dynamiques, il faut passer les arguments suivants:

```
gcc (...) -Wl,-Bstatic -lyocto-static -Wl,-Bdynamic -lm -lpthread -lusb-1.0 -lstdc++
```

Intégration en librairie dynamique

L'intégration de la librairie Yoctopuce sous forme de librairie dynamique permet de produire un exécutable plus petit que les deux méthodes précédentes, et de mettre éventuellement à jour cette librairie si un correctif s'avérait nécessaire sans devoir recompiler le code source de l'application. Par contre, c'est un mode d'intégration qui exigera systématiquement de copier la librairie dynamique sur la machine cible ou l'application devra être lancée (**yocto.dll** sous Windows, **libyocto.so.1.0.1** sous Mac OS X et Linux).

Pour intégrer la librairie dynamique Yoctopuce à votre projet, vous devez inclure le répertoire `Sources` de la librairie Yoctopuce à votre **IncludePath**, et ajouter le sous-répertoire de `Binaries/...` correspondant à votre système d'exploitation à votre **LibPath**.

Ensuite, pour que votre projet se construise ensuite correctement, il faudra linker avec votre projet la librairie dynamique Yoctopuce et les librairies systèmes requises:

- Pour Windows: **yocto.lib**
- Pour Mac OS X: **libyocto**, **IOKit.framework** et **CoreFoundation.framework**
- Pour Linux: **libyocto**, **libm**, **libpthread**, **libusb1.0** et **libstdc++**.

Avec GCC, la ligne de commande de compilation est simplement:

```
gcc (...) -lyocto -lm -lpthread -lusb-1.0 -lstdc++
```

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:

```
#include "yocto_api.h"
```

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.
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.

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

ySetDelegate(**object**)
(Objective-C uniquement) Enregistre un objet délégué qui doit se conformer au procole `YDeviceHotPlug`.

ySetTimeout(**callback**, **ms_timeout**, **optional_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.

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.

yCheckLogicalName()

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

```
bool yCheckLogicalName( const 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.

yDisableExceptions()

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

```
void yDisableExceptions( )
```

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

yEnableExceptions()

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

```
void yEnableExceptions( )
```

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

Cette fonction est utilisée uniquement sous Android.

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)

En cas d'erreur, déclenche une exception

---

yFreeAPI()

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

```
void yFreeAPI( )
```

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

---

yGetAPIVersion()

Retourne la version de la librairie Yoctopuce utilisée.

```
string yGetAPIVersion( )
```

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

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

Retourne :
une chaîne de caractères décrivant la version de la librairie.

---

yGetTickCount()

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

```
u64 yGetTickCount( )
```

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

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

---

yHandleEvents()

Maintient la communication de la librairie avec les modules Yoctopuce.

```
YRETCODE yHandleEvents( string& errmsg)
```

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

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

Paramètres :

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

Retourne :

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

yInitAPI()

Initialise la librairie de programmation de Yoctopuce explicitement.

```
YRETCODE yInitAPI( int mode, string& errmsg)
```

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

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

Paramètres :

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

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

Retourne :

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

yRegisterDeviceArrivalCallback()

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

```
void yRegisterDeviceArrivalCallback( yDeviceUpdateCallback arrivalCallback)
```

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

Paramètres :

arrivalCallback une procédure qui prend un YModule en paramètre, ou null pour supprimer un callback déja enregistré.

yRegisterDeviceRemovalCallback()

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

```
void yRegisterDeviceRemovalCallback( yDeviceUpdateCallback removalCallback)
```

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

Paramètres :

removalCallback une procédure qui prend un `YModule` en paramètre, ou `null` pour supprimer un callback déja enregistré.

yRegisterHub()

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

```
YRETCODE yRegisterHub( const string& url, string& errmsg)
```

Dans le cas d'une utilisation avec la passerelle VirtualHub, vous devez donner en paramètre l'adresse de la machine où tourne le VirtualHub (typiquement `"http://127.0.0.1:4444"`, qui désigne la machine locale). Si vous utilisez un langage qui a un accès direct à USB, vous pouvez utiliser la pseudo-adresse `"usb"` à la place.

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 VirtualHub sur lequel le controle d'accès a été activé, vous devez donner le paramètre url sous la forme: `http://nom:mot_de_passe@adresse:port`

Paramètres :

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

yRegisterLogFunction()

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

```
void yRegisterLogFunction( yLogFunction logfun)
```

Utile pour débugger le fonctionnement de l'API.

Paramètres :

logfun une procedure qui prend une chaîne de caractère en paramètre, ou `null` pour supprimer un callback déja enregistré.

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

Les methodes `yDeviceArrival` et `yDeviceRemoval` seront appelées pendant l'éxecution de la fonction `yHandleDeviceList`, que vous devrez appeler régulièrement.

Paramètres :

object	un objet qui soit se conformer au procol `YAPIDelegate`, ou `nil` pour supprimer un objet déja enregistré.

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

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

Paramètres :

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

Retourne :

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

`ySleep()`

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

```
YRETCODE ySleep( unsigned ms_duration, string& errmsg)
```

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

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

Paramètres :

ms_duration	un entier correspondant à la durée de la pause, en millisecondes
errmsg	une chaîne de caractères passée par référence, dans laquelle sera stocké un éventuel message d'erreur.

Retourne :

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

`yUnregisterHub()`

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

```
void yUnregisterHub( const string& url)
```

Paramètres :

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

---

yUpdateDeviceList()

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

```
YRETCODE yUpdateDeviceList( string& errmsg)
```

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

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

Paramètres :

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

Retourne :

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

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

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

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

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

Paramètres :

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

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

Retourne :

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

3.2. Interface de la fonction AnButton

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
#include "yocto_anbutton.h"
```

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 la fonction.

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 la fonction.

anbutton→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

anbutton→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

anbutton→get_hardwareId()
Retourne l'identifiant unique de la fonction.

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_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 255 inclus) pour le déclanchement de callbacks.

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

anbutton→isOnline()
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

anbutton→isOnline_async(callback, context)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

anbutton→load(msValidity)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

anbutton→load_async(msValidity, callback, context)
Met en cache les valeurs courantes de la fonction, 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→set_analogCalibration(newval)
Enclenche ou déclenche le procédure de calibration.

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

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

anbutton→set_logicalName(newval)
Modifie le nom logique de l'entrée analogique.

anbutton→set_sensitivity(newval)
Modifie la sensibilité pour l'entrée (entre 1 et 255 inclus) pour le déclanchement de callbacks.

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

yFindAnButton()

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

```
YAnButton* yFindAnButton( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YAnButton.isOnline()` pour tester si l'entrée analogique est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :
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.

yFirstAnButton()

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

```
YAnButton* yFirstAnButton( )
```

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

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

anbutton→describe()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

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()

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

```
Y_ANALOGCALIBRATION_enum 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()

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

```
unsigned 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()

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

```
unsigned 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()

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

```
unsigned 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()

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

```
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 fonction

anbutton→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

anbutton→get_anbuttonDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

anbutton→get_isPressed()

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

```
Y_ISPRESSED_enum 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()

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).

```
unsigned 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()

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).

```
unsigned get_lastTimeReleased( )
```

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

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

anbutton→get_logicalName()

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()

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`

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

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

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

Paramètres :

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

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

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

---

anbutton→get_rawValue()

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

```
unsigned 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()

Retourne la sensibilité pour l'entrée (entre 1 et 255 inclus) pour le déclanchement de callbacks.

```
unsigned get_sensitivity( )
```

Retourne :
un entier représentant la sensibilité pour l'entrée (entre 1 et 255 inclus) pour le déclanchement de callbacks

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

---

anbutton→get_userData()

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

```
void * 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()

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

```
bool isOnline( )
```

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

Retourne :
`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :
callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen
context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

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

anbutton→load()

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

```
YRETCODE load( int msValidity)
```

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

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

Retourne :

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

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

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

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

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)

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

Retourne :

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

---

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()

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

```
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.

---

anbutton→set_analogCalibration()

Enclenche ou déclenche le procédure de calibration.

```
int set_analogCalibration( Y_ANALOGCALIBRATION_enum 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()

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

```
int set_calibrationMax( unsigned 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()

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

```
int set_calibrationMin( unsigned 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()

Modifie le nom logique de l'entrée analogique.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

anbutton→set_sensitivity()

Modifie la sensibilité pour l'entrée (entre 1 et 255 inclus) pour le déclanchement de callbacks.

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

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 255 inclus) pour le déclanchement de callbacks

Retourne :

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

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

anbutton→set_userData()

Enregistre 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( void* 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

3.3. Interface de la fonction CarbonDioxide

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
#include "yocto_carbondioxide.h"
```

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 la fonction.

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

carbondioxide→get_currentRawValue()

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

carbondioxide→get_currentValue()
Retourne la valeur mesurée actuelle.

carbondioxide→get_errorMessage()
Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

carbondioxide→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

carbondioxide→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

carbondioxide→get_hardwareId()
Retourne l'identifiant unique de la fonction.

carbondioxide→get_highestValue()
Retourne la valeur maximale observée.

carbondioxide→get_logicalName()
Retourne le nom logique du capteur de CO2.

carbondioxide→get_lowestValue()
Retourne la valeur minimale observée.

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_resolution()
Retourne la résolution des valeurs mesurées.

carbondioxide→get_unit()
Retourne l'unité dans laquelle la valeur mesurée 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 la fonction est joignable, sans déclencher d'erreur.

carbondioxide→isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

carbondioxide→load(**msValidity**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

carbondioxide→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

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

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_logicalName(newval)
Modifie le nom logique du capteur de CO2.

carbondioxide→set_lowestValue(newval)
Modifie la mémoire de valeur minimale observée.

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`.

yFindCarbonDioxide()

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

```
YCarbonDioxide* yFindCarbonDioxide( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YCarbonDioxide.isOnline()` pour tester si le capteur de CO2 est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :
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.

yFirstCarbonDioxide()

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

```
YCarbonDioxide* yFirstCarbonDioxide( )
```

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

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

carbondioxide→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( floatArr rawValues,
                                         floatArr 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()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :

une chaîne de caractères décrivant la fonction

---

carbondioxide→get_advertisedValue()

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()

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`.

---

carbondioxide→get_currentValue()

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`.

---

carbondioxide→get_errorMessage()

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

```
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 fonction

carbondioxide→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

carbondioxide→get_carbondioxideDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

carbondioxide→get_highestValue()

Retourne la valeur maximale observée.

```
double get_highestValue( )
```

Retourne :
une valeur numérique représentant la valeur maximale observée

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

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()

Retourne la valeur minimale observée.

```
double get_lowestValue( )
```

Retourne :
une valeur numérique représentant la valeur minimale observée

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

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`

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

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

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

Paramètres :

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

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

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

carbondioxide→get_resolution()

Retourne la résolution des valeurs mesurées.

```
double get_resolution( )
```

La résolution correspond à la précision de la représentation numérique 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()

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

```
string get_unit( )
```

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la valeur mesurée est exprimée

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

---

carbondioxide→get_userData()

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

```
void * 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()

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

```
bool isOnline( )
```

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

Retourne :
`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :

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

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

Retourne :

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

carbondioxide→load()

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

```
YRETCODE load( int msValidity)
```

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

Paramètres :

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

Retourne :

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

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

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

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

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

Retourne :

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

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→registerValueCallback()

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

```
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.

---

carbondioxide→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.

---

carbondioxide→set_logicalName()

Modifie le nom logique du capteur de CO2.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

---

carbondioxide→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.

---

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( void* 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 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:
```
#include "yocto_colorled.h"
```

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 la fonction.

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 fonction.

colorled→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

colorled→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

colorled→get_hardwareId()
Retourne l'identifiant unique de la fonction.

colorled→get_hslColor()
Retourne la couleur HSL courante de la led.

colorled→get_logicalName()
Retourne le nom logique de la led RGB.

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

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

colorled→get_rgbColor()
Retourne la couleur RGB courante de la led.

colorled→get_rgbColorAtPowerOn()

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

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

colorled→hslMove(hsl_target, ms_duration)
Effectue une transition continue dans l'espace HSL entre la couleur courante et une nouvelle couleur.

colorled→isOnline()
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

colorled→isOnline_async(callback, context)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

colorled→load(msValidity)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

colorled→load_async(msValidity, callback, context)
Met en cache les valeurs courantes de la fonction, 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`.

yFindColorLed()

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

```
YColorLed* yFindColorLed( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YColorLed.isOnline()` pour tester si la led RGB est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :
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.

---

yFirstColorLed()

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

```
YColorLed* yFirstColorLed( )
```

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

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

---

colorled→describe()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

---

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()

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

```
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 fonction

---

colorled→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

colorled→get_colorledDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

colorled→get_hslColor()

Retourne la couleur HSL courante de la led.

```
unsigned 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()

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()

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

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

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

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

Paramètres :

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

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

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

---

colorled→get_rgbColor()

Retourne la couleur RGB courante de la led.

```
unsigned 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()

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

```
unsigned 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()

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

```
void * 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()

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()

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

```
bool isOnline( )
```

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

Retourne :
`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :

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

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

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

---

colorled→load()

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

```
YRETCODE load( int msValidity)
```

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

Paramètres :

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

Retourne :

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

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

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

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

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

Retourne :

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

---

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()

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

```
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.

colorled→rgbMove()

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()

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

```
int set_hslColor( unsigned newval)
```

L'encodage est réalisé de la manière suivante: 0xHHSSLL.

Paramètres :

newval un entier représentant la couleur courante de la led, en utilisant une couleur HSL spécifiée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→set_logicalName()

Modifie le nom logique de la led RGB.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

colorled→set_rgbColor()

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

```
int set_rgbColor( unsigned 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()

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

```
int set_rgbColorAtPowerOn( unsigned newval)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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( void* 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

3.5. Interface de la fonction Current

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
#include "yocto_current.h"
```

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 la fonction.
current→get_advertisedValue() Retourne la valeur courante du capteur de courant (pas plus de 6 caractères).
current→get_currentRawValue()

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

current→get_currentValue()
Retourne la valeur mesurée actuelle.

current→get_errorMessage()
Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

current→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

current→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

current→get_hardwareId()
Retourne l'identifiant unique de la fonction.

current→get_highestValue()
Retourne la valeur maximale observée.

current→get_logicalName()
Retourne le nom logique du capteur de courant.

current→get_lowestValue()
Retourne la valeur minimale observée.

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_resolution()
Retourne la résolution des valeurs mesurées.

current→get_unit()
Retourne l'unité dans laquelle la valeur mesurée 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 la fonction est joignable, sans déclencher d'erreur.

current→isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

current→load(**msValidity**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

current→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

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

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_logicalName(**newval**)

Modifie le nom logique du capteur de courant.

current→set_lowestValue(newval)
Modifie la mémoire de valeur minimale observée.

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`.

yFindCurrent()

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

```
YCurrent* yFindCurrent( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YCurrent.isOnline()` pour tester si le capteur de courant est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :
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.

yFirstCurrent()

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

```
YCurrent* yFirstCurrent( )
```

Utiliser la fonction `YCurrent.nextCurrent()` pour itérer sur les autres capteurs de courant.

Retourne :
un pointeur sur un objet `YCurrent`, correspondant à le premier capteur de courant accessible en ligne, ou `null` si il n'y a pas de capteurs de courant disponibles.

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( floatArr rawValues,
                                         floatArr 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()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :

une chaîne de caractères décrivant la fonction

---

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()

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`.

---

current→get_currentValue()

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`.

---

current→get_errorMessage()

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

```
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 fonction

current→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

current→get_currentDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

current→get_highestValue()

Retourne la valeur maximale observée.

```
double get_highestValue( )
```

Retourne :
une valeur numérique représentant la valeur maximale observée

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

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()

Retourne la valeur minimale observée.

```
double get_lowestValue( )
```

Retourne :
une valeur numérique représentant la valeur minimale observée

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

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`

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

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

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

Paramètres :

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

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

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

current→get_resolution()

Retourne la résolution des valeurs mesurées.

```
double get_resolution( )
```

La résolution correspond à la précision de la représentation numérique 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()

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

```
string get_unit( )
```

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la valeur mesurée est exprimée

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

---

current→get_userData()

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

```
void * 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()

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

```
bool isOnline( )
```

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

Retourne :
`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :
callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen
context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

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

current→load()

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

```
YRETCODE load( int msValidity)
```

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

Paramètres :

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

Retourne :

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

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

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

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

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

Retourne :

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

current→nextCurrent()

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→registerValueCallback()

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

```
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.

---

current→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.

---

current→set_logicalName()

Modifie le nom logique du capteur de courant.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

---

current→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.

---

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( void* 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 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 librairie de programmation Yoctopuce permet de contrôler le fonctionnement de l'enregistreur de données interne. Dans la mesure où les capteurs n'ont pas de pile intégrée, ils ne contiennent pas de référence de temps absolue. C'est pourquoi les mesures sont simplement indexées par le numéro de Run (période continue de fonctionnement lors d'une mise sous tension), et à l'intervalle de temps depuis le début du Run. Il est par contre possible d'indiquer par logiciel à l'enregistreur de données l'heure UTC à un moment donnée, afin qu'il en tienne compte jusqu'à la prochaine mise hors tension.

Pour utiliser les fonctions décrites ici, vous devez inclure:
```
#include "yocto_datalogger.h"
```

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 la fonction.

datalogger→forgetAllDataStreams()
Efface tout l'historique des mesures de l'enregistreur de données.

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

datalogger→get_autoStart()
Retourne le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

datalogger→get_currentRunIndex()
Retourne le numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active.

datalogger→get_dataRun(**runIdx**)
Retourne un objet YDataRun contenant toutes les données mesurées pour une période d'enclanchement du module donnée (un Run).

datalogger→get_dataStreams(**v**)
Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur.

datalogger→get_errorMessage()
Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

datalogger→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

datalogger→get_functionDescriptor()

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

datalogger→get_hardwareId()
Retourne l'identifiant unique de la fonction.

datalogger→get_logicalName()
Retourne le nom logique de l'enregistreur de données.

datalogger→get_measureNames()
Retourne les noms des valeurs mesurées par 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_oldestRunIndex()
Retourne le numéro du Run le plus ancien pour lequel la mémoire non-volatile contient encore des données.

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 la fonction est joignable, sans déclencher d'erreur.

datalogger→isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

datalogger→load(**msValidity**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

datalogger→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

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

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

datalogger→set_autoStart(**newval**)
Modifie le mode d'activation automatique de l'enregistreur de données à la mise sous tension.

datalogger→set_logicalName(**newval**)
Modifie le nom logique de l'enregistreur de données.

datalogger→set_recording(**newval**)
Modifie l'état d'activation de l'enregistreur de données.

datalogger→set_timeUTC(**newval**)
Modifie la référence de temps UTC, afin de l'attacher aux données enregistrées.

datalogger→set_userData(**data**)

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

yFindDataLogger()

Permet de retrouver un enregistreur de données d'après un identifiant donné.

```
YDataLogger* yFindDataLogger( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDataLogger.isOnline()` pour tester si l'enregistreur de données est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :
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.

yFirstDataLogger()

Commence l'énumération des enregistreurs de données accessibles par la librairie.

```
YDataLogger* yFirstDataLogger( )
```

Utiliser la fonction `YDataLogger.nextDataLogger()` pour itérer sur les autres enregistreurs de données.

Retourne :
un pointeur sur un objet `YDataLogger`, correspondant à le premier enregistreur de données accessible en ligne, ou `null` si il n'y a pas de enregistreurs de données disponibles.

datalogger→describe()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

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()

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()

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

```
Y_AUTOSTART_enum get_autoStart( )
```

Retourne :
soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon le mode d'activation automatique de l'enregistreur de données à la mise sous tension

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

---

datalogger→get_currentRunIndex()

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.

```
unsigned 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`.

Retourne un objet YDataRun contenant toutes les données mesurées pour une période d'enclanchement du module donnée (un Run).

Cet objet pourra être utilisé pour récupérer les mesures (valeur min, valeur moyenne et valeur max) avec la granularité désirée.

Paramètres :
runIdx l'index du Run désiré

Retourne :
un objet `YDataRun`

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

---

datalogger→get_dataStreams()

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur.

```
int get_dataStreams( )
```

L'appelant doit passer par référence un tableau vide pout stocker les objets YDataStream, et la méthode va les remplire avec des objets décrivant les séquences de données disponibles.

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()

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

```
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 fonction

---

datalogger→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

---

datalogger→get_dataloggerDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :

une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

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`.

Retourne les noms des valeurs mesurées par l'enregistreur de données.

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.

---

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`

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

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

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

Paramètres :

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

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

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

---

datalogger→get_oldestRunIndex()

Retourne le numéro du Run le plus ancien pour lequel la mémoire non-volatile contient encore des données.

```
unsigned get_oldestRunIndex( )
```

Retourne :
un entier représentant le numéro du Run le plus ancien pour lequel la mémoire non-volatile contient encore des données

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

datalogger→get_recording()

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

```
Y_RECORDING_enum 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()

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

```
unsigned 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()

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

```
void * get_userData( )
```

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

Retourne :
l'objet stocké précédemment par l'appelant.

datalogger→isOnline()

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

```
bool isOnline( )
```

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

Retourne :
`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :

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

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

Retourne :

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

---

datalogger→load()

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

```
YRETCODE load( int msValidity)
```

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

Paramètres :

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

Retourne :

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

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

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

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

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

Retourne :

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

---

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()

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

```
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.

---

datalogger→set_autoStart()

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

```
int set_autoStart( Y_AUTOSTART_enum newval)
```

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

Paramètres :
newval soit `Y_AUTOSTART_OFF`, soit `Y_AUTOSTART_ON`, selon le mode d'activation automatique de l'enregistreur de données à la mise sous tension

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

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

---

datalogger→set_logicalName()

Modifie le nom logique de l'enregistreur de données.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

---

datalogger→set_recording()

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

```
int set_recording( Y_RECORDING_enum 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()`

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

```
int set_timeUTC( unsigned 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()`

Enregistre 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( void* 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. 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:
```
#include "yocto_datalogger.h"
```

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.

Retourne la valeur moyenne des mesures observées au moment choisi.

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.

Retourne la durée (en secondes) du Run.

Lorsque cette méthode est appellée 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.

Retourne la valeur maximale des mesures observées au moment choisi.

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.

Retourne les noms des valeurs mesurées par l'enregistreur de données.

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.

Retourne la valeur minimale des mesures observées au moment choisi.

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.

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.

Retourne le nombre de valeurs accessibles dans ce Run, étant donné l'intervalle de temps choisi entre les valeurs.

Lorsque cette méthode est appellée dur le Run courant et que l'enregistreur de données est actif, l'appel à cette méthode force un rechargement de la dernière séquence du module pour s'assurer que la réponse prend en compte les dernières données enregistrées.

Retourne :

un entier positif correspondant à la durée du Run divisée par l'intervalle entre les valeurs.

Retourne l'intervalle de temps représenté par chaque valeur de ce run.

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.

Change l'intervalle de temps représenté par chaque valeur de ce run.

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

3.8. Séquence de données enregistrées

Les objets DataStream représentent des séquences de mesures enregistrées. Ils sont retournés par l'enregistreur de données présent dans les senseurs de Yoctopuce.

Pour utiliser les fonctions décrites ici, vous devez inclure:
```
#include "yocto_datalogger.h"
```

Méthodes des objets `YDataStream`

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_dataSamplesInterval()
Retourne le nombre de secondes entre chaque mesure de la séquence.

datastream→get_rowCount()
Retourne le nombre d'enregistrement contenus dans la séquence.

datastream→get_runIndex()
Retourne le numéro de Run de la séquence de données.

datastream→get_startTime()
Retourne le nombre de secondes entre le début du Run (mise sous tension du module) et le début de la séquence de données.

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_columnCount()`

Retourne le nombre de colonnes de données contenus dans la séquence.

```
unsigned get_columnCount( )
```

La sémantique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Cette méthode déclanche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

Retourne :
un entier positif correspondant au nombre de colonnes.

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

`datastream→get_columnNames()`

Retourne le nom (la sémantique) des colonnes de données contenus dans la séquence.

```
const vector<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 d'archivage résumant des séquence, un suffixe est ajouté à l'identifiant du capteur: _min pour la valeur minimale, _avg pour la valeur moyenne et _max pour la valeur maximale.

Cette méthode déclanche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

Retourne :
une liste de chaîne de caractères.

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

---

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( unsigned row, unsigned col)
```

La sémentique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode get_columnNames().

Cette méthode déclanche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

Paramètres :
row index de l'enregistrement (ligne)
col index de la mesure (colonne)

Retourne :
un nombre décimal

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

---

datastream→get_dataRows()

Retourne toutes les données mesurées contenues dans la séquence, sous forme d'une liste de vecteurs (table bidimensionnelle).

```
const vector< vector<double> >& get_dataRows( )
```

La sémentique des données présentes dans chaque colonne peut être obtenue à l'aide de la méthode `get_columnNames()`.

Cette méthode déclanche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

Retourne :
une liste d'enregistrements, chaque enregistrement étant lui-même une liste de nombres décimaux.

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

---

datastream→get_dataSamplesInterval()

Retourne le nombre de secondes entre chaque mesure de la séquence.

```
unsigned get_dataSamplesInterval( )
```

Par défaut, l'enregistreur mémorise une mesure par seconde, mais la création de séquences d'archive synthétisant de plus longue période peut produire des séquences plus espacées.

Cette méthode ne provoque pas d'accès au module, les données étant préchargées dans l'objet au moment où il est instancié.

Retourne :
un entier positif correspondant au nombre de secondes entre deux mesures consécutives.

datastream→get_rowCount()

Retourne le nombre d'enregistrement contenus dans la séquence.

```
unsigned get_rowCount( )
```

Cette méthode déclanche le chargement de toutes les données de la séquence, si cela n'était pas encore fait.

Retourne :
un entier positif correspondant au nombre d'enregistrements.

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

datastream→get_runIndex()

Retourne le numéro de Run de la séquence de données.

```
unsigned get_runIndex( )
```

Un Run peut être composé de plusieurs séquences, couvrant différents intervalles de temps.

Cette méthode ne provoque pas d'accès au module, les données étant préchargées dans l'objet au moment où il est instancié.

Retourne :
un entier positif correspondant au numéro du Run

datastream→get_startTime()

Retourne le nombre de secondes entre le début du Run (mise sous tension du module) et le début de la séquence de données.

```
unsigned get_startTime( )
```

Si vous désirez obtenir l'heure absolue du début de la séquence, utilisez `get_startTimeUTC()`.

Cette méthode ne provoque pas d'accès au module, les données étant préchargées dans l'objet au moment où il est instancié.

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()

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).

```
const time_t & get_startTimeUTC( )
```

Si l'heure UTC n'était pas configurée dans l'enregistreur de données au début de la séquence, cette méthode retourne 0.

Cette méthode ne provoque pas d'accès au module, les données étant préchargées dans l'objet au moment où il est instancié.

Retourne :
un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et le début de la séquence enregistrée.

3.9. 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:
```
#include "yocto_dualpower.h"
```

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 la fonction.

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 de la fonction.

dualpower→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

dualpower→get_extVoltage()
Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.

dualpower→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

dualpower→get_hardwareId()
Retourne l'identifiant unique de la fonction.

dualpower→get_logicalName()
Retourne le nom logique du contrôle d'alimentation.

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

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

dualpower→get_powerControl()
Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

dualpower→get_powerState()
Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

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

dualpower→isOnline()
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

dualpower→isOnline_async(callback, context)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

dualpower→load(msValidity)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

dualpower→load_async(msValidity, callback, context)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

dualpower→nextDualPower()
Continue l'énumération des contrôles d'alimentation commencée à l'aide de `yFirstDualPower()`.

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

dualpower→set_logicalName(newval)
Modifie le nom logique du contrôle d'alimentation.

dualpower→set_powerControl(newval)
Modifie le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

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

yFindDualPower()

Permet de retrouver un contrôle d'alimentation d'après un identifiant donné.

```
YDualPower* yFindDualPower( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDualPower.isOnline()` pour tester si le contrôle d'alimentation est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :
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.

yFirstDualPower()

Commence l'énumération des contrôles d'alimentation accessibles par la librairie.

```
YDualPower* yFirstDualPower( )
```

Utiliser la fonction `YDualPower.nextDualPower()` pour itérer sur les autres contrôles d'alimentation.

Retourne :
un pointeur sur un objet `YDualPower`, correspondant à le premier contrôle d'alimentation accessible en ligne, ou `null` si il n'y a pas de contrôles d'alimentation disponibles.

---

dualpower→describe()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

---

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()

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

```
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 fonction

---

dualpower→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

---

dualpower→get_extVoltage()

Retourne la tension mesurée sur l'alimentation de puissance externe, en millivolts.

```
unsigned 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_dualpowerDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

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()

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`

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

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui

n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

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

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

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

---

dualpower→get_powerControl()

Retourne le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

```
Y_POWERCONTROL_enum 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()

Retourne la source d'alimentation active pour les fonctions du module consommant beaucoup de courant.

```
Y_POWERSTATE_enum 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()

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

```
void * 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()

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

```
bool isOnline( )
```

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

Retourne :

`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :

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

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

Retourne :

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

dualpower→load()

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

```
YRETCODE load( int msValidity)
```

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

Paramètres :

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

Retourne :

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

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

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

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

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

Retourne :

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

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()

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

```
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.

dualpower→set_logicalName()

Modifie le nom logique du contrôle d'alimentation.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

dualpower→set_powerControl()

Modifie le mode d'alimentation choisi pour les fonctions du module consommant beaucoup de courant.

```
int set_powerControl( Y_POWERCONTROL_enum 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()

Enregistre 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( void* 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. Interface d'un port de Yocto-hub

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
#include "yocto_hubport.h"
```

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 la fonction.

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 de la fonction.

hubport→get_errorType()

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

hubport→get_functionDescriptor()

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

hubport→get_hardwareId()

Retourne l'identifiant unique de la fonction.

hubport→get_logicalName()

Retourne le nom logique du port de Yocto-hub, qui est toujours le numéro de série du module qui y est connecté.

hubport→get_module()

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

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

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

hubport→get_portState()

Retourne l'état actuel du port de Yocto-hub.

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

hubport→isOnline()
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

hubport→isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

hubport→load(**msValidity**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

hubport→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

hubport→nextHubPort()
Continue l'énumération des port de Yocto-hub commencée à l'aide de `yFirstHubPort()`.

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

hubport→set_enabled(**newval**)
Modifie le mode d'activation du port du Yocto-hub.

hubport→set_logicalName(**newval**)
Il n'est pas possible de configurer le nom logique d'un port de Yocto-hub.

hubport→set_userData(**data**)
Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

yFindHubPort()

Permet de retrouver un port de Yocto-hub d'après un identifiant donné.

```
YHubPort* yFindHubPort( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YHubPort.isOnline()` pour tester si le port de Yocto-hub est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :
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.

yFirstHubPort()

Commence l'énumération des port de Yocto-hub accessibles par la librairie.

```
YHubPort* yFirstHubPort( )
```

Utiliser la fonction `YHubPort.nextHubPort()` pour itérer sur les autres port de Yocto-hub.

Retourne :
un pointeur sur un objet `YHubPort`, correspondant à le premier port de Yocto-hub accessible en ligne, ou `null` si il n'y a pas de port de Yocto-hub disponibles.

---

hubport→describe()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

---

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()

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()

Retourne vrai si le port du Yocto-hub est alimenté, faux sinon.

```
Y_ENABLED_enum 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()

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

```
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 fonction

---

hubport→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

---

hubport→get_hubportDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

---

hubport→get_logicalName()

Retourne le nom logique du port de Yocto-hub, qui est toujours le numéro de série du module qui y est connecté.

```
string get_logicalName( )
```

Retourne :
une chaîne de caractères représentant le nom logique du port de Yocto-hub, qui est toujours le numéro de série du module qui y est connecté

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

---

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

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

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

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

Paramètres :

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

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

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

---

hubport→get_portState()

Retourne l'état actuel du port de Yocto-hub.

```
Y_PORTSTATE_enum get_portState( )
```

Retourne :
une valeur parmi Y_PORTSTATE_OFF, Y_PORTSTATE_ON et Y_PORTSTATE_RUN représentant l'état actuel du port de Yocto-hub

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

---

hubport→get_userData()

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

```
void * 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()

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

```
bool isOnline( )
```

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

Retourne :
`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :
callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen
context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

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

---

hubport→load()

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

```
YRETCODE load( int msValidity)
```

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

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

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

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

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

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

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

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

---

hubport→nextHubPort()

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()

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

```
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.

---

hubport→set_enabled()

Modifie le mode d'activation du port du Yocto-hub.

```
int set_enabled( Y_ENABLED_enum 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()

Il n'est pas possible de configurer le nom logique d'un port de Yocto-hub.

```
int set_logicalName( const string& newval)
```

Son nom est automatiquement configuré comme le numéro de série du module qui y est connecté.

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.

`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( void* 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

3.11. Interface de la fonction Humidity

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

Pour utiliser les fonctions décrites ici, vous devez inclure:
```
#include "yocto_humidity.h"
```

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 la fonction.

humidity→get_advertisedValue()
Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

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

humidity→get_currentValue()
Retourne la valeur mesurée actuelle.

humidity→get_errorMessage()
Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

humidity→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

humidity→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

humidity→get_hardwareId()
Retourne l'identifiant unique de la fonction.

humidity→get_highestValue()
Retourne la valeur maximale observée.

humidity→get_logicalName()
Retourne le nom logique du capteur d'humidité.

humidity→get_lowestValue()
Retourne la valeur minimale observée.

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_resolution()
Retourne la résolution des valeurs mesurées.

humidity→get_unit()
Retourne l'unité dans laquelle la valeur mesurée 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 la fonction est joignable, sans déclencher d'erreur.

humidity→isOnline_async(callback, context)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

humidity→load(msValidity)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

humidity→load_async(msValidity, callback, context)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

humidity→nextHumidity()
Continue l'énumération des capteurs d'humidité commencée à l'aide de `yFirstHumidity()`.

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_logicalName(newval)
Modifie le nom logique du capteur d'humidité.

humidity→set_lowestValue(newval)
Modifie la mémoire de valeur minimale observée.

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`.

yFindHumidity()

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

```
YHumidity* yFindHumidity( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YHumidity.isOnline()` pour tester si le capteur d'humidité est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

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.

---

yFirstHumidity()

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

```
YHumidity* yFirstHumidity( )
```

Utiliser la fonction `YHumidity.nextHumidity()` pour itérer sur les autres capteurs d'humidité.

Retourne :

un pointeur sur un objet `YHumidity`, correspondant à le premier capteur d'humidité accessible en ligne, ou `null` si il n'y a pas de capteurs d'humidité disponibles.

---

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( floatArr rawValues,
                                         floatArr 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()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

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()

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`.

humidity→get_currentValue()

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`.

humidity→get_errorMessage()

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

```
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 fonction

humidity→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

humidity→get_humidityDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

humidity→get_highestValue()

Retourne la valeur maximale observée.

```
double get_highestValue( )
```

Retourne :
une valeur numérique représentant la valeur maximale observée

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

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()

Retourne la valeur minimale observée.

```
double get_lowestValue( )
```

Retourne :
une valeur numérique représentant la valeur minimale observée

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

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`

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

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

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

Paramètres :

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

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

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

humidity→get_resolution()

Retourne la résolution des valeurs mesurées.

```
double get_resolution( )
```

La résolution correspond à la précision de la représentation numérique 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()

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

```
string get_unit( )
```

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la valeur mesurée est exprimée

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

---

humidity→get_userData()

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

```
void * 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()

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

```
bool isOnline( )
```

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

Retourne :
`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :

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

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

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

---

humidity→load()

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

```
YRETCODE load( int msValidity)
```

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

utilisée pour marquer occasionellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

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

Retourne :

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

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

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

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

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

Retourne :

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

---

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→registerValueCallback()

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

```
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.

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_logicalName()

Modifie le nom logique du capteur d'humidité.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

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_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( void* 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

3.12. 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:
```
#include "yocto_led.h"
```

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 la fonction.

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 fonction.

led→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

led→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

led→get_hardwareId()
Retourne l'identifiant unique de la fonction.

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 fonction est joignable, sans déclencher d'erreur.

led→isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

led→load(**msValidity**)

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

led→load_async(msValidity, callback, context)
Met en cache les valeurs courantes de la fonction, 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`.

yFindLed()

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

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

Paramètres :
func une chaîne de caractères qui référence la led sans ambiguïté

Retourne :
un objet de classe `YLed` qui permet ensuite de contrôler la led.

yFirstLed()

Commence l'énumération des leds accessibles par la librairie.

```
YLed* yFirstLed( )
```

Utiliser la fonction `YLed.nextLed()` pour itérer sur les autres leds.

Retourne :

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

led→describe()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

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()

Retourne le mode de signalisation de la led.

```
Y_BLINKING_enum 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()

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

```
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 fonction

led→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

---

led→get_ledDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

---

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()

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()

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

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

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

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

Paramètres :

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

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

Retourne :

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

---

led→get_power()

Retourne l'état courant de la led.

```
Y_POWER_enum 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()

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

```
void * 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()

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

```
bool isOnline( )
```

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

Retourne :

`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :

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

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

Retourne :

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

`led→load()`

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

```
YRETCODE load( int msValidity)
```

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

Paramètres :

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

Retourne :

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

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

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

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

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

Retourne :

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

`led→nextLed()`

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()

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

```
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.

led→set_blinking()

Modifie le mode de signalisation de la led.

```
int set_blinking( Y_BLINKING_enum 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()

Modifie le nom logique de la led.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

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()`

Modifie l'état courant de la led.

```
int set_power( Y_POWER_enum 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()`

Enregistre 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( void* 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

3.13. Interface de la fonction LightSensor

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

Pour utiliser les fonctions décrites ici, vous devez inclure:
```
#include "yocto_lightsensor.h"
```

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 la fonction.

lightsensor→get_advertisedValue()
Retourne la valeur courante du capteur de lumière (pas plus de 6 caractères).

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

lightsensor→get_currentValue()
Retourne la valeur mesurée actuelle.

lightsensor→get_errorMessage()
Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

lightsensor→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

lightsensor→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

lightsensor→get_hardwareId()
Retourne l'identifiant unique de la fonction.

lightsensor→get_highestValue()
Retourne la valeur maximale observée.

lightsensor→get_logicalName()
Retourne le nom logique du capteur de lumière.

lightsensor→get_lowestValue()
Retourne la valeur minimale observée.

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_resolution()
Retourne la résolution des valeurs mesurées.

lightsensor→get_unit()
Retourne l'unité dans laquelle la valeur mesurée 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 la fonction est joignable, sans déclencher d'erreur.

lightsensor→isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

lightsensor→load(**msValidity**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

lightsensor→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

lightsensor→nextLightSensor()
Continue l'énumération des capteurs de lumière commencée à l'aide de `yFirstLightSensor()`.

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_logicalName(newval)
Modifie le nom logique du capteur de lumière.

lightsensor→set_lowestValue(newval)
Modifie la mémoire de valeur minimale observée.

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`.

yFindLightSensor()

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

```
YLightSensor* yFindLightSensor( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YLightSensor.isOnline()` pour tester si le capteur de lumière est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :
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.

yFirstLightSensor()

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

```
YLightSensor* yFirstLightSensor( )
```

Utiliser la fonction `YLightSensor.nextLightSensor()` pour itérer sur les autres capteurs de lumière.

Retourne :
un pointeur sur un objet `YLightSensor`, correspondant à le premier capteur de lumière accessible en ligne, ou `null` si il n'y a pas de capteurs de lumière disponibles.

lightsensor→calibrate()

Modifie le paramètre de calibration spécifique du senseur de sorte à ce que la valeur actuelle corresponde à une consigne donnée (correction linéaire).

```
int calibrate( double calibratedVal)
```

Paramètres :

calibratedVal la consigne de valeur désirée.

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

Retourne :

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

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

---

lightsensor→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( floatArr rawValues,
                                          floatArr 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()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :

une chaîne de caractères décrivant la fonction

---

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()

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`.

lightsensor→get_currentValue()

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`.

lightsensor→get_errorMessage()

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

```
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 fonction

lightsensor→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

lightsensor→get_lightsensorDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

lightsensor→get_highestValue()

Retourne la valeur maximale observée.

```
double get_highestValue( )
```

Retourne :
une valeur numérique représentant la valeur maximale observée

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

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()

Retourne la valeur minimale observée.

```
double get_lowestValue( )
```

Retourne :
une valeur numérique représentant la valeur minimale observée

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

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`

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

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

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

Paramètres :

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

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

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

---

lightsensor→get_resolution()

Retourne la résolution des valeurs mesurées.

```
double get_resolution( )
```

La résolution correspond à la précision de la représentation numérique 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()

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

```
string get_unit( )
```

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la valeur mesurée est exprimée

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

---

lightsensor→get_userData()

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

```
void * get_userData( )
```

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

Retourne :
l'objet stocké précédemment par l'appelant.

---

lightsensor→isOnline()

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

```
bool isOnline( )
```

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

Retourne :
`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :
callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen
context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

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

`lightsensor→load()`

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

```
YRETCODE load( int msValidity)
```

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

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

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

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

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

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

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

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

---

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→registerValueCallback()

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

```
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.

---

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_logicalName()

Modifie le nom logique du capteur de lumière.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

`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_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( void* 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

3.14. 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:
`#include "yocto_api.h"`

Fonction globales

yFindModule(func)
Permet de retrouver un module d'après son numéro de série ou son nom logique.

yFirstModule()
Commence l'énumération des modules accessibles par la librairie.

Méthodes des objets `YModule`

module→describe()
Retourne un court texte décrivant le module.

module→functionCount()
Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

module→functionId(functionIndex)
Retourne l'identifiant matériel de la *n*ième fonction du module.

module→functionName(functionIndex)

Retourne le nom logique de la *n*ième fonction du module.

module→functionValue(functionIndex)
Retourne la valeur publiée par la *n*ième fonction du module.

module→get_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_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

module→get_hardwareId()
Retourne l'identifiant unique du module.

module→get_icon2d()
Retourne l'icone du module.

module→get_logicalName()
Retourne le nom logique du module.

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

module→get_persistentSettings()
Retourne l'état courant des réglages persistents du module.

module→get_productId()
Retourne l'identifiant USB du module, préprogrammé en usine.

module→get_productName()
Retourne le nom commercial du module, préprogrammé en usine.

module→get_productRelease()
Retourne le numéro de version matériel du module, préprogrammé en usine.

module→get_rebootCountdown()
Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

module→get_serialNumber()
Retourne le numéro de série du module, préprogrammé en usine.

module→get_upTime()
Retourne le numbre de millisecondes écoulées depuis la mise sous tension du module

module→get_usbBandwidth()
Retourne le nombre d'interface USB utilisé par le module.

module→get_usbCurrent()
Retourne le courant consommé par le module sur le bus USB, en milliampères.

module→get_userData()

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

module→isOnline()
Vérifie si le module est joignable, sans déclencher d'erreur.

module→isOnline_async(**callback**, **context**)
Vérifie si le module est joignable, sans déclencher d'erreur.

module→load(**msValidity**)
Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

module→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

module→nextModule()
Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

module→reboot(**secBeforeReboot**)
Agende un simple redémarrage du module dans un nombre donné de secondes.

module→revertFromFlash()
Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

module→saveToFlash()
Sauve les réglages courants dans la mémoire non volatile du module.

module→set_beacon(**newval**)
Allume ou éteint la balise de localisation du module.

module→set_logicalName(**newval**)
Change le nom logique du module.

module→set_luminosity(**newval**)
Modifie la luminosité des leds informatives du module.

module→set_usbBandwidth(**newval**)
Modifie le nombre d'interface USB utilisé par le module.

module→set_userData(**data**)
Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

module→triggerFirmwareUpdate(**secBeforeReboot**)
Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

yFindModule()

Permet de retrouver un module d'après son numéro de série ou son nom logique.

```
YModule* yFindModule( const string& func)
```

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

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.

---

yFirstModule()

Commence l'énumération des modules accessibles par la librairie.

```
YModule* yFirstModule( )
```

Utiliser la fonction `YModule.nextModule()` pour itérer sur les autres modules.

Retourne :
un pointeur sur un objet `YModule`, correspondant au premier module accessible en ligne, ou `null` si aucun module n'a été trouvé.

---

module→describe()

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→functionCount()

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

```
int functionCount( )
```

Retourne :
le nombre de fonctions sur le module

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

---

module→functionId()

Retourne l'identifiant matériel de la *n*ième fonction du module.

```
string functionId( int functionIndex)
```

Paramètres :
functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :
une chaîne de caractères correspondant à l'identifiant matériel unique de la fonction désirée

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

---

module→functionName()

Retourne le nom logique de la *n*ième fonction du module.

```
string functionName( int functionIndex)
```

Paramètres :
functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :
une chaîne de caractères correspondant au nom logique de la fonction désirée

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

---

module→functionValue()

Retourne la valeur publiée par la *n*ième fonction du module.

```
string functionValue( int functionIndex)
```

Paramètres :
functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :
une chaîne de caractères correspondant à la valeur publiée par la fonction désirée

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

---

module→get_beacon()

Retourne l'état de la balise de localisation.

```
Y_BEACON_enum 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()

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()

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

```
YRETCODE 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()

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_moduleDescriptor()

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

```
YFUN_DESCR 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`

---

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

Retourne l'icone du module.

L'icone est au format png et a une taille maximale de 1024 octets.

Retourne :
un buffer binaire contenant l'icone, au format png.

---

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()

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()

Retourne l'état courant des réglages persistents du module.

```
Y_PERSISTENTSETTINGS_enum 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()

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()

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()

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()

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()

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()

Retourne le numbre de millisecondes écoulées depuis la mise sous tension du module

```
unsigned get_upTime( )
```

Retourne :
un entier représentant le numbre de millisecondes écoulées depuis la mise sous tension du module

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

module→get_usbBandwidth()

Retourne le nombre d'interface USB utilisé par le module.

```
Y_USBBANDWIDTH_enum get_usbBandwidth( )
```

Retourne :
soit `Y_USBBANDWIDTH_SIMPLE`, soit `Y_USBBANDWIDTH_DOUBLE`, selon le nombre d'interface USB utilisé par le module

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

module→get_usbCurrent()

Retourne le courant consommé par le module sur le bus USB, en milliampères.

```
unsigned 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()

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

```
void * get_userData( )
```

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

Retourne :
l'objet stocké précédemment par l'appelant.

module→isOnline()

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

```
bool 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

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

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

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

Paramètres :
callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet module concerné et le résultat booléen
context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

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

---

module→load()

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

```
YRETCODE load( int msValidity)
```

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

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

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

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

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

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

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet module concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

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

---

module→nextModule()

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()

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→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()

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'appelez pas cette fonction dans une boucle.

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

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

---

module→set_beacon()

Allume ou éteint la balise de localisation du module.

```
int set_beacon( Y_BEACON_enum 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()

Change le nom logique du module.

```
int set_logicalName( const 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()

Modifie la luminosité des leds informatives du module.

```
int set_luminosity( int newval)
```

Le paramêtre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :
newval un entier représentant la luminosité des leds informatives du module

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

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

---

module→set_usbBandwidth()

Modifie le nombre d'interface USB utilisé par le module.

```
int set_usbBandwidth( Y_USBBANDWIDTH_enum newval)
```

Paramètres :
newval soit `Y_USBBANDWIDTH_SIMPLE`, soit `Y_USBBANDWIDTH_DOUBLE`, selon le nombre d'interface USB utilisé par le module

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

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

---

module→set_userData()

Enregistre 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( void* data)
```

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

Paramètres :
data objet quelconque à mémoriser

module→triggerFirmwareUpdate()

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.

3.15. Interface de la fonction Network

Les objets YNetwork permettent de controller les paramètres TCP/IP des modules Yoctopuce dotés d'une interface réseau.

Pour utiliser les fonctions décrites ici, vous devez inclure:
```
#include "yocto_network.h"
```

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 la fonction.

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 si il a été configuré, ou sinon une chaîne vide.

network→get_callbackMaxDelay()

Retourne l'attente maximale entre deux notifications par callback, en secondes.

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_errorMessage()
Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

network→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

network→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

network→get_hardwareId()
Retourne l'identifiant unique de la fonction.

network→get_ipAddress()
Retourne l'adresse IP utilisée par le module Yoctopuce.

network→get_logicalName()
Retourne le nom logique de l'interface réseau, qui correspond au nom réseau du module.

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_primaryDNS()
Retourne l'adresse IP du serveur de nom 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 nom 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→isOnline()
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

network→isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

network→load(msValidity)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

network→load_async(msValidity, callback, context)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

network→nextNetwork()
Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()`.

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_callbackMaxDelay(newval)
Modifie l'attente maximale entre deux notifications par callback, en secondes.

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_logicalName(newval)
Modifie le nom logique de l'interface réseau, qui correspond au nom réseau du module.

network→set_primaryDNS(newval)
Modifie l'adresse IP du serveur de nom 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→useDHCP(fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)
Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

network→useStaticIP(ipAddress, subnetMaskLen, router)
Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

yFindNetwork()

Permet de retrouver une interface réseau d'après un identifiant donné.

```
YNetwork* yFindNetwork( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YNetwork.isOnline()` pour tester si l'interface réseau est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :
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.

---

yFirstNetwork()

Commence l'énumération des interfaces réseau accessibles par la librairie.

```
YNetwork* yFirstNetwork( )
```

Utiliser la fonction `YNetwork.nextNetwork()` pour itérer sur les autres interfaces réseau.

Retourne :
un pointeur sur un objet `YNetwork`, correspondant à la première interface réseau accessible en ligne, ou `null` si il n'y a pas de interfaces réseau disponibles.

---

network→callbackLogin()

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()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

---

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()

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()

Retourne une version hashée du laisser-passer pour le callback de notification si 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 si il a été configuré, ou sinon une chaîne vide

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

network→get_callbackMaxDelay()

Retourne l'attente maximale entre deux notifications par callback, en secondes.

```
unsigned 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_callbackMinDelay()

Retourne l'attente minimale entre deux notifications par callback, en secondes.

```
unsigned 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()

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_errorMessage()

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

```
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 fonction

network→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

network→get_networkDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

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()

Retourne le nom logique de l'interface réseau, qui correspond au nom réseau du module.

```
string get_logicalName( )
```

Retourne :
une chaîne de caractères représentant le nom logique de l'interface réseau, qui correspond au nom réseau du module

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

---

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()

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`

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

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

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

Paramètres :

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

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

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

---

network→get_primaryDNS()

Retourne l'adresse IP du serveur de nom primaire que le module doit utiliser.

```
string get_primaryDNS( )
```

Retourne :
une chaîne de caractères représentant l'adresse IP du serveur de nom primaire que le module doit utiliser

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

---

network→get_readiness()

Retourne l'état de fonctionnement atteint par l'interface réseau.

```
Y_READINESS_enum get_readiness( )
```

Le niveau zéro (DOWN_0) signifie qu'aucun support réseau matériel. Soit il n'y a pas de signal sur le cable réseau, soit le point d'accès sans fil choisi n'est pas détecté. Le niveau 1 (LIVE_1) est atteint lorsque le réseau est détecté, mais n'est pas encore connecté. Pour un réseau sans fil, cela confirme la l'existence du SSID configuré. Le niveau 2 (LINK_2) est atteint lorsque le support matériel du réseau est fonctionnel. Pour une connection réseau filaire, le niveau 2 signifie que le cable est connecté aux deux bouts. Pour une connection à un point d'accès réseau sans fil, il démontre que les paramètres de sécurités configurés sont corrects. Pour une connection sans fil en mode ad-hoc, cela signifie qu'il y a au moins un partenaire sur le réseau ad-hoc. Le niveau 3 (DHCP_3) est atteint lorsque qu'une adresse IP a été obtenue par DHCP. Le niveau 4 (DNS_4) est atteint lorsqu'un serveur DNS est joignable par le réseau. Le niveau 5 (WWW_5) est atteint lorsque la connectivité globale à internet est avérée par l'obtention de l'heure courante sur une serveur NTP.

Retourne :
une valeur parmi `Y_READINESS_DOWN`, `Y_READINESS_EXISTS`, `Y_READINESS_LINKED`, `Y_READINESS_LAN_OK` et `Y_READINESS_WWW_OK` représentant l'état de fonctionnement atteint par l'interface réseau

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

---

network→get_router()

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()

Retourne l'adresse IP du serveur de nom secondaire que le module doit utiliser.

```
string get_secondaryDNS( )
```

Retourne :
une chaîne de caractères représentant l'adresse IP du serveur de nom secondaire que le module doit utiliser

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

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()

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

```
void * 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()

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→isOnline()

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

```
bool isOnline( )
```

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

Retourne :
`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :

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

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

Retourne :

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

---

network→load()

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

```
YRETCODE load( int msValidity)
```

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

Paramètres :

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

Retourne :

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

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

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

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

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

Retourne :

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

---

network→nextNetwork()

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→registerValueCallback()

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

```
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.

---

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( const 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()

Modifie le laisser-passer pour se connecter à l'adresse de callback.

```
int set_callbackCredentials( const 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_callbackMaxDelay()

Modifie l'attente maximale entre deux notifications par callback, en secondes.

```
int set_callbackMaxDelay( unsigned 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_callbackMinDelay()

Modifie l'attente minimale entre deux notifications par callback, en secondes.

```
int set_callbackMinDelay( unsigned 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()

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

```
int set_callbackUrl( const 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_logicalName()

Modifie le nom logique de l'interface réseau, qui correspond au nom réseau du module.

```
int set_logicalName( const 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, qui correspond au nom réseau 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_primaryDNS()

Modifie l'adresse IP du serveur de nom primaire que le module doit utiliser.

```
int set_primaryDNS( const 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 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()

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

```
int set_secondaryDNS( const 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()

Enregistre 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( void* 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()

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

```
int set_userPassword( const 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→useDHCP()

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()

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.

3.16. Interface de la fonction Pressure

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

Pour utiliser les fonctions décrites ici, vous devez inclure:
```
#include "yocto_pressure.h"
```

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 la fonction.

pressure→get_advertisedValue()
Retourne la valeur courante du capteur de pression (pas plus de 6 caractères).

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

pressure→get_currentValue()
Retourne la valeur mesurée actuelle.

pressure→get_errorMessage()
Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

pressure→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

pressure→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

pressure→get_hardwareId()
Retourne l'identifiant unique de la fonction.

pressure→get_highestValue()
Retourne la valeur maximale observée.

pressure→get_logicalName()
Retourne le nom logique du capteur de pression.

pressure→get_lowestValue()
Retourne la valeur minimale observée.

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_resolution()
Retourne la résolution des valeurs mesurées.

pressure→get_unit()
Retourne l'unité dans laquelle la valeur mesurée 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 la fonction est joignable, sans déclencher d'erreur.

pressure→isOnline_async(callback, context)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

pressure→load(msValidity)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

pressure→load_async(msValidity, callback, context)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

pressure→nextPressure()
Continue l'énumération des capteurs de pression commencée à l'aide de `yFirstPressure()`.

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_logicalName(newval)
Modifie le nom logique du capteur de pression.

pressure→set_lowestValue(newval)
Modifie la mémoire de valeur minimale observée.

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`.

yFindPressure()

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

```
YPressure* yFindPressure( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPressure.isOnline()` pour tester si le capteur de pression est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :
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.

yFirstPressure()

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

```
YPressure* yFirstPressure( )
```

Utiliser la fonction `YPressure.nextPressure()` pour itérer sur les autres capteurs de pression.

Retourne :
un pointeur sur un objet `YPressure`, correspondant à le premier capteur de pression accessible en ligne, ou `null` si il n'y a pas de capteurs de pression disponibles.

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( floatArr rawValues,
                                              floatArr 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()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

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()

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`.

pressure→get_currentValue()

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`.

pressure→get_errorMessage()

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

```
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 fonction

pressure→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

pressure→get_pressureDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

pressure→get_highestValue()

Retourne la valeur maximale observée.

```
double get_highestValue( )
```

Retourne :
une valeur numérique représentant la valeur maximale observée

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

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()

Retourne la valeur minimale observée.

```
double get_lowestValue( )
```

Retourne :
une valeur numérique représentant la valeur minimale observée

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

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`

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

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

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

Paramètres :

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

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

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

---

pressure→get_resolution()

Retourne la résolution des valeurs mesurées.

```
double get_resolution( )
```

La résolution correspond à la précision de la représentation numérique 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()

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

```
string get_unit( )
```

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la valeur mesurée est exprimée

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

---

pressure→get_userData()

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

```
void * 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()

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

```
bool isOnline( )
```

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

Retourne :
`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :
callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen
context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

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

`pressure→load()`

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

```
YRETCODE load( int msValidity)
```

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

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

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

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

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

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

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

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

---

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→registerValueCallback()

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

```
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.

---

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_logicalName()

Modifie le nom logique du capteur de pression.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

pressure→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.

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( void* 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

3.17. 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 (relais inverseur), les deux sorties sont appelées A et B, la sortie A correspondant a la position de repos (hors tension) et la sortie B correspondant à l'état actif. Si vous préféreriez l'état par défaut opposé, vous pouvez simplement changer vos fils sur le bornier.

Pour utiliser les fonctions décrites ici, vous devez inclure:
```
#include "yocto_relay.h"
```

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→describe()
Retourne un court texte décrivant la fonction.

relay→get_advertisedValue()
Retourne la valeur courante du relais (pas plus de 6 caractères).

relay→get_errorMessage()

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

relay→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

relay→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

relay→get_hardwareId()
Retourne l'identifiant unique de la fonction.

relay→get_logicalName()
Retourne le nom logique du relais.

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_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 la fonction est joignable, sans déclencher d'erreur.

relay→isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

relay→load(**msValidity**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

relay→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

relay→nextRelay()
Continue l'énumération des relais commencée à l'aide de `yFirstRelay()`.

relay→pulse(**ms_duration**)
Commute le relais à l'état B (actif) pour un durée spécifiée, puis revient ensuite spontanément vers l'état A (état de repos).

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

relay→set_logicalName(**newval**)
Modifie le nom logique du relais.

relay→set_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_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`.

yFindRelay()

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

```
YRelay* yFindRelay( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YRelay.isOnline()` pour tester si le relais est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :
func une chaîne de caractères qui référence le relais sans ambiguïté

Retourne :
un objet de classe `YRelay` qui permet ensuite de contrôler le relais.

yFirstRelay()

Commence l'énumération des relais accessibles par la librairie.

```
YRelay* yFirstRelay( )
```

Utiliser la fonction `YRelay.nextRelay()` pour itérer sur les autres relais.

Retourne :
un pointeur sur un objet `YRelay`, correspondant à le premier relais accessible en ligne, ou `null` si il n'y a pas de relais disponibles.

relay→describe()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

relay→get_advertisedValue()

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_errorMessage()

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

```
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 fonction

relay→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

relay→get_relayDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

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_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`

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

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

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

Paramètres :

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

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

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

---

relay→get_output()

Retourne l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

```
Y_OUTPUT_enum 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()

Retourne le nombre de millisecondes restantes avant le retour à la position de repos (état A), durant la génération d'une impulsion mesurée.

```
unsigned 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()

Retourne l'état du relais (A pour la position de repos, B pour l'état actif).

```
Y_STATE_enum 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_userData()

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

```
void * get_userData( )
```

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

Retourne :
l'objet stocké précédemment par l'appelant.

relay→isOnline()

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

```
bool isOnline( )
```

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

Retourne :
`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :

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

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

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

relay→load()

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

```
YRETCODE load( int msValidity)
```

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

Paramètres :

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

Retourne :

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

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

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

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

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

Retourne :

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

---

relay→nextRelay()

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()

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()

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

```
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.

---

relay→set_logicalName()

Modifie le nom logique du relais.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

---

relay→set_output()

Modifie l'état de la sortie du relais, lorsqu'il est utilisé comme un simple interrupteur.

```
int set_output( Y_OUTPUT_enum 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()

Modifie l'état du relais (A pour la position de repos, B pour l'état actif).

```
int set_state( Y_STATE_enum 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_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( void* 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

3.18. Interface de la fonction Servo

La librairie de programmation Yoctopuce permet non seulement de déplacer le servo vers une position donnée, mais aussi de spécifier l'interval de temps dans lequel le mouvement doit être fait, de sorte à pouvoir synchroniser un mouvement sur plusieurs servos.

Pour utiliser les fonctions décrites ici, vous devez inclure:
```
#include "yocto_servo.h"
```

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 la fonction.

servo→get_advertisedValue()
Retourne la valeur courante du servo (pas plus de 6 caractères).

servo→get_errorMessage()
Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

servo→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

servo→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

servo→get_hardwareId()
Retourne l'identifiant unique de la fonction.

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_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 la fonction est joignable, sans déclencher d'erreur.

servo→isOnline_async(callback, context)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

servo→load(msValidity)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

servo→load_async(msValidity, callback, context)
Met en cache les valeurs courantes de la fonction, 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_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_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`.

yFindServo()

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

```
YServo* yFindServo( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YServo.isOnline()` pour tester si le servo est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :
func une chaîne de caractères qui référence le servo sans ambiguïté

Retourne :
un objet de classe `YServo` qui permet ensuite de contrôler le servo.

---

yFirstServo()

Commence l'énumération des servo accessibles par la librairie.

```
YServo* yFirstServo( )
```

Utiliser la fonction `YServo.nextServo()` pour itérer sur les autres servo.

Retourne :
un pointeur sur un objet `YServo`, correspondant à le premier servo accessible en ligne, ou `null` si il n'y a pas de servo disponibles.

---

servo→describe()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

---

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_errorMessage()

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

```
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 fonction

servo→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

servo→get_servoDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

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()

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`

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

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

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

Paramètres :

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

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

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

---

servo→get_neutral()

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()

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_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()

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

```
void * get_userData( )
```

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

Retourne :
l'objet stocké précédemment par l'appelant.

---

servo→isOnline()

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

```
bool isOnline( )
```

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

Retourne :
`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :
callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen
context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

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

---

servo→load()

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

```
YRETCODE load( int msValidity)
```

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

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

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

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

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

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

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

Retourne :

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

---

servo→move()

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()

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()

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

```
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.

---

servo→set_logicalName()

Modifie le nom logique du servo.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

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()

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_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()`

Enregistre 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( void* 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

3.19. Interface de la fonction Temperature

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

Pour utiliser les fonctions décrites ici, vous devez inclure:
```
#include "yocto_temperature.h"
```

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 la fonction.

temperature→get_advertisedValue()
Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

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

temperature→get_currentValue()
Retourne la valeur mesurée actuelle.

temperature→get_errorMessage()
Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

temperature→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

temperature→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

temperature→get_hardwareId()

Retourne l'identifiant unique de la fonction.

temperature→get_highestValue()
Retourne la valeur maximale observée.

temperature→get_logicalName()
Retourne le nom logique du capteur de température.

temperature→get_lowestValue()
Retourne la valeur minimale observée.

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_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 valeur mesurée 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 la fonction est joignable, sans déclencher d'erreur.

temperature→isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

temperature→load(**msValidity**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

temperature→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

temperature→nextTemperature()
Continue l'énumération des capteurs de température commencée à l'aide de `yFirstTemperature()`.

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_logicalName(**newval**)
Modifie le nom logique du capteur de température.

temperature→set_lowestValue(**newval**)
Modifie la mémoire de valeur minimale observée.

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`.

yFindTemperature()

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

```
YTemperature* yFindTemperature( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YTemperature.isOnline()` pour tester si le capteur de température est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

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.

yFirstTemperature()

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

```
YTemperature* yFirstTemperature( )
```

Utiliser la fonction `YTemperature.nextTemperature()` pour itérer sur les autres capteurs de température.

Retourne :

un pointeur sur un objet `YTemperature`, correspondant à le premier capteur de température accessible en ligne, ou `null` si il n'y a pas de capteurs de température disponibles.

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( floatArr rawValues,
                                         floatArr 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()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

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()

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`.

temperature→get_currentValue()

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`.

temperature→get_errorMessage()

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

```
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 fonction

temperature→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

temperature→get_temperatureDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

temperature→get_highestValue()

Retourne la valeur maximale observée.

```
double get_highestValue( )
```

Retourne :
une valeur numérique représentant la valeur maximale observée

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

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()

Retourne la valeur minimale observée.

```
double get_lowestValue( )
```

Retourne :
une valeur numérique représentant la valeur minimale observée

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

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`

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

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

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

Paramètres :

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

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

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

temperature→get_resolution()

Retourne la résolution des valeurs mesurées.

```
double get_resolution( )
```

La résolution correspond à la précision de la représentation numérique 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()

Retourne le type de capteur de température utilisé par le module

```
Y_SENSORTYPE_enum 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 et Y_SENSORTYPE_TYPE_T 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()

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

```
string get_unit( )
```

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la valeur mesurée est exprimée

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

---

temperature→get_userData()

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

```
void * get_userData( )
```

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

Retourne :
l'objet stocké précédemment par l'appelant.

---

temperature→isOnline()

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

```
bool isOnline( )
```

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

Retourne :
true si la fonction est joignable, false sinon

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

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

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

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

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

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

---

temperature→load()

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

```
YRETCODE load( int msValidity)
```

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

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

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

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

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

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

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

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

---

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→registerValueCallback()

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

```
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.

---

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_logicalName()

Modifie le nom logique du capteur de température.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

---

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_sensorType()

Change le type de senseur utilisé par le module.

```
int set_sensorType( Y_SENSORTYPE_enum newval)
```

Cette function sert à spécifier le type de thermocouple (K,E, etc..) raccordé au module. Cette fonction n'aura pas d'effet si le module utilise un capteur digital. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur parmi `Y_SENSORTYPE_DIGITAL`, `Y_SENSORTYPE_TYPE_K`, `Y_SENSORTYPE_TYPE_E`, `Y_SENSORTYPE_TYPE_J`, `Y_SENSORTYPE_TYPE_N`, `Y_SENSORTYPE_TYPE_R`, `Y_SENSORTYPE_TYPE_S` et `Y_SENSORTYPE_TYPE_T`

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()`

Enregistre 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( void* 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

3.20. Interface de la fonction Voltage

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
#include "yocto_voltage.h"
```

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 la fonction.

voltage→get_advertisedValue()
Retourne la valeur courante du capteur de tension (pas plus de 6 caractères).

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

voltage→get_currentValue()
Retourne la valeur mesurée actuelle.

voltage→get_errorMessage()
Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

voltage→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

voltage→get_functionDescriptor()
Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

voltage→get_hardwareId()
Retourne l'identifiant unique de la fonction.

voltage→get_highestValue()
Retourne la valeur maximale observée.

voltage→get_logicalName()
Retourne le nom logique du capteur de tension.

voltage→get_lowestValue()
Retourne la valeur minimale observée.

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_resolution()
Retourne la résolution des valeurs mesurées.

voltage→get_unit()
Retourne l'unité dans laquelle la valeur mesurée 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 la fonction est joignable, sans déclencher d'erreur.

voltage→isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

voltage→load(**msValidity**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

voltage→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

voltage→nextVoltage()
Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstVoltage()`.

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_logicalName(**newval**)
Modifie le nom logique du capteur de tension.

voltage→set_lowestValue(**newval**)
Modifie la mémoire de valeur minimale observée.

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`.

yFindVoltage()

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

```
YVoltage* yFindVoltage( const 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 `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.

yFirstVoltage()

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

```
YVoltage* yFirstVoltage( )
```

Utiliser la fonction `YVoltage.nextVoltage()` pour itérer sur les autres capteurs de tension.

Retourne :

un pointeur sur un objet `YVoltage`, correspondant à le premier capteur de tension accessible en ligne, ou `null` si il n'y a pas de capteurs de tension disponibles.

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( floatArr rawValues,
                                         floatArr 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()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

---

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()

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`.

---

voltage→get_currentValue()

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`.

---

voltage→get_errorMessage()

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

```
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 fonction

---

voltage→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

---

voltage→get_voltageDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne Y_HARDWAREID_INVALID.

---

voltage→get_highestValue()

Retourne la valeur maximale observée.

```
double get_highestValue( )
```

Retourne :
une valeur numérique représentant la valeur maximale observée

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

---

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()

Retourne la valeur minimale observée.

```
double get_lowestValue( )
```

Retourne :
une valeur numérique représentant la valeur minimale observée

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

---

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`

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

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

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

Paramètres :

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

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

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

---

voltage→get_resolution()

Retourne la résolution des valeurs mesurées.

```
double get_resolution( )
```

La résolution correspond à la précision de la représentation numérique 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()

Retourne l'unité dans laquelle la valeur mesurée est exprimée.

```
string get_unit( )
```

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la valeur mesurée est exprimée

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

voltage→get_userData()

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

```
void * 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()

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

```
bool isOnline( )
```

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

Retourne :
`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :

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

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

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

voltage→load()

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

```
YRETCODE load( int msValidity)
```

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

Paramètres :

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

Retourne :

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

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

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

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

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

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

Retourne :

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

---

voltage→nextVoltage()

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→registerValueCallback()

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

```
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.

---

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_logicalName()

Modifie le nom logique du capteur de tension.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

---

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_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( void* 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 Source de tension

La librairie de programmation Yoctopuce permet de commande la tension de srotir du module. Vous pouvez affecter une valeur fixe,ou faire des transition de voltage.

Pour utiliser les fonctions décrites ici, vous devez inclure:
```
#include "yocto_vsource.h"
```

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.

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

vsource→get_hardwareId()
Retourne l'identifiant unique de la fonction.

vsource→get_logicalName()
Retourne le nom logique de la source de tension.

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

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

vsource→get_overCurrent()
Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

vsource→get_overHeat()
Rend TRUE si le module est en surchauffe.

vsource→get_overLoad()
Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

vsource→get_regulationFailure()

Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

vsource→get_unit()
Retourne l'unité dans laquelle la tension est exprimée.

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

vsource→get_voltage()
Retourne la valeur de la commande de tension de sortie en mV

vsource→isOnline()
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

vsource→isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

vsource→load(**msValidity**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

vsource→load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

vsource→nextVSource()
Continue l'énumération des sources de tension commencée à l'aide de `yFirstVSource()`.

vsource→pulse(**voltage**, **ms_duration**)
Active la sortie à une tension donnée, et pour durée spécifiée, puis revient ensuite spontanément à zéro volt.

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

vsource→reset()
Réinitilialise la sortie du module.

vsource→set_logicalName(**newval**)
Modifie le nom logique de la source de tension.

vsource→set_userData(**data**)
Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

vsource→set_voltage(**newval**)
Règle la tension de sortie du module (en milliVolts).

vsource→voltageMove(**target**, **ms_duration**)
Déclenche une variation constante de la sortie vers une valeur donnée.

yFindVSource()

Permet de retrouver une source de tension d'après un identifiant donné.

```
YVSource* yFindVSource( const 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 ou 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()

Commence l'énumération des sources de tension accessibles par la librairie.

```
YVSource* yFirstVSource( )
```

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()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

---

vsource→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`.

---

vsource→get_errorMessage()

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

```
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 fonction

vsource→get_errorType()

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

```
YRETCODE get_errorType( )
```

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

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

vsource→get_extPowerFailure()

Rend TRUE si le voltage de l'alimentation externe est trop bas.

```
Y_EXTPOWERFAILURE_enum get_extPowerFailure( )
```

Retourne :
soit `Y_EXTPOWERFAILURE_FALSE`, soit `Y_EXTPOWERFAILURE_TRUE`

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

vsource→get_failure()

Indique si le module est en condition d'erreur.

```
Y_FAILURE_enum get_failure( )
```

Il possible de savoir de quelle erreur il s'agit en testant get_overheat, get_overcurrent etc... Lorsqu'un condition d'erreur est rencontée, la tension de sortie est mise à zéro est ne peut pas être changée tant la fonction reset() n'aura pas appellée.

Retourne :
soit `Y_FAILURE_FALSE`, soit `Y_FAILURE_TRUE`

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

vsource→get_vsourceDescriptor()

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

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

vsource→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`.

---

vsource→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`

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

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

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

Paramètres :

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

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

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

---

vsource→get_overCurrent()

Rend TRUE si l'appareil connecté à la sortie du module consomme trop de courant.

```
Y_OVERCURRENT_enum get_overCurrent( )
```

Retourne :
soit `Y_OVERCURRENT_FALSE`, soit `Y_OVERCURRENT_TRUE`

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

---

vsource→get_overHeat()

Rend TRUE si le module est en surchauffe.

```
Y_OVERHEAT_enum 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()

Rend TRUE si le module n'est pas capable de tenir la tension de sortie demandée.

```
Y_OVERLOAD_enum get_overLoad( )
```

Retourne :
soit Y_OVERLOAD_FALSE, soit Y_OVERLOAD_TRUE

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

vsource→get_regulationFailure()

Rend TRUE si le voltage de sortie de trop élevé par report à la tension demandée demandée.

```
Y_REGULATIONFAILURE_enum 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()

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.

vsource→get_userData()

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

```
void * 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()

Retourne la valeur de la commande de tension de sortie en mV

```
int get_voltage( )
```

Retourne :
un entier représentant la valeur de la commande de tension de sortie en mV

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

---

vsource→isOnline()

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

```
bool isOnline( )
```

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

Retourne :
`true` si la fonction est joignable, `false` sinon

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

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

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

Paramètres :
callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen
context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

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

---

vsource→load()

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

```
YRETCODE load( int msValidity)
```

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

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

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

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

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

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

Paramètres :

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

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

---

vsource→nextVSource()

Continue l'énumération des sources de tension commencée à l'aide de `yFirstVSource()`.

```
YVSource * nextVSource( )
```

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

---

vsource→pulse()

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)
```

Paramètres :

voltage	tension demandée, en millivolts
ms_duration	durée de l'impulsion, en millisecondes

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

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

---

vsource→registerValueCallback()

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

```
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→reset()

Réinitialise la sortie du module.

```
int reset( )
```

Cette fonction doit être appelée après une condition d'erreur. Après toute condition d'erreur, le voltage de sortie est mis à zéro et ne peut pas être changé tant que cette fonction n'aura pas été appelée.

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

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

vsource→set_logicalName()

Modifie le nom logique de la source de tension.

```
int set_logicalName( const 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'opération se déroule sans erreur.

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

vsource→set_userData()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

```
void set_userData( void* 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()

Règle la tension de sortie du module (en milliVolts).

```
int set_voltage( int newval)
```

Paramètres :
newval un entier

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

vsource→voltageMove()

Déclenche une variation constante de la sortie vers une valeur donnée.

```
int voltageMove( int target, int 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.

3.22. Interface de la fonction Wireless

Pour utiliser les fonctions décrites ici, vous devez inclure:
```
#include "yocto_wireless.h"
```

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 la fonction.

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.

wireless→get_errorMessage()
Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

wireless→get_errorType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

wireless→get_functionDescriptor()
Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

wireless→get_hardwareId()
Retourne l'identifiant unique de la fonction.

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_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 la fonction est joignable, sans déclencher d'erreur.

wireless→isOnline_async(callback, context)
Vérifie si le module hébergeant la fonction 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 la fonction, avec une durée de validité spécifiée.

wireless→load_async(msValidity, callback, context)
Met en cache les valeurs courantes de la fonction, 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`.

yFindWireless()

Permet de retrouver une interface réseau sans fil d'après un identifiant donné.

```
YWireless* yFindWireless( const 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 ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YWireless.isOnline()` pour tester si l'interface réseau sans fil est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la

première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

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.

---

yFirstWireless()

Commence l'énumération des interfaces réseau sans fil accessibles par la librairie.

```
YWireless* yFirstWireless( )
```

Utiliser la fonction `YWireless.nextWireless()` pour itérer sur les autres interfaces réseau sans fil.

Retourne :

un pointeur sur un objet `YWireless`, correspondant à la première interface réseau sans fil accessible en ligne, ou `null` si il n'y a pas de interfaces réseau sans fil disponibles.

---

wireless→adhocNetwork()

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)
```

Si une clef d'accès est spécifiée, le réseau sera protégé par une sécurité WEP128 (l'utilisation de WPA n'est pas standardisée en mode ad-hoc). N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

Paramètres :

ssid nom du réseau sans fil à créer

securityKey clé d'accès de réseau, sous forme de chaîne de caractères

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

wireless→describe()

Retourne un court texte décrivant la fonction.

```
string describe( )
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :

une chaîne de caractères décrivant la fonction

---

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()

Retourne le numéro du canal 802.

```
unsigned get_channel( )
```

11 utilisé, ou 0 si le réseau sélectionné n'a pas été trouvé.

Retourne :
un entier représentant le numéro du canal 802

En cas d'erreur, déclenche une exception ou retourne Y_CHANNEL_INVALID.

wireless→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

```
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 fonction

wireless→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

```
YRETCODE get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :
un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la fonction

wireless→get_wirelessDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

```
YFUN_DESCR 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_hardwareId()

Retourne l'identifiant unique de la fonction.

```
string get_hardwareId( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction.

Retourne :
une chaîne de caractères identifiant la fonction En cas d'erreur, déclenche une exception ou retourne `Y_HARDWAREID_INVALID`.

---

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()

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_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`

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :
rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

wireless→get_security()

Retourne l'algorithme de sécurité utilisé par le réseau sans-fil sélectionné.

```
Y_SECURITY_enum 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()

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()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

```
void * get_userData( )
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :
l'objet stocké précédemment par l'appelant.

wireless→isOnline()

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

```
bool isOnline( )
```

Si les valeurs des attributs en cache de la fonction sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si la fonction est joignable, `false` sinon

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

Si les valeurs des attributs en cache de la fonction sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :
rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

wireless→joinNetwork()

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()

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

```
YRETCODE load( int msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :
rien du tout : le résultat sera passé en paramètre à la fonction de callback.

---

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()

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
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.

---

wireless→set_logicalName()

Modifie le nom logique de l'interface réseau sans fil.

```
int set_logicalName( const 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'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

---

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( void* 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

4. Index