Yocto-Motor-DC

Mode d'emploi

Table des matières

1. Introduction **1**

1.1. Informations de sécurité 2

1.2. Conditions environnementales 3

2. Présentation **5**

2.1. Les éléments communs 5

2.2. Les éléments spécifiques 6

2.3. Isolation électrique fonctionnelle 7

2.4. Accessoires optionnels 8

3. Fonctionnement du Yocto-Motor-DC **11**

3.1. Principe 11

3.2. Fréquence 11

3.3. Fonctionnalités 12

3.4. Protections 13

4. Premiers pas **15**

4.1. Prérequis 15

4.2. Test de la connectivité USB 17

4.3. Localisation 17

4.4. Test du module 17

4.5. Configuration 18

5. Montage et connectique **21**

5.1. Chaleur 21

5.2. Fixation 22

5.3. Contraintes d'alimentation par USB 23

6. Programmation, concepts généraux **25**

6.1. Paradigme de programmation 25

6.2. Le module Yocto-Motor-DC 27

6.3. Module 28

6.4. Motor 30

6.5. Current 31

6.6. Voltage ... 32
6.7. Temperature ... 33
6.8. Quelle interface: Native, DLL ou Service? ... 34
6.9. Programmation, par où commencer? ... 37

7. Utilisation du Yocto-Motor-DC en ligne de commande ... 39
7.1. Installation ... 39
7.2. Utilisation: description générale ... 39
7.3. Contrôle de la fonction Motor ... 40
7.4. Contrôle de la partie module ... 41
7.5. Limitations ... 41

8. Utilisation du Yocto-Motor-DC en Python ... 43
8.1. Fichiers sources ... 43
8.2. Librairie dynamique ... 43
8.3. Contrôle de la fonction Motor ... 43
8.4. Contrôle de la partie module ... 45
8.5. Gestion des erreurs ... 47

9. Utilisation du Yocto-Motor-DC en C++ ... 49
9.1. Contrôle de la fonction Motor ... 49
9.2. Contrôle de la partie module ... 52
9.3. Gestion des erreurs ... 54
9.4. Intégration de la librairie Yoctopuce en C++ ... 55

10. Utilisation du Yocto-Motor-DC en C# ... 57
10.1. Installation ... 57
10.2. Utilisation l'API yoctopuce dans un projet Visual C# ... 57
10.3. Contrôle de la fonction Motor ... 58
10.4. Contrôle de la partie module ... 60
10.5. Gestion des erreurs ... 62

11. Utilisation du Yocto-Motor-DC avec LabVIEW ... 65
11.1. Architecture ... 65
11.2. Compatibilité ... 66
11.3. Installation ... 66
11.4. Présentation des VIs Yoctopuce ... 71
11.5. Fonctionnement et utilisation des VIs ... 74
11.6. Utilisation des objets ... 76
11.7. Gestion du datalogger ... 78
11.8. Énumération de fonctions ... 79
11.9. Un mot sur les performances ... 80
11.10. Un exemple complet de programme LabVIEW ... 80
11.11. Différences avec les autres API Yoctopuce ... 81

12. Utilisation du Yocto-Motor-DC en Java ... 83
12.1. Préparation ... 83
12.2. Contrôle de la fonction Motor ... 83
12.3. Contrôle de la partie module ... 85
12.4. Gestion des erreurs ... 88

13. Utilisation du Yocto-Motor-DC avec Android ... 89

13.1. Accès Natif et Virtual Hub. ... 89
13.2. Préparation ... 89
13.3. Compatibilité ... 89
13.4. Activer le port USB sous Android ... 90
13.5. Contrôle de la fonction Motor ... 92
13.6. Contrôle de la partie module ... 95
13.7. Gestion des erreurs ... 99

14. Utilisation du Yocto-Motor-DC en TypeScript ... 101
14.1. Utiliser la librairie Yoctopuce pour TypeScript ... 102
14.2. Petit rappel sur les fonctions asynchrones en JavaScript ... 102
14.3. Contrôle de la fonction Motor ... 103
14.4. Contrôle de la partie module ... 106
14.5. Gestion des erreurs ... 108

15. Utilisation du Yocto-Motor-DC en JavaScript / EcmaScript ... 111
15.1. Fonctions bloquantes et fonctions asynchrones en JavaScript ... 112
15.2. Utiliser la librairie Yoctopuce pour JavaScript / EcmaScript 2017 ... 113
15.3. Contrôle de la fonction Motor ... 115
15.4. Contrôle de la partie module ... 119
15.5. Gestion des erreurs ... 121

16. Utilisation du Yocto-Motor-DC en PHP ... 123
16.1. Préparation ... 123
16.2. Contrôle de la fonction Motor ... 123
16.3. Contrôle de la partie module ... 125
16.4. API par callback HTTP et filtres NAT ... 128
16.5. Gestion des erreurs ... 131

17. Utilisation du Yocto-Motor-DC en VisualBasic .NET ... 133
17.1. Installation ... 133
17.2. Utilisation l'API yoctopuce dans un projet Visual Basic ... 133
17.3. Contrôle de la fonction Motor ... 134
17.4. Contrôle de la partie module ... 136
17.5. Gestion des erreurs ... 138

18. Utilisation du Yocto-Motor-DC en Delphi ... 141
18.1. Préparation ... 141
18.2. Contrôle de la fonction Motor ... 141
18.3. Contrôle de la partie module ... 144
18.4. Gestion des erreurs ... 146

19. Utilisation du Yocto-Motor-DC avec Universal Windows Platform ... 149
19.1. Fonctions bloquantes et fonctions asynchrones ... 149
19.2. Installation ... 150
19.3. Utilisation l'API Yoctopuce dans un projet Visual Studio ... 150
19.4. Contrôle de la fonction Motor ... 151
19.5. Un exemple concret ... 152
19.6. Contrôle de la partie module ... 153
19.7. Gestion des erreurs ... 155

20. Utilisation du Yocto-Motor-DC en Objective-C ... 157

20.1. Contrôle de la fonction Motor ... 157
20.2. Contrôle de la partie module ... 160
20.3. Gestion des erreurs ... 162

21. Utilisation avec des langages non supportés ... 165
21.1. Utilisation en ligne de commande ... 165
21.2. Assembly .NET ... 165
21.3. Virtual Hub et HTTP GET ... 167
21.4. Utilisation des librairies dynamiques ... 169
21.5. Port de la librairie haut niveau ... 172

22. Programmation avancée ... 173
22.1. Programmation par événements ... 173

23. Mise à jour du firmware ... 175
23.1. Le VirtualHub ou le YoctoHub ... 175
23.2. La librairie ligne de commandes ... 175
23.3. L'application Android Yocto-Firmware ... 175
23.4. La librairie de programmation ... 176
23.5. Le mode "mise à jour" ... 178

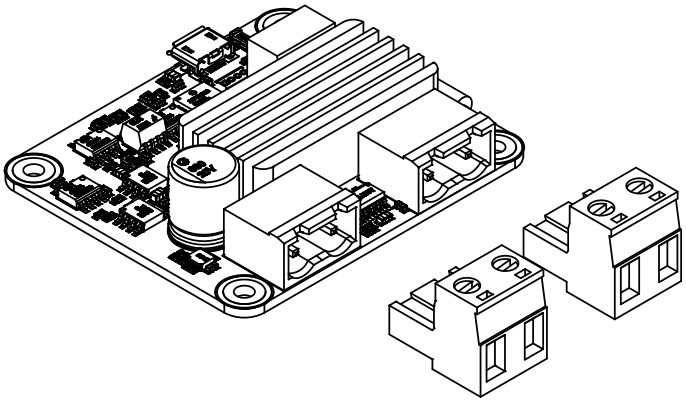
24. Référence de l'API de haut niveau ... 179
24.1. La classe YAPI ... 180
24.2. La classe YModule ... 221
24.3. La classe YMotor ... 298
24.4. La classe YVoltage ... 366
24.5. La classe YCurrent ... 440
24.6. La classe YTemperature ... 514

25. Problèmes courants ... 597
25.1. Par où commencer ? ... 597
25.2. Linux et USB ... 597
25.3. Plateformes ARM: HF et EL ... 598
25.4. Les exemples de programmation n'ont pas l'air de marcher ... 598
25.5. Module alimenté mais invisible pour l'OS ... 598
25.6. Another process named xxx is already using yAPI ... 598
25.7. Déconnexions, comportement erratique ... 599
25.8. RegisterHub d'un VirtualHub déconnecte le précédent ... 599
25.9. Commandes ignorées ... 599
25.10. Module endommagé ... 599

26. Caractéristiques ... 601

1. Introduction

Le Yocto-Motor-DC est un module électronique de 58x50mm qui est essentiellement destiné à contrôler un moteur à balais, de 3 à 18V et jusqu'à 20 ampères en continue (30A en pointe). Mais il peut aussi être utilisé pour piloter l'intensité d'un éclairage à basse tension, ou de tout autre appareil supportant d'être alimenté avec du courant haché. C'est un module isolé: la partie USB est électriquement isolé de la partie puissance.



Le module Yocto-Motor-DC

Le Yocto-Motor-DC n'est pas en lui-même un produit complet. C'est un composant destiné à être intégré dans une solution d'automatisation en laboratoire, ou pour le contrôle de procédés industriels, ou pour des applications similaires en milieu résidentiel ou commercial. Pour pouvoir l'utiliser, il faut au minimum l'installer à l'intérieur d'un boîtier de protection et le raccorder à un ordinateur de contrôle.

Yoctopuce vous remercie d'avoir fait l'acquisition de ce Yocto-Motor-DC et espère sincèrement qu'il vous donnera entière satisfaction. Les ingénieurs Yoctopuce se sont donné beaucoup de mal pour que votre Yocto-Motor-DC soit facile à installer n'importe où et soit facile à piloter depuis un maximum de langages de programmation. Néanmoins, si ce module venait à vous décevoir, ou si vous avez besoin d'informations supplémentaires, n'hésitez pas à contacter Yoctopuce:

Adresse e-mail:	support@yoctopuce.com
Site Internet:	www.yoctopuce.com
Adresse postale:	Chemin des Journaliers, 1
Localité:	1236 Cartigny
Pays:	Suisse

1.1. Informations de sécurité

Le Yocto-Motor-DC est conçu pour respecter la norme de sécurité IEC 61010-1:2010. Il ne causera pas de danger majeur pour l'opérateur et la zone environnante, même en condition de premier défaut, pour autant qu'il soit intégré et utilisé conformément aux instructions contenues dans cette documentation, et en particulier dans cette section.

Boîtier de protection

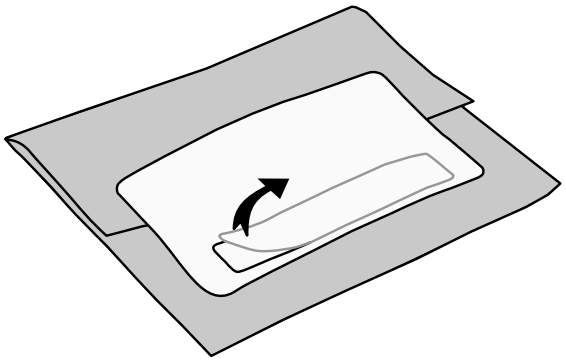
Le Yocto-Motor-DC ne doit pas être utilisé sans boîtier de protection, en raison des composants électriques à nu. Pour une sécurité optimale, il devrait être mis dans un boîtier non métallique, non-inflammable, résistant à un choc de 5 J, par exemple en polycarbonate (LEXAN ou autre) d'indice de protection IK08 et classifié V-1 ou mieux selon la norme IEC 60695-11-10. L'utilisation d'un boîtier de qualité inférieure peut nécessiter des avertissements spécifiques pour l'utilisateur et/ou compromettre la conformité avec la norme de sécurité.

Entretien

Si un dégat est constaté sur le circuit électronique ou sur le boîtier, il doit être remplacé afin de ne pas compromettre la sécurité d'utilisation et d'éviter d'endommager d'autres parties du système par les surcharges éventuelles que pourrait causer un court-circuit.

Identification

Pour faciliter l'entretien du circuit et l'identification des risques lors de la maintenance, vous devriez coller l'étiquette autocollante synthétique identifiant le Yocto-Motor-DC, fournie avec le circuit électronique, à proximité immédiate du module. Si le module est dans un boîtier dédié, l'étiquette devrait être collée sur la surface extérieur du boîtier. L'étiquette est résistante à l'humidité et au frottement usuel qui peut survenir durant un entretien normal.



L'étiquette d'identification est intégrée à l'étiquette de l'emballage.

Applications

La norme de sécurité vérifiée correspond aux instruments de laboratoire, pour le contrôle de procédés industriels, ou pour des applications similaires en milieu résidentiel ou commercial. Si vous comptez l'utiliser le Yocto-Motor-DC pour un autre type d'applications, vous devrez vérifier les critères de conformité en fonction de la norme applicable à votre application.

En particulier, le Yocto-Motor-DC n'est *pas* certifié pour utilisation dans un environnement médical, ni pour les applications critiques à la santé, ni pour toute autre application menaçant la vie humaine.

Environnement

Le Yocto-Motor-DC n'est *pas* certifié pour utilisation dans les zones dangereuses, ni pour les environnements explosifs. Les conditions environnementales assignées sont décrites ci-dessous.

Risque de brûlure et de feu

Lorsque de forts courants traversent le module, le circuit imprimé ainsi que certains composants peuvent chauffer à plus de 60°C. Un contact direct avec la peau produirait une brûlure, et une utilisation dans un boîtier inadéquat créerait un risque de feu non négligeable. Lors d'une exposition continue à de forts courants, pensez donc à prévenir tant les risques de brûlure que les risques d'incendie. Un système de ventilation aide à réduire la température, mais pour que le système reste sans danger même en cas de panne du ventilateur, il est essentiel d'utiliser un boîtier classé V-1 ou V-0 selon la norme IEC 60695-11-10. Assurez-vous que les aérations du boîtier soient orientés latéralement ou vers le haut, mais pas vers le bas.

Si vous utilisez de forts courants, utilisez du câble électrique spécifié pour supporter les hautes températures, car la chaleur se propagera à travers le bornier sur le câble électrique lui-même.

Classe de protection III (IEC 61140)

Le module Yocto-Motor-DC a été conçu pour travailler uniquement avec des très basses tension de sécurité. Ne dépassez pas les tensions indiquées dans ce manuel, et ne raccordez en aucun cas sur le bornier du Yocto-Motor-DC un fil susceptible d'être connecté au réseau secteur.

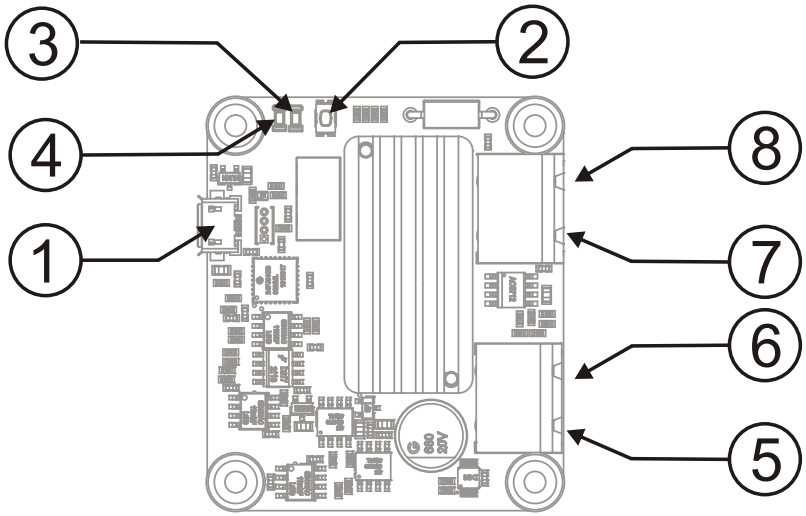
1.2. Conditions environnementales

Les produits Yoctopuce sont conçus pour une utilisation intérieure dans un environnement usuel de bureau ou de laboratoire (*degré de pollution 2* selon IEC 60664): la pollution de l'air doit être faible et essentiellement non conductrice. L'humidité relative prévue est de 10% à 90% RH, sans condensation. L'utilisation dans un environnement avec une pollution solide ou conductrice significative exige de protéger le module contre cette pollution par un boîtier certifié IP67 ou IP68. Les produits sont conçus pour une utilisation jusqu'à une altitude de 2000m.

Le fonctionnement de tous les modules Yoctopuce est garanti conforme à la documentation et aux spécifications de précision pour des conditions de température ambiante normales selon IEC61010-1, soit 5°C à 40°C. De plus, la plupart des modules peuvent aussi être utilisés sur une plage de température étendue, à laquelle quelques limitations peuvent s'appliquer selon les cas.

La plage de température de fonctionnement étendue du Yocto-Motor-DC est -30...70°C. Cette plage de température a été déterminée en fonction des recommandations officielles des fabricants des composants utilisés dans le Yocto-Motor-DC, et par des tests de durée limitée (1h) dans les conditions extrêmes, en environnement controllé. Si vous envisagez d'utiliser le Yocto-Motor-DC dans des conditions de température extrêmes pour une période prolongée, il est recommandé de faire des tests extensifs avant la mise en production.

2. Présentation



1: Prise USB micro-B
2: Yocto-bouton
3: Yocto-Led
4: Led d'erreur
5: Alimentation de la partie puissance (+)
6: Alimentation de la partie puissance (-)
7: Sortie Moteur (+)
8: Sortie Moteur (-)

2.1. Les éléments communs

Tous les Yocto-modules ont un certain nombre de fonctionnalités en commun.

Le connecteur USB

Les modules de Yoctopuce sont tous équipés d'une connectique USB 2.0 au format micro-B. Attention, le connecteur USB est simplement soudé en surface et peut être arraché si la prise USB venait à faire levier. Si les pistes sont restées en place, le connecteur peut être ressoudé à l'aide d'un bon fer et de flux. Alternativement, vous pouvez souder un fil USB directement dans les trous espacés de 1.27mm prévus à cet effet, prêt du connecteur.

Si vous utilisez une source de tension autre qu'un port USB hôte standard pour alimenter le module par le connecteur USB, vous devez respecter les caractéristiques assignées par le standard USB 2.0:

- **Tension min.:** 4.75 V DC

- **Tension max.:** 5.25 V DC
- **Protection contre les surintensités:** max. 5.0 A

En cas de tension supérieure, le module risque fort d'être détruit. En cas de tension inférieure, le comportement n'est pas déterminé, mais il peut conduire à une corruption du firmware.

Le Yocto-bouton

Le Yocto-bouton a deux fonctions. Premièrement, il permet d'activer la Yocto-balise (voir la Yocto-led ci-dessous). Deuxièmement, si vous branchez un Yocto-module en maintenant ce bouton appuyé, il vous sera possible de reprogrammer son firmware avec une nouvelle version. Notez qu'il existe une méthode plus simple pour mettre à jour le firmware depuis l'interface utilisateur, mais cette méthode-là peut fonctionner même lorsque le firmware chargé sur le module est incomplet ou corrompu.

La Yocto-Led

En temps normal la Yocto-Led sert à indiquer le bon fonctionnement du module: elle émet alors une faible lumière bleue qui varie lentement mimant ainsi une respiration. La Yocto-Led cesse de respirer lorsque le module ne communique plus, par exemple si il est alimenté par un hub sans connexion avec un ordinateur allumé.

Lorsque vous appuyez sur le Yocto-bouton, la Led passe en mode Yocto-balise: elle se met alors à flasher plus vite et beaucoup plus fort, dans le but de permettre une localisation facile d'un module lorsqu'on en a plusieurs identiques. Il est en effet possible de déclencher la Yocto-balise par logiciel, tout comme il est possible de détecter par logiciel une Yocto-balise allumée.

La Yocto-Led a une troisième fonctionnalité moins plaisante: lorsque ce logiciel interne qui contrôle le module rencontre une erreur fatale, elle se met à flasher SOS en morse[1]. Si cela arrivait débranchez puis rebranchez le module. Si le problème venait à se reproduire vérifiez que le module contient bien la dernière version du firmware, et dans l'affirmative contactez le support Yoctopuce[2].

La sonde de courant

Chaque Yocto-module est capable de mesurer sa propre consommation de courant sur le bus USB. La distribution du courant sur un bus USB étant relativement critique, cette fonctionnalité peut être d'un grand secours. La consommation de courant du module est consultable par logiciel uniquement.

Le numéro de série

Chaque Yocto-module a un numéro de série unique attribué en usine, pour les modules Yocto-Motor-DC ce numéro commence par MOTORCTL. Le module peut être piloté par logiciel en utilisant ce numéro de série. Ce numéro de série ne peut pas être changé.

Le nom logique

Le nom logique est similaire au numéro de série, c'est une chaine de caractère sensée être unique qui permet référencer le module par logiciel. Cependant, contrairement au numéro de série, le nom logique peut être modifié à volonté. L'intérêt est de pouvoir fabriquer plusieurs exemplaires du même projet sans avoir à modifier le logiciel de pilotage. Il suffit de programmer les même noms logiques dans chaque exemplaire. Attention le comportement d'un projet devient imprévisible s'il contient plusieurs modules avec le même nom logique et que le logiciel de pilotage essaye d'accéder à l'un de ces module à l'aide de son nom logique. A leur sortie d'usine, les modules n'ont pas de nom logique assigné, c'est à vous de le définir.

2.2. Les éléments spécifiques

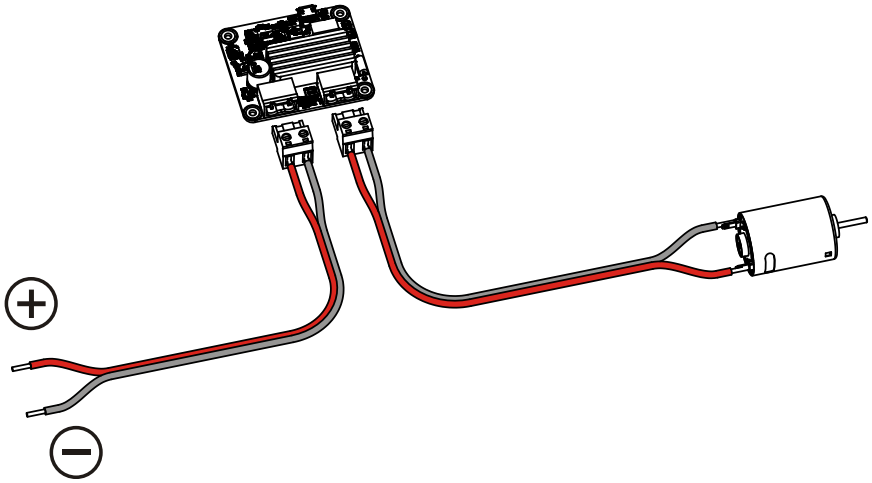
Connecteurs

Les connexions sont très simples: il suffit de brancher la source d'alimentation et le moteur sur les connecteurs adéquats.

[1] court-court-court long-long-long court-court-court
[2] support@yoctopuce.com

Attention: Il est primordial de respecter la polarité de l'alimentation, car le module n'est pas protégé contre les inversions. Brancher l'alimentation à l'envers détruirait instantanément le module. **Brancher l'alimentation sur la sortie moteur détruirait aussi instantanément le module**.

Si votre application utilise des fort courants, veillez à utiliser des câbles aussi courts que possible, en particulier entre la source de courant et le Yocto-Motor-DC.



Branchement de la partie puissance: attention aux inversions de polarité et de prises!

Si le Yocto-Motor-DC est utilisé dans une application où la polarité a son importance, tel qu'un éclairage à LED, il est aussi primordial de respecter la polarité en sortie de module.

Attention, l'entrée et la sortie du Yocto-Motor-DC ne sont pas en masse commune. Les quatre fils doivent impérativement être câblés séparément, sous peine de créer un court-circuit qui détruirait le module.

Radiateurs de dissipation

Lorsque le courant fournit est élevé, le Yocto-Motor-DC doit dissiper une quantité importante d'énergie en chaleur.

Selon le courant consommé, les radiateurs du Yocto-Motor-DC peuvent atteindre (voire dépasser) les 80°C. Évitez donc de les toucher en fonctionnement, et veillez à installer le module de façon à ce que la chaleur puisse s'évacuer si nécessaire. Pour plus de détail, référez-vous au chapitre montage et connectique.

Utilisé à basse puissance (jusqu'à 2-3 ampères), vous pouvez utiliser un boîtier Yoctopuce de type YoctoBox-Watt-Transp pour protéger votre module.

2.3. Isolation électrique fonctionnelle

Le Yocto-Motor-DC est conçu sous forme de deux circuits électriques bien distincts, séparés par une barrière d'isolation fonctionnelle. Cette isolation ne joue aucun rôle pour la sécurité, puisque les deux circuits du Yocto-Motor-DC travaillent en très basses tensions de sécurité (TBTS) et sont accessibles sans risque pour l'utilisateur à tout moment. L'isolation est simplement destinée à améliorer la fiabilité et le confort d'utilisation du Yocto-Motor-DC, permettant aux deux circuits de travailler avec des références de terre différentes.

[3] La présente description de l'isolation électrique s'applique à la dernière révision du produit. Certaines révisions antérieures du produit pourraient disposer de distances d'isolement et de ligne de fuite plus faible. Pour obtenir la distance d'isolement/ ligne de fuite d'un module plus ancien, contactez le support Yoctopuce en communiquant le numéro de série du module ou la référence d'achat à votre demande.

Bien qu'elle ne joue pas de rôle pour la sécurité, la barrière d'isolation a été conçue selon les règles qui s'appliqueraient pour une isolation supplémentaire sur un circuit secondaire. Ses caractéristiques sont les suivantes [3]:

- **Tension d'isolation[4]:** 0.5kV
- **Distance d'isolement:** 0.55mm
- **Ligne de fuite:** 0.55mm
- **Groupe de matériau:** Cat IIIa (FR4)

2.4. Accessoires optionnels

Les accessoires ci-dessous ne sont pas nécessaires à l'utilisation du module Yocto-Motor-DC, mais pourraient vous être utiles selon l'utilisation que vous en faites. Il s'agit en général de produits courants que vous pouvez vous procurer chez vos fournisseurs habituels de matériel de bricolage. Pour vous éviter des recherches, ces produits sont en général aussi disponibles sur le shop de Yoctopuce.

Vis et entretoises

Pour fixer le module Yocto-Motor-DC à un support, vous pouvez placer des petites vis de 3mm avec une tête de 8mm au maximum dans les trous prévus ad-hoc. Il est conseillé de les visser dans des entretoises filetées, que vous pourrez fixer sur le support. Vous trouverez plus de détail à ce sujet dans le chapitre concernant le montage et la connectique.

Micro-hub USB

Si vous désirez placer plusieurs modules Yoctopuce dans un espace très restreint, vous pouvez les connecter ensemble à l'aide d'un micro-hub USB. Yoctopuce fabrique des hubs particulièrement petits précisément destinés à cet usage, dont la taille peut être réduite à 20mm par 36mm, et qui se montent en soudant directement les modules au hub via des connecteurs droits ou des câbles nappe. Pour plus de détails, consulter la fiche produit du micro-hub USB.

YoctoHub-Ethernet, YoctoHub-Wireless and YoctoHub-GSM

Vous pouvez ajouter une connectivité réseau à votre Yocto-Motor-DC grâce aux hubs YoctoHub-Ethernet, YoctoHub-Wireless et YoctoHub-GSM qui offrent respectivement une connectivité Ethernet, Wifi et GSM. Chacun de ces hubs peut piloter jusqu'à trois modules Yoctopuce et se comporte exactement comme un ordinateur normal qui ferait tourner un *VirtualHub*.

Connecteurs 1.27mm (ou 1.25mm)

Si vous désirez raccorder le module Yocto-Motor-DC à un Micro-hub USB ou a un YoctoHub en évitant l'encombrement d'un vrai cable USB, vous pouvez utiliser les 4 pads au pas 1.27mm juste derrière le connecteur USB. Vous avez alors deux possibilités.

Vous pouvez monter directement le module sur le hub à l'aide d'un jeu de vis et entretoises, et les connecter à l'aide de connecteurs board-to-board au pas 1.27mm. Pour éviter les court-circuits, soudez de préférence le connecteur femelle sur le hub et le connecteur mâle sur le Yocto-Motor-DC.

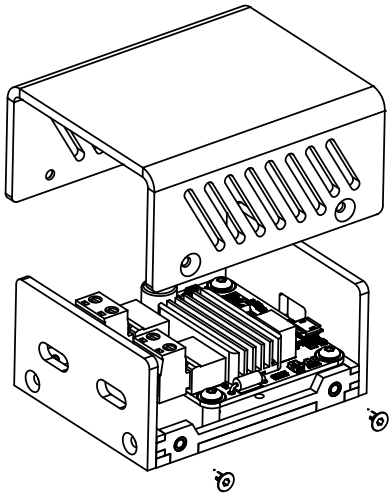
Vous pouvez aussi utiliser un petit câble à 4 fils doté de connecteurs au pas 1.27mm (ou 1.25mm, la différence est négligeable pour 4 pins), ce qui vous permet de déporter le module d'une dizaine de centimètres. N'allongez pas trop la distance si vous utilisez ce genre de câble, car il n'est pas blindé et risque donc de provoquer des émissions électromagnétiques indésirables.

Boîtier

Votre Yocto-Motor-DC a été conçu pour pouvoir être installé tel quel dans votre projet. Néanmoins Yoctopuce commercialise des boîtiers spécialement conçus pour les modules Yoctopuce. Vous

[4] **Valeur nominale non testée.**

trouverez plus d'informations à propos de ces boîtiers sur le site de Yoctopuce[5]. Le boîtier recommandé pour votre Yocto-Motor-DC est le modèle $SPECBOX$



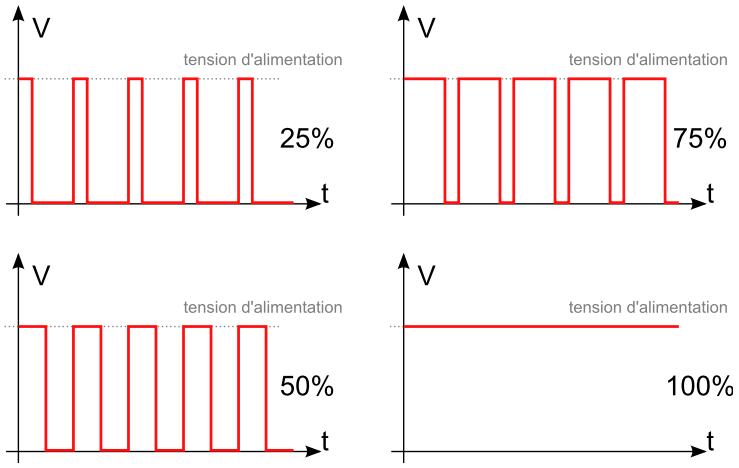
Vous pouvez installer votre Yocto-Motor-DC dans un boîtier optionnel.

[5] http://www.yoctopuce.com/EN/products/category/enclosures

3. Fonctionnement du Yocto-Motor-DC

3.1. Principe

Le Yocto-Motor-DC fonctionne en hachant la tension qu'il trouve sur son entrée *Power*. Le courant est envoyé au moteur sous la forme de plusieurs milliers d'impulsions par seconde. L'amplitude des impulsion est fixe: elle correspond directement à la tension d'alimentation. La puissance est contrôlée en faisant varier le rapport durée des impulsions / période. C'est une méthode classique pour contrôler les moteurs à balais qui se contentent parfaitement d'un tel traitement. Mais elle est aussi utilisable pour contrôler l'intensité de certains éclairages, en particulier ceux à LED.



Le moteur est alimenté par des impulsions dont la longueur est proportionnelle à la puissance désirée.

3.2. Fréquence

La fréquence de fonctionnement est configurable de 250Hz à 32KHz, ce qui permet d'optimiser le fonctionnement du Yocto-Motor-DC en fonction de l'application choisie.

Dans le cas d'un moteur, une fréquence faible donne de meilleurs résultats au prix d'un sifflement audible assez caractéristique. Ce bruit peut être éliminé en réglant la fréquence à une valeur supérieure à 20KHz, mais le Yocto-Motor-DC chauffera alors plus et certains moteurs pourraient avoir de la peine à démarrer.

Pour faciliter le démarrage, le Yocto-Motor-DC peut appliquer une courte impulsion continue de quelques millisecondes lors de la mise sous tension du moteur, indépendamment de la fréquence choisie. La durée de ce starter est configurable.

3.3. Fonctionnalités

Contrôle de puissance

Le Yocto-Motor-DC est capable de faire tourner un moteur dans les deux sens. La puissance est donnée sous la forme d'un nombre entre -100 et 100%. Pour faire tourner le moteur à l'envers, le Yocto-Motor-DC inverse la polarité de la sortie. Il vous faudra donc être très prudent si vous utilisez le Yocto-Motor-DC pour piloter un dispositif qui ne supporte pas les inversions de polarité.

Freinage

Le Yocto-Motor-DC est capable de freiner le moteur qu'il contrôle en court-circuitant les pôles du moteur. Il s'agit d'un freinage électro-magnétique avec les limitations que cela comporte: il n'est réellement efficace qu'à grande vitesse.

Contrôle de tension

Le Yocto-Motor-DC mesure en permanence la tension présente sur son alimentation, et fournit cette mesure à travers l'API Yoctopuce. Si la tension passe en dessous de 3V, même pour un bref instant, le Yocto-Motor-DC s'arrêtera automatiquement. Cette séurité est liée au fait que la partie puissance ne fonctionne pas en dessous de 3V.

Il est possible de configurer une limite arbitraire supérieure à 3V: dans ce cas, si la tension d'alimentation passe en dessous de cette limite pendant plus d'une seconde, le Yocto-Motor-DC s'arrêtera. Cette limite permet de se protéger d'une sur-décharge dans le cas d'une alimentation sur batterie.

Attention: même à l'arrêt, le Yocto-Motor-DC peut laisser consommer environ 1mA sur l'alimentation. Par conséquent, si une batterie reste branchée sur le Yocto-Motor-DC, elle finira tôt ou tard par être complètement déchargée.

Le Yocto-Motor-DC n'est pas protégé contre les surtensions. Si vous appliquez une tension supérieure à 18V sur l'entrée, vous risquez d'endommager irrémédiablement le module.

A l'arrêt, lorsque rien n'est branché sur l'alimentation, le Yocto-Motor-DC peut détecter quelque volts sur l'entrée, c'est un phénomène normal. Cela ne l'empêchera pas de détecter l'absence d'alimentation à la première tentative de mise en marche.

Contrôle de courant

Le Yocto-Motor-DC mesure en permanence le courant moyen qui le traverse, et fournit cette mesure à travers l'API Yoctopuce. Si le courant qui traverse le Yocto-Motor-DC passe, même brièvement, au dessus de 32A, le Yocto-Motor-DC s'arrêtera automatiquement.

Il est possible de configurer une limite de courant inférieure arbitraire: dans ce cas, si le courant qui traverse passe au dessus de cette limite pendant plus d'une seconde, le Yocto-Motor-DC s'arrêtera. Cette limite permet de protéger l'équipement contrôlé par le Yocto-Motor-DC contre des courants excessifs.

Contrôle de température

Le Yocto-Motor-DC mesure en permanence sa température, et fournit cette mesure à travers l'API Yoctopuce. Si la température du Yocto-Motor-DC dépasse les 80°C, le Yocto-Motor-DC s'arrêtera automatiquement pour éviter d'endommager le matériel.

Fail-safe

Il est possible de configurer le Yocto-Motor-DC pour qu'il s'arrête automatiquement s'il se rend compte qu'il n'a pas été contacté par le processus de contrôle depuis un temps donné. Cela permet d'éviter de voir un robot échapper à tout contrôle parce que l'application censée le piloter aura crashé.

3.4. Protections

Les protections décrites précédemment sont destinées à garantir le fonctionnement en toute sécurité du Yocto-Motor-DC. Si une de ces protections est déclenchée, le Yocto-Motor-DC s'arrêtera et la LED rouge présente sur le module s'allumera. L'état du Yocto-Motor-DC devra alors être réinitialisé explicitement à l'aide de la fonction idoine de l'API.

4. Premiers pas

Par design, tous les modules Yoctopuce se pilotent de la même façon, c'est pourquoi les documentations des modules de la gamme sont très semblables. Si vous avez déjà épluché la documentation d'un autre module Yoctopuce, vous pouvez directement sauter à la description de sa configuration.

4.1. Prérequis

Pour pouvoir profiter pleinement de votre module Yocto-Motor-DC, vous devriez disposer des éléments suivants.

Un ordinateur

Les modules de Yoctopuce sont destinés à être pilotés par un ordinateur (ou éventuellement un microprocesseur embarqué). Vous écrirez vous-même le programme qui pilotera le module selon vos besoin, à l'aide des informations fournies dans ce manuel.

Yoctopuce fourni les librairies logicielles permettant de piloter ses modules pour les systèmes d'exploitation suivants: Windows, macOS, Linux et Android. Les modules Yoctopuce ne nécessitent pas l'installation de driver (ou pilote) spécifiques, car ils utilisent le driver HID[1] fourni en standard dans tous les systèmes d'exploitation.

Les versions de Windows actuellement supportées sont Windows XP, Windows 2003, Windows Vista, Windows 7, Windows 8 et Windows 10. Les versions 32 bit et 64 bit sont supportées. La librairie de programmation est aussi disponible pour la Plateforme Windows Universelle (UWP) supportées par toutes les versions Windows 10, y compris Windows 10 IoT. Yoctopuce teste régulièrement le bon fonctionnement des modules sur Windows 7 et Windows 10.

Les versions de macOS actuellement supportées sont Mac OS X 10.9 (Maverick), 10.10 (Yosemite), 10.11 (El Capitan), macOS 10.12 (Sierra), macOS 10.13 (High Sierra) and macOS 10.14 (Mojave). Yoctopuce teste régulièrement le bon fonctionnement des modules sur macOS 10.14.

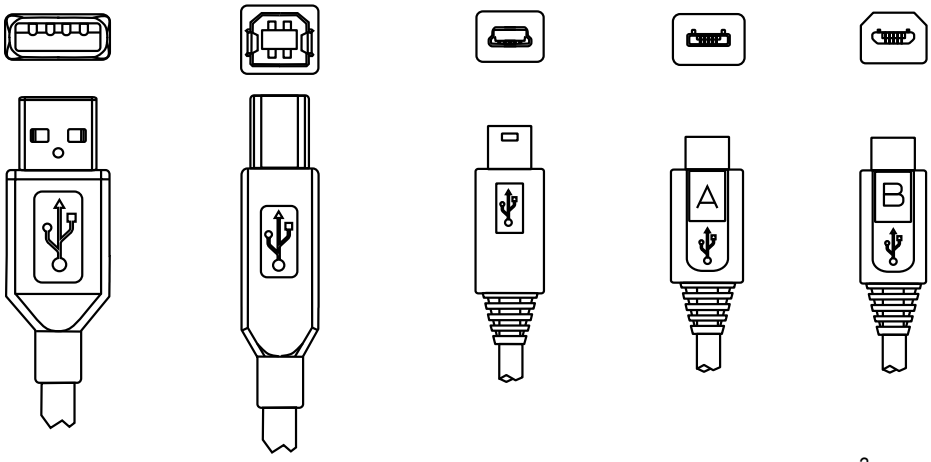
Les versions de Linux supportées sont les kernels 2.6, 3.x et 4.x. D'autre versions du kernel et même d'autres variantes d'Unix sont très susceptibles d'être utilisées sans problème, puisque le support de Linux est fait via l'API standard de la **libusb**, disponible aussi pour FreeBSD par exemple. Yoctopuce teste régulièrement le bon fonctionnement des modules sur un kernel Linux 4.15 (Ubuntu 18.04 LTS).

[1] Le driver HID est celui qui gère les périphériques tels que la souris, le clavier, etc.

Les versions de Android actuellement supportées sont 3.1 et suivantes. De plus, il est nécessaire que la tablette ou le téléphone supporte le mode USB *Host*. Yoctopuce teste régulièrement le bon fonctionnement des modules avec Android 7.x sur un Samsung Galaxy A6 avec la librairie Java pour Android.

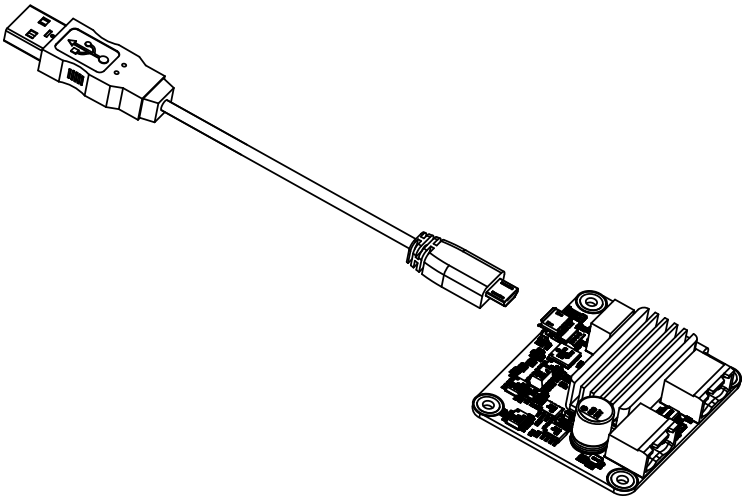
Un cable USB 2.0 de type A-micro B

Il existe trois tailles de connecteurs USB 2.0, la taille "normale" que vous utilisez probablement pour brancher votre imprimante. La taille mini encore très courante et enfin la taille micro, souvent utilisée pour raccorder les téléphones portables, pour autant qu'ils n'arborent pas une pomme. Les modules de Yoctopuce sont tous équipés d'une connectique au format micro-USB.



Les connecteurs USB 2.0 les plus courants: A, B, Mini B, Micro A, Micro B.[2]

Pour connecter votre module Yocto-Motor-DC à un ordinateur, vous avez besoin d'un cable USB 2.0 de type A-micro B. Vous trouverez ce cable en vente à des prix très variables selon les sources, sous la dénomination *USB A to micro B Data cable*. Prenez garde à ne pas acheter par mégarde un simple câble de charge, qui ne fournirait que le courant mais sans les fils de données. Le bon câble est disponible sur le shop de Yoctopuce.



Vous devez raccorder votre module Yocto-Motor-DC à l'aide d'un cable USB 2.0 de type A - micro B

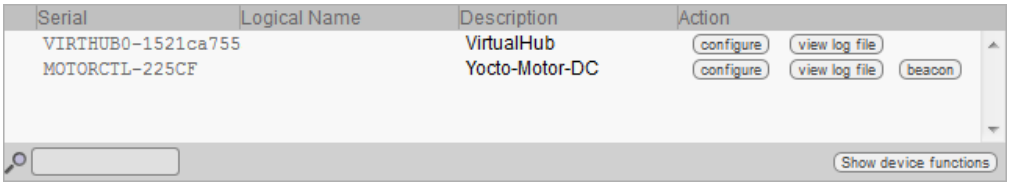
Si vous branchez un hub USB entre l'ordinateur et le module Yocto-Motor-DC, prenez garde à ne pas dépasser les limites de courant imposées par USB, sous peine de faire face des comportements instables non prévisibles. Vous trouverez plus de détail à ce sujet dans le chapitre concernant le montage et la connectique.

[2] *Le connecteur Mini A a existé quelque temps, mais a été retiré du standard USB http://www.usb.org/developers/Deprecation_Announcement_052507.pdf*

4.2. Test de la connectivité USB

Arrivé à ce point, votre Yocto-Motor-DC devrait être branché à votre ordinateur, qui devrait l'avoir reconnu. Il est temps de le faire fonctionner.

Rendez-vous sur le site de Yoctopuce et téléchargez le programme *Virtual Hub*[3], Il est disponible pour Windows, Linux et Mac OS X. En temps normal le programme Virtual Hub sert de couche d'abstraction pour les langages qui ne peuvent pas accéder aux couches matérielles de votre ordinateur. Mais il offre aussi une interface sommaire pour configurer vos modules et tester les fonctions de base, on accède à cette interface à l'aide d'un simple browser web [4]. Lancez le *Virtual Hub* en ligne de commande, ouvrez votre browser préféré et tapez l'adresse *http://127.0.0.1:4444*. Vous devriez voir apparaître la liste des modules Yoctopuce raccordés à votre ordinateur.



Liste des modules telle qu'elle apparaît dans votre browser.

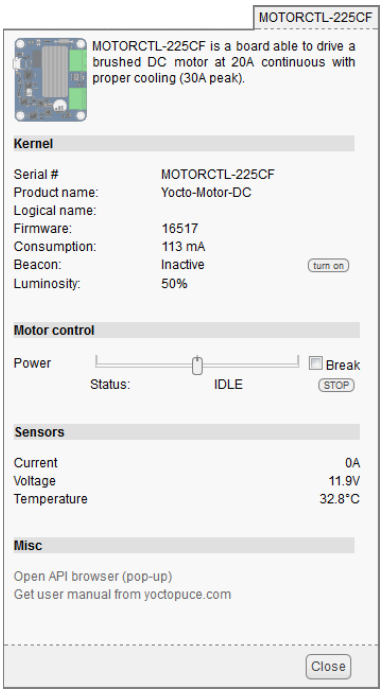
4.3. Localisation

Il est alors possible de localiser physiquement chacun des modules affichés en cliquant sur le bouton **beacon**, cela a pour effet de mettre la Yocto-Led du module correspondant en mode "balise", elle se met alors à clignoter ce qui permet de la localiser facilement. Cela a aussi pour effet d'afficher une petite pastille bleue à l'écran. Vous obtiendrez le même comportement en appuyant sur le Yocto-bouton d'un module.

4.4. Test du module

La première chose à vérifier est le bon fonctionnement de votre module: cliquez sur le numéro de série correspondant à votre module, et une fenêtre résumant les propriétés de votre Yocto-Motor-DC.

[3] www.yoctopuce.com/FR/virtualhub.php
[4] L'interface est testée avec Chrome, FireFox, Safari, Edge et IE 11.

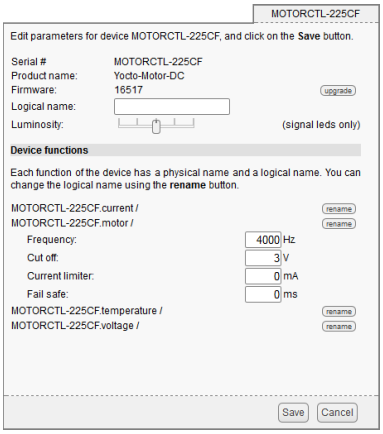


Propriétés du module Yocto-Motor-DC.

Cette fenêtre vous permet entre autres de jouer avec votre module, en déplaçant les curseurs vous pouvez modifier la puissance ou le freinage appliqué à la sortie. La tension de l'alimentation, le courant qui traverse le module, et la température sont affichés en direct.

4.5. Configuration

Si, dans la liste de modules, vous cliquez sur le bouton **configure** correspondant à votre module, la fenêtre de configuration apparaît.



Configuration du module Yocto-Motor-DC.

Firmware

Le firmware du module peut être facilement mis à jour à l'aide de l'interface. Les firmwares destinés aux modules Yoctopuce se présentent sous la forme de fichiers .byn et peuvent être téléchargés depuis le site web de Yoctopuce.

Pour mettre à jour un firmware, cliquez simplement sur le bouton **upgrade** de la fenêtre de configuration et suivez les instructions. Si pour une raison ou une autre, la mise à jour venait à échouer, débranchez puis rebranchez le module. Recommencer la procédure devrait résoudre alors le problème. Si le module a été débranché alors qu'il était en cours de reprogrammation, il ne

fonctionnera probablement plus et ne sera plus listé dans l'interface. Mais il sera toujours possible de le reprogrammer correctement en utilisant le programme *Virtual Hub*[5] en ligne de commande [6].

Nom logique du module

Le nom logique est un nom choisi par vous, qui vous permettra d'accéder à votre module, de la même manière qu'un nom de fichier vous permet d'accéder à son contenu. Un nom logique doit faire au maximum 19 caractères, les caractères autorisés sont les caractères `A..Z a..z 0..9 _` et `-`. Si vous donnez le même nom logique à deux modules raccordés au même ordinateur, et que vous tentez d'accéder à l'un des modules à l'aide de ce nom logique, le comportement est indéterminé: vous n'avez aucun moyen de savoir lequel des deux va répondre.

Luminosité

Ce paramètre vous permet d'agir sur l'intensité maximale des leds présentes sur le module. Ce qui vous permet, si nécessaire, de le rendre un peu plus discret tout en limitant sa consommation. Notez que ce paramètre agit sur toutes les leds de signalisation du module, y compris la Yocto-Led. Si vous branchez un module et que rien ne s'allume, cela veut peut être dire que sa luminosité a été réglée à zéro.

Nom logique des fonctions

Chaque module Yoctopuce a un numéro de série, et un nom logique. De manière analogue, chaque fonction présente sur chaque module Yoctopuce a un nom matériel et un nom logique, ce dernier pouvant être librement choisi par l'utilisateur. Utiliser des noms logiques pour les fonctions permet une plus grande flexibilité au niveau de la programmation des modules

Les fonctions fournies par le module Yocto-Motor-DC sont les suivantes:

Motor

La fonction *motor* permet de contrôler le moteur proprement dit: marche avant, arrière, freinage. Elle permet aussi de gérer les diverses condition d'erreurs qui peut être générée par les protections du Yocto-Motor-DC.

Voltage

La fonction *voltage* décrit en permanence la tension présente l'entrée du Yocto-Motor-DC.

Current

La fonction *current* décrit en permanence le courant moyen qui traverse le Yocto-Motor-DC.

Temperature

La fonction *temperature* décrit en permanence la température du Yocto-Motor-DC, mesuré, au niveau de la partie puissance, sous le radiateur supérieur.

Les fonctions *Voltage*, *Current* et *Temperature* sont exactement les mêmes que les celles que l'on trouve sur d'autres capteurs Yoctopuce tels que le Yocto-Volt, le Yocto-Amp ou encore le Yocto-Temperature. Elles se gèrent de la même façon. Cependant, le $PRODANME$ ne disposant pas d'enregistreur de données, il n'est pas possible de demander au $PRODANME$ d'enregistrer automatiquement les valeurs mesurées par ses différents capteurs.

[5] www.yoctopuce.com/FR/virtualhub.php
[6] Consultez la documentation du virtual hub pour plus de détails

5. Montage et connectique

Ce chapitre fournit des explications importantes pour utiliser votre module Yocto-Motor-DC en situation réelle. Prenez soin de le lire avant d'aller trop loin dans votre projet si vous voulez éviter les mauvaises surprises.

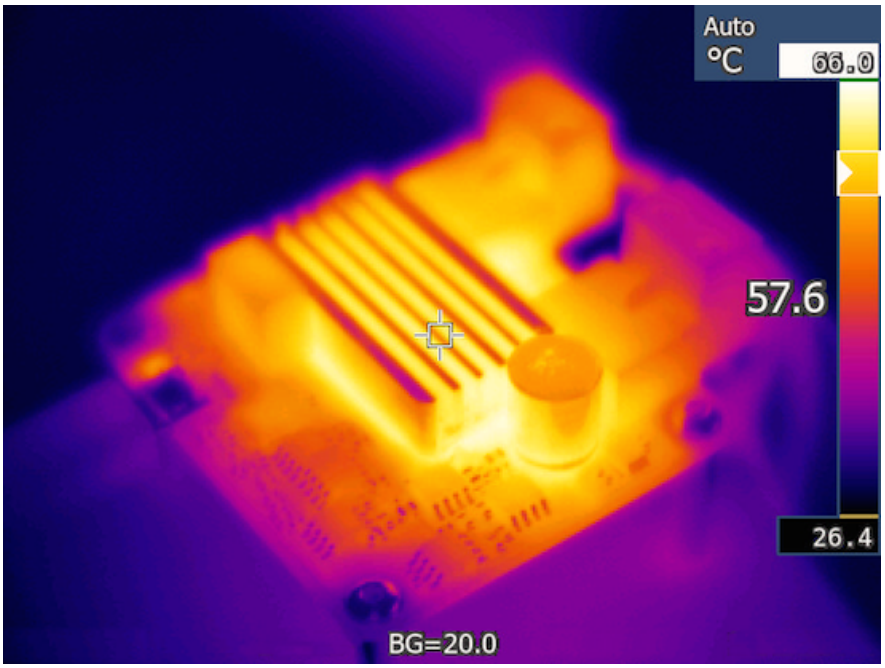
5.1. Chaleur

Suivant l'utilisation qui en est faite, la partie puissance du Yocto-Motor-DC peut dépasser les 80°C. A cette température, le Yocto-Motor-DC se met en protection et coupe l'alimentation du moteur. C'est pourquoi il ne faut pas compter utiliser le Yocto-Motor-DC à plus de 20A pendant une période prolongée. Si vous devez utiliser le Yocto-Motor-DC à forte puissance, veillez à ce qu'il soit refroidi efficacement.

L'échauffement causé par la circulation du courant dans les pistes du circuit se calcule par rapport à la température ambiante. Jusqu'à 10 ampères, le circuit est conçu pour ne pas produire un échauffement de plus de 10 à 15°C. Il suffit donc de placer le module de sorte à ce que la température ambiante ne s'élève pas (en laissant la chaleur s'évacuer), et la température se stabilisera par simple convection thermique. La position verticale est la meilleure pour la circulation naturelle par convection.

À 15 ampères, l'électricité produit un échauffement de 20°C par rapport à la température ambiante. À 20 ampères, l'échauffement monte à 30°C. À 25 ampères, l'échauffement des pistes atteint 60°C par rapport à la température ambiante ! Il est donc essentiel de garantir une évacuation rapide de l'air chaud autour du radiateur, sans quoi la température ambiante monte et par conséquent la température du circuit atteint les 80°C, limite de fonctionnement du module.

La manière la plus simple de garantir l'évacuation de l'air chaud du radiateur consiste à faire circuler l'air à l'aide d'un petit ventilateur, soufflant dans l'axe des ailettes des radiateurs. On trouve facilement ce genre de petits ventilateurs carrés de 40mm, fonctionnant directement en 5V ou 12V.

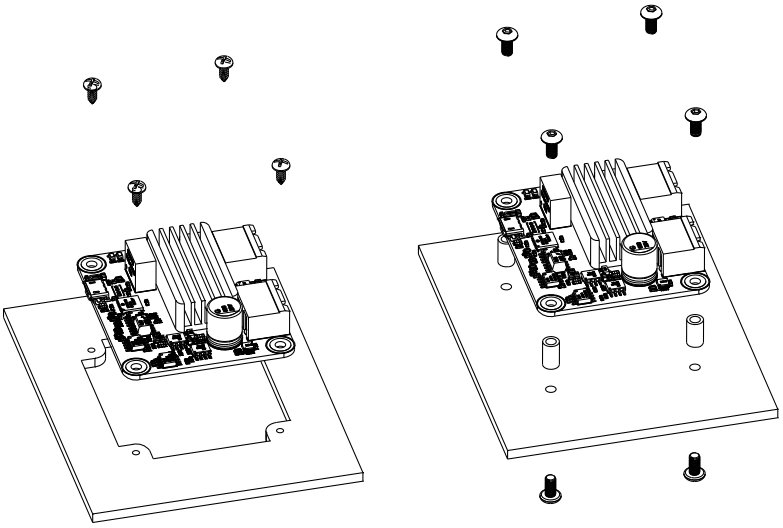


Quand vous poussez la puissance à fond, la partie puissance du module devient rapidement très chaude.

Pour conclure, n'oubliez pas de tenir compte du fait que la régulation de puissance est faite en hachant le courant. Par exemple, si le contrôleur mesure 12,5A de moyenne à 50% de puissance, cela signifie que vous faites en réalité circuler des pointes de 25A durant 50% du temps. Dans ce cas, il faut donc compter avec un échauffement de 30°C (50% de 60°C). En règle générale, il vaut donc mieux adapter la charge (le moteur) aux besoins plutôt que d'utiliser un moteur sur-dimensionné en sous-régime.

5.2. Fixation

Pendant la mise au point de votre projet vous pouvez vous contenter de laisser le module se promener au bout de son câble. Veillez simplement à ce qu'il ne soit pas en contact avec quoi que soit de conducteur (comme vos outils), ou sensible à la chaleur. Une fois votre projet pratiquement terminé il faudra penser à faire en sorte que votre module soit solidement arrimé pour éviter tout accident.



Exemples de montage sur un support.

Le module Yocto-Motor-DC dispose de trous de montage 3mm. Vous pouvez utiliser ces trous pour y passer des vis. Le diamètre de la tête de ces vis ne devra pas dépasser 8mm, sous peine d'endommager les circuits du module. Veillez à ce que la surface inférieure du module ne soit pas en

contact avec le support. La méthode recommandée consiste à utiliser des entretoises, mais il en existe d'autres: vous pouvez utiliser de la colle, pour autant qu'elle résiste à la chaleur.

Si vous comptez visser votre module directement contre une paroi conductrice, un chassis métallique par exemple, intercalez une couche isolante entre les deux. Sinon vous aller à coup sûr provoquer un court-circuit: il y a des pads à nu sous votre module. Du simple ruban adhésif isolant devrait faire l'affaire.

5.3. Contraintes d'alimentation par USB

Bien que USB signifie *Universal Serial BUS*, les périphériques USB ne sont pas organisés physiquement en bus mais en arbre, avec des connections point-à-point. Cela a des conséquences en termes de distribution électrique: en simplifiant, chaque port USB doit alimenter électriquement tous les périphériques qui lui sont directement ou indirectement connectés. Et USB impose des limites.

En théorie, un port USB fournit 100mA, et peut lui fournir (à sa guise) jusqu'à 500mA si le périphérique les réclame explicitement. Dans le cas d'un hub non-alimenté, il a droit à 100mA pour lui-même et doit permettre à chacun de ses 4 ports d'utiliser 100mA au maximum. C'est tout, et c'est pas beaucoup. Cela veut dire en particulier qu'en théorie, brancher deux hub USB non-alimentés en cascade ne marche pas. Pour cascader des hubs USB, il faut utiliser des hubs USB alimentés, qui offriront 500mA sur chaque port.

En pratique, USB n'aurait pas eu le succès qu'il a si il était si contraignant. Il se trouve que par économie, les fabricants de hubs omettent presque toujours d'implémenter la limitation de courant sur les ports: ils se contentent de connecter l'alimentation de tous les ports directement à l'ordinateur, tout en se déclarant comme *hub alimenté* même lorsqu'ils ne le sont pas (afin de désactiver tous les contrôles de consommation dans le système d'exploitation). C'est assez malpropre, mais dans la mesure où les ports des ordinateurs sont eux en général protégés par une limitation de courant matérielle vers 2000mA, ça ne marche pas trop mal, et cela fait rarement des dégâts.

Ce que vous devez en retenir: si vous branchez des modules Yoctopuce via un ou des hubs non alimentés, vous n'aurez aucun garde-fou et dépendrez entièrement du soin qu'aura mis le fabricant de votre ordinateur pour fournir un maximum de courant sur les ports USB et signaler les excès avant qu'ils ne conduisent à des pannes ou des dégâts matériels. Si les modules sont sous-alimentés, ils pourraient avoir un comportement bizarre et produire des pannes ou des bugs peu reproductibles. Si vous voulez éviter tout risque, ne cascadez pas les hubs non-alimentés, et ne branchez pas de périphérique consommant plus de 100mA derrière un hub non-alimenté.

Pour vous faciliter le contrôle et la planification de la consommation totale de votre projet, tous les modules Yoctopuce sont équipés d'une sonde de courant qui indique (à 5mA près) la consommation du module sur le bus USB.

Notez enfin que le câble USB lui-même peut aussi représenter une cause de problème d'alimentation, en particulier si les fils sont trop fins ou si le câble est trop long [1]. Les bons câbles utilisent en général des fils AWG 26 ou AWG 28 pour les fils de données et des fils AWG 24 pour les fils d'alimentation.

5.4. Compatibilité électromagnétique (EMI)

Les choix de connectique pour intégrer le Yocto-Motor-DC ont naturellement une incidence sur les émissions électromagnétiques du système, et donc sur la conformité avec les normes concernées.

Les mesures de référence que nous effectuons pour valider la conformité avec la norme IEC CISPR 11 sont faites sans aucun boîtier, mais en raccordant les modules par un câble USB blindé, conforme à la spécification USB 2.0: le blindage du câble est relié au blindage des deux connecteurs, et la résistance totale entre le blindage des deux connecteurs est inférieure 0.6Ω. Le

[1] www.yoctopuce.com/FR/article/cables-usb-la-taille-compte

câble utilisé fait 3m, de sorte à exposer un segment d'un mètre horizontal, un segment d'un mètre vertical et de garder le dernier mètre le plus proche de l'ordinateur hôte à l'intérieur d'un bloc de ferrite.

Si vous utilisez un câble non blindé ou incorrectement blindé, votre système fonctionnera sans problème mais vous risquez de n'être pas conforme à la norme. Dans le cadre de systèmes composés de plusieurs modules raccordés par des câbles au pas 1.27mm, ou de capteurs déportés, vous pourrez en général récupérer la conformité avec la norme d'émission en utilisant un boîtier métallique offrant une enveloppe de blindage externe.

Toujours par rapport aux normes de compatibilité électromagnétique, la longueur maximale supportée du câble USB est de 3m. En plus de pouvoir causer des problèmes de chute de tension, l'utilisation de câbles plus long aurait des incidences sur les test d'immunité électromagnétiques à effectuer pour respecter les normes.

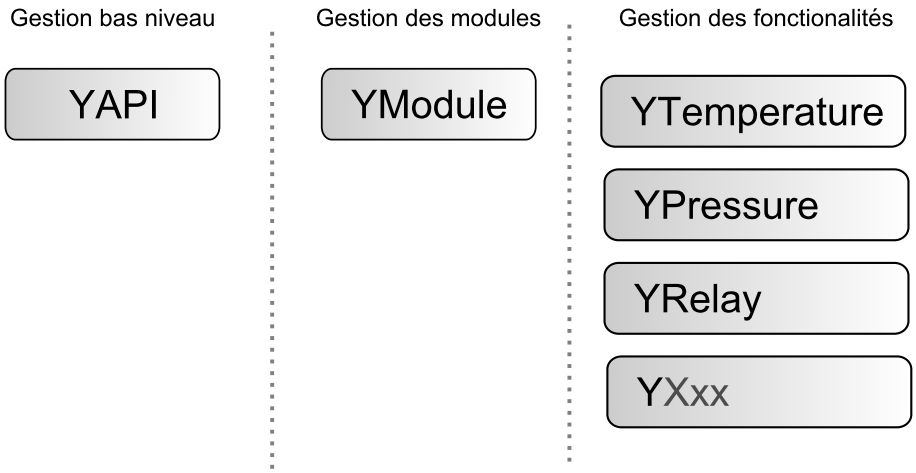
6. Programmation, concepts généraux

L'API Yoctopuce a été pensée pour être à la fois simple à utiliser, et suffisamment générique pour que les concepts utilisés soient valables pour tous les modules de la gamme Yoctopuce et ce dans tous les langages de programmation disponibles. Ainsi, une fois que vous aurez compris comment piloter votre Yocto-Motor-DC dans votre langage de programmation favori, il est très probable qu'apprendre à utiliser un autre module, même dans un autre langage, ne vous prendra qu'un minimum de temps.

6.1. Paradigme de programmation

L'API Yoctopuce est une API orientée objet. Mais dans un souci de simplicité, seules les bases de la programmation objet ont été utilisées. Même si la programmation objet ne vous est pas familière, il est peu probable que cela vous soit un obstacle à l'utilisation des produits Yoctopuce. Notez que vous n'aurez jamais à allouer ou désallouer un objet lié à l'API Yoctopuce: cela est géré automatiquement.

Il existe une classe par type de fonctionnalité Yoctopuce. Le nom de ces classes commence toujours par un Y suivi du nom de la fonctionnalité, par exemple *YTemperature*, *YRelay*, *YPressure*, etc.. Il existe aussi une classe *YModule*, dédiée à la gestion des modules en temps que tels, et enfin il existe la classe statique YAPI, qui supervise le fonctionnement global de l'API et gère les communications à bas niveau.



Structure de l'API Yoctopuce.

La classe YSensor

A chaque fonctionnalité d'un module Yoctopuce, correspond une classe: YTemperature pour mesurer la température, YVoltage pour mesurer une tension, YRelay pour contrôler un relais, etc. Il existe cependant une classe spéciale qui peut faire plus: YSensor.

Cette classe YSensor est la classe parente de tous les senseurs Yoctopuce, elle permet de contrôler n'importe quel senseur, quel que soit son type, en donnant accès au fonctions communes à tous les senseurs. Cette classe permet de simplifier la programmation d'applications qui utilisent beaucoup de senseurs différents. Mieux encore, si vous programmez une application basée sur la classe YSensor elle sera compatible avec tous les senseurs Yoctopuce, y compris ceux qui n'existent pas encore.

Programmation

Dans l'API Yoctopuce, la priorité a été mise sur la facilité d'accès aux fonctionnalités des modules en offrant la possibilité de faire abstraction des modules qui les implémentent. Ainsi, il est parfaitement possible de travailler avec un ensemble de fonctionnalités sans jamais savoir exactement quel module les héberge au niveau matériel. Cela permet de considérablement simplifier la programmation de projets comprenant un nombre important de modules.

Du point de vue programmation, votre Yocto-Motor-DC se présente sous la forme d'un module hébergeant un certain nombre de fonctionnalités. Dans l'API , ces fonctionnalités se présentent sous la forme d'objets qui peuvent être retrouvés de manière indépendante, et ce de plusieurs manières.

Accès aux fonctionnalités d'un module

Accès par nom logique

Chacune des fonctionnalités peut se voir assigner un nom logique arbitraire et persistant: il restera stocké dans la mémoire flash du module, même si ce dernier est débranché. Un objet correspondant à une fonctionnalité *Xxx* munie d'un nom logique pourra ensuite être retrouvée directement à l'aide de ce nom logique et de la méthode *YXxx.FindXxx*. Notez cependant qu'un nom logique doit être unique parmi tous les modules connectés.

Accès par énumération

Vous pouvez énumérer toutes les fonctionnalités d'un même type sur l'ensemble des modules connectés à l'aide des fonctions classiques d'énumération *FirstXxx* et *nextXxxx* disponibles dans chacune des classes *YXxx*.

Accès par nom hardware

Chaque fonctionnalité d'un module dispose d'un nom hardware, assigné en usine qui ne peut être modifié. Les fonctionnalités d'un module peuvent aussi être retrouvées directement à l'aide de ce nom hardware et de la fonction *YXxx.FindXxx* de la classe correspondante.

Différence entre *Find* et *First*

Les méthodes *YXxx.FindXxxx* et *YXxx.FirstXxxx* ne fonctionnent pas exactement de la même manière. Si aucun module n'est disponible *YXxx.FirstXxxx* renvoie une valeur nulle. En revanche, même si aucun module ne correspond, *YXxx.FindXxxx* renverra objet valide, qui ne sera pas "online" mais qui pourra le devenir, si le module correspondant est connecté plus tard.

Manipulation des fonctionnalités

Une fois l'objet correspondant à une fonctionnalité retrouvé, ses méthodes sont disponibles de manière tout à fait classique. Notez que la plupart de ces sous-fonctions nécessitent que le module hébergeant la fonctionnalité soit branché pour pouvoir être manipulées. Ce qui n'est en général jamais garanti, puisqu'un module USB peut être débranché après le démarrage du programme de contrôle. La méthode *isOnline()*, disponible dans chaque classe, vous sera alors d'un grand secours.

Accès aux modules

Bien qu'il soit parfaitement possible de construire un projet en faisant abstraction de la répartition des fonctionnalités sur les différents modules, ces derniers peuvent être facilement retrouvés à l'aide de l'API. En fait, ils se manipulent d'une manière assez semblable aux fonctionnalités. Ils disposent d'un numéro de série affecté en usine qui permet de retrouver l'objet correspondant à l'aide de *YModule.Find()*. Les modules peuvent aussi se voir affecter un nom logique arbitraire qui permettra de les retrouver ensuite plus facilement. Et enfin la classe *YModule* comprend les méthodes d'énumération *YModule.FirstModule()* et *nextModule()* qui permettent de dresser la liste des modules connectés.

Interaction Function / Module

Du point de vue de l'API, les modules et leurs fonctionnalités sont donc fortement décorrélés à dessein. Mais l'API offre néanmoins la possibilité de passer de l'un à l'autre. Ainsi la méthode *get_module()*, disponible dans chaque classe de fonctionnalité, permet de retrouver l'objet correspondant au module hébergeant cette fonctionnalité. Inversement, la classe *YModule* dispose d'un certain nombre de méthodes permettant d'énumérer les fonctionnalités disponibles sur un module.

6.2. Le module Yocto-Motor-DC

Le module Yocto-Motor-DC offre une instance de la fonction Motor, qui permet de piloter un moteur à courant continu (variateur), ainsi qu'une fonction Voltage pour surveiller la tension de l'alimentation, une fonction Current pour contrôler la consommation instantanée du moteur et une fonction tempéreture pour surveiller la température du module lui-même.

module : Module

attribut	type	modifiable ?
productName	Texte	lecture seule
serialNumber	Texte	lecture seule
logicalName	Texte	modifiable
productId	Entier (hexadécimal)	lecture seule
productRelease	Entier (hexadécimal)	lecture seule
firmwareRelease	Texte	lecture seule
persistentSettings	Type énuméré	modifiable
luminosity	0..100%	modifiable
beacon	On/Off	modifiable
upTime	Temps	lecture seule
usbCurrent	Courant consommé (en mA)	lecture seule
rebootCountdown	Nombre entier	modifiable
userVar	Nombre entier	modifiable

motor : Motor

attribut	type	modifiable ?
logicalName	Texte	modifiable
advertisedValue	Texte	modifiable
motorStatus	Type énuméré	modifiable
drivingForce	Nombre (virgule fixe)	modifiable
brakingForce	Nombre (virgule fixe)	modifiable
cutOffVoltage	Nombre (virgule fixe)	modifiable
overCurrentLimit	Nombre entier	modifiable
frequency	Nombre (virgule fixe)	modifiable
starterTime	Nombre entier	modifiable
failSafeTimeout	Nombre entier	modifiable
command	Texte	modifiable

voltage : Voltage

attribut	type	modifiable ?

logicalName	Texte	modifiable
advertisedValue	Texte	modifiable
unit	Texte	lecture seule
currentValue	Nombre (virgule fixe)	lecture seule
lowestValue	Nombre (virgule fixe)	modifiable
highestValue	Nombre (virgule fixe)	modifiable
currentRawValue	Nombre (virgule fixe)	lecture seule
logFrequency	Fréquence	modifiable
reportFrequency	Fréquence	modifiable
advMode	Type énuméré	modifiable
calibrationParam	Paramètrs de calibration	modifiable
resolution	Nombre (virgule fixe)	modifiable
sensorState	Nombre entier	lecture seule
enabled	Booléen	modifiable

current : Current

attribut	type	modifiable ?
logicalName	Texte	modifiable
advertisedValue	Texte	modifiable
unit	Texte	lecture seule
currentValue	Nombre (virgule fixe)	lecture seule
lowestValue	Nombre (virgule fixe)	modifiable
highestValue	Nombre (virgule fixe)	modifiable
currentRawValue	Nombre (virgule fixe)	lecture seule
logFrequency	Fréquence	modifiable
reportFrequency	Fréquence	modifiable
advMode	Type énuméré	modifiable
calibrationParam	Paramètrs de calibration	modifiable
resolution	Nombre (virgule fixe)	modifiable
sensorState	Nombre entier	lecture seule
enabled	Booléen	modifiable

temperature : Temperature

attribut	type	modifiable ?
logicalName	Texte	modifiable
advertisedValue	Texte	modifiable
unit	Texte	modifiable
currentValue	Nombre (virgule fixe)	lecture seule
lowestValue	Nombre (virgule fixe)	modifiable
highestValue	Nombre (virgule fixe)	modifiable
currentRawValue	Nombre (virgule fixe)	lecture seule
logFrequency	Fréquence	modifiable
reportFrequency	Fréquence	modifiable
advMode	Type énuméré	modifiable
calibrationParam	Paramètrs de calibration	modifiable
resolution	Nombre (virgule fixe)	modifiable
sensorState	Nombre entier	lecture seule
sensorType	Type énuméré	modifiable
signalValue	Nombre (virgule fixe)	lecture seule
signalUnit	Texte	lecture seule
command	Texte	modifiable

6.3. Module

Interface de contrôle des paramètres généraux des modules Yoctopuce

La classe `YModule` est utilisable avec tous les modules USB de Yoctopuce. Elle permet de contrôler les paramètres généraux du module, et d'énumérer les fonctions fournies par chaque module.

productName

Chaîne de caractères contenant le nom commercial du module, préprogrammé en usine.

serialNumber

Chaine de caractères contenant le numéro de série, unique et préprogrammé en usine. Pour un module Yocto-Motor-DC, ce numéro de série commence toujours par MOTORCTL. Il peut servir comme point de départ pour accéder par programmation à un module particulier.

logicalName

Chaine de caractères contenant le nom logique du module, initialement vide. Cet attribut peut être changé au bon vouloir de l'utilisateur. Une fois initialisé à une valeur non vide, il peut servir de point de départ pour accéder à un module particulier. Si deux modules avec le même nom logique se trouvent sur le même montage, il n'y a pas moyen de déterminer lequel va répondre si l'on tente un accès par ce nom logique. Le nom logique du module est limité à 19 caractères parmi `A..Z`,`a..z`,`0..9`,_ et -.

productId

Identifiant USB du module, préprogrammé à la valeur 22 en usine.

productRelease

Numéro de révision du module hardware, préprogrammé en usine. La révision originale du retourne la valeur 1, la révision B retourne la valeur 2, etc.

firmwareRelease

Version du logiciel embarqué du module, elle change à chaque fois que le logiciel embarqué est mis à jour.

persistentSettings

Etat des réglages persistants du module: chargés depuis la mémoire non-volatile, modifiés par l'utilisateur ou sauvegardés dans la mémoire non volatile.

luminosity

Intensité lumineuse maximale des leds informatives (comme la Yocto-Led) présentes sur le module. C'est une valeur entière variant entre 0 (leds éteintes) et 100 (leds à l'intensité maximum). La valeur par défaut est 50. Pour changer l'intensité maximale des leds de signalisation du module, ou les éteindre complètement, il suffit donc de modifier cette valeur.

beacon

Etat de la balise de localisation du module.

upTime

Temps écoulé depuis la dernière mise sous tension du module.

usbCurrent

Courant consommé par le module sur le bus USB, en milli-ampères.

rebootCountdown

Compte à rebours pour déclencher un redémarrage spontané du module.

userVar

Attribut de type entier 32 bits à disposition de l'utilisateur.

6.4. Motor

Interface pour intéragir avec les moteurs, disponibles par exemple dans le Yocto-Motor-DC

La classe `YMotor` permet de piloter un moteur DC. Elle permet de configurer la puissance envoyée au moteur pour le faire tourner aussi bien dans un sens que dans l'autre, mais aussi de piloter des accélérations linéaires: le moteur accélére alors tout seul sans que vous vous ayez à vous en occuper. La librairie permet aussi de freiner le moteur: cela est réalisé en court-circuitant les pôles du moteur, ce qui le transforme en frein électro-magnétique.

logicalName

Chaîne de caractères contenant le nom logique du moteur, initialement vide. Cet attribut peut être changé au bon vouloir de l'utilisateur. Un fois initialisé à une valeur non vide, il peut servir de point de départ pour accéder à directement au moteur. Si deux moteurs portent le même nom logique dans un projet, il n'y a pas moyen de déterminer lequel va répondre si l'on tente un accès par ce nom logique. Le nom logique du module est limité à 19 caractères parmi `A..Z`,`a..z`,`0..9`,`_` et `-`.

advertisedValue

Courte chaîne de caractères résumant l'état actuel du moteur, et qui sera publiée automatiquement jusqu'au hub parent. Pour un moteur, la valeur publiée est son status (IDLE, BRAKE, FORWD, BACKWD, LOVOLT, HICURR, HIHEAT, FAILSF).

motorStatus

Etat du contrôleur du moteur, permettant de savoir si le moteur est à l'arrêt, en marche ou en condition d'erreur.

drivingForce

Puissance actuelle envoyée au moteur, sous forme de nombre réel entre -100% et +100%.

brakingForce

Force de freinage actuelle, en pourcents. La valeur 0 correspond à ne pas freiner (moteur en roue libre).

cutOffVoltage

Valeur limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation. Ce réglage permet d'éviter d'endommager un accumulateur un continuant à l'utiliser une fois "vide".

overCurrentLimit

Valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

frequency

Fréquence du signal PWM utilisée pour contrôler le moteur. Une fréquence basse est généralement plus efficace (les composant chauffent moins et le moteur démarre plus facilement), mais un bruit audible peut être généré. Une fréquence élevée peut réduire le bruit, mais il y a plus d'énergie perdue en chaleur.

starterTime

Durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

failSafeTimeout

Temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle. Passé ce delai, le contrôleur arrêtera le moteur et passera en mode erreur FAILSAFE. La sécurité failsafe est désactivée quand la valeur est à zéro (comportement par défaut).

command

Attribut magique permettant d'envoyer des commandes particulières au contrôleur. Si une commande n'est pas interprétée comme attendu, consultez les logs du module.

6.5. Current

Interface pour intéragir avec les capteurs de courant, disponibles par exemple dans le Yocto-Amp, le Yocto-Motor-DC et le Yocto-Watt

La classe `YCurrent` permet de lire et de configurer les capteurs de courant Yoctopuce. Elle hérite de la classe `YSensor` toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données.

logicalName

Chaîne de caractères contenant le nom logique du capteur de courant, initialement vide. Cet attribut peut être changé au bon vouloir de l'utilisateur. Un fois initialisé à une valeur non vide, il peut servir de point de départ pour accéder à directement au capteur de courant. Si deux capteurs de courant portent le même nom logique dans un projet, il n'y a pas moyen de déterminer lequel va répondre si l'on tente un accès par ce nom logique. Le nom logique du module est limité à 19 caractères parmi `A..Z`,`a..z`,`0..9`,`_` et `-`.

advertisedValue

Courte chaîne de caractères résumant l'état actuel du capteur de courant, et qui sera publiée automatiquement jusqu'au hub parent. Pour un capteur de courant, la valeur publiée est la valeur courante du courant.

unit

Courte chaîne de catactères représentant l'unité dans laquelle le courant est exprimée.

currentValue

Valeur actuelle du courant, en mA, sous forme de nombre à virgule.

lowestValue

Valeur minimale du courant, en mA, sous forme de nombre à virgule.

highestValue

Valeur maximale du courant, en mA, sous forme de nombre à virgule.

currentRawValue

Valeur brute mesurée par le capteur (sans arrondi ni calibration), sous forme de nombre à virgule.

logFrequency

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

reportFrequency

Fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques de valeurs sont désactivées.

advMode

Mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

calibrationParam

Paramètres de calibration supplémentaires (par exemple pour compenser l'effet d'un boîtier), sous forme de tableau d'entiers 16 bit.

resolution

Résolution de la mesure (précision de la représentation, mais pas forcément de la mesure elle-même).

sensorState

Etat du capteur (zero lorsque qu'une mesure actuelle est disponible).

enabled

Activation/désactivation de la mesure.

6.6. Voltage

Interface pour intéragir avec les capteurs de tension, disponibles par exemple dans le Yocto-Motor-DC, le Yocto-Volt et le Yocto-Watt

La classe `YVoltage` permet de lire et de configurer les capteurs de tension Yoctopuce. Elle hérite de la classe `YSensor` toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données.

logicalName

Chaîne de caractères contenant le nom logique du capteur de tension, initialement vide. Cet attribut peut être changé au bon vouloir de l'utilisateur. Un fois initialisé à une valeur non vide, il peut servir de point de départ pour accéder à directement au capteur de tension. Si deux capteurs de tension portent le même nom logique dans un projet, il n'y a pas moyen de déterminer lequel va répondre si l'on tente un accès par ce nom logique. Le nom logique du module est limité à 19 caractères parmi `A..Z`,`a..z`,`0..9`,`_` et `-`.

advertisedValue

Courte chaîne de caractères résumant l'état actuel du capteur de tension, et qui sera publiée automatiquement jusqu'au hub parent. Pour un capteur de tension, la valeur publiée est la valeur courante de la tension.

unit

Courte chaîne de catactères représentant l'unité dans laquelle la tension est exprimée.

currentValue

Valeur actuelle de la tension, en Volt, sous forme de nombre à virgule.

lowestValue

Valeur minimale de la tension, en Volt, sous forme de nombre à virgule.

highestValue

Valeur maximale de la tension, en Volt, sous forme de nombre à virgule.

currentRawValue

Valeur brute mesurée par le capteur (sans arrondi ni calibration), sous forme de nombre à virgule.

logFrequency

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

reportFrequency

Fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques de valeurs sont désactivées.

advMode

Mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

calibrationParam

Paramètres de calibration supplémentaires (par exemple pour compenser l'effet d'un boîtier), sous forme de tableau d'entiers 16 bit.

resolution

Résolution de la mesure (précision de la représentation, mais pas forcément de la mesure elle-même).

sensorState

Etat du capteur (zero lorsque qu'une mesure actuelle est disponible).

enabled

Activation/désactivation de la mesure.

6.7. Temperature

Interface pour intéragir avec les capteurs de température, disponibles par exemple dans le Yocto-Meteo-V2, le Yocto-PT100, le Yocto-Temperature et le Yocto-Thermocouple

La classe `YTemperature` permet de lire et de configurer les capteurs de température Yoctopuce. Elle hérite de la classe `YSensor` toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, elle permet de configurer les paramètres spécifiques de certains types de capteur (type de connection, table d'étalonnage).

logicalName

Chaîne de caractères contenant le nom logique du capteur de température, initialement vide. Cet attribut peut être changé au bon vouloir de l'utilisateur. Un fois initialisé à une valeur non vide, il peut servir de point de départ pour accéder à directement au capteur de température. Si deux capteurs de température portent le même nom logique dans un projet, il n'y a pas moyen de déterminer lequel va répondre si l'on tente un accès par ce nom logique. Le nom logique du module est limité à 19 caractères parmi `A..Z`,`a..z`,`0..9`,`_` et `-`.

advertisedValue

Courte chaîne de caractères résumant l'état actuel du capteur de température, et qui sera publiée automatiquement jusqu'au hub parent. Pour un capteur de température, la valeur publiée est la valeur courante de la température.

unit

Courte chaîne de catactères représentant l'unité dans laquelle la température est exprimée.

currentValue

Valeur actuelle de la température, en degrés Celsius, sous forme de nombre à virgule.

lowestValue

Valeur minimale de la température, en degrés Celsius, sous forme de nombre à virgule.

highestValue

Valeur maximale de la température, en degrés Celsius, sous forme de nombre à virgule.

currentRawValue

Valeur brute mesurée par le capteur (sans arrondi ni calibration), sous forme de nombre à virgule.

logFrequency

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

reportFrequency

Fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques de valeurs sont désactivées.

advMode

Mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

calibrationParam

Paramètres de calibration supplémentaires (par exemple pour compenser l'effet d'un boîtier), sous forme de tableau d'entiers 16 bit.

resolution

Résolution de la mesure (précision de la représentation, mais pas forcément de la mesure elle-même).

sensorState

Etat du capteur (zero lorsque qu'une mesure actuelle est disponible).

sensorType

Type de senseur utilisé pour réaliser la mesure de température. Le senseur peut être digital, un type de thermocouple, un type de PT100, un thermistor ou encore un capteur infrarouge

signalValue

Valeur actuelle du signal électrique mesuré par le capteur (sauf pour les senseurs digitaux) sous forme de nombre à virgule.

signalUnit

Courte chaîne de catactères représentant l'unité du signal électrique utilisé par le capteur.

command

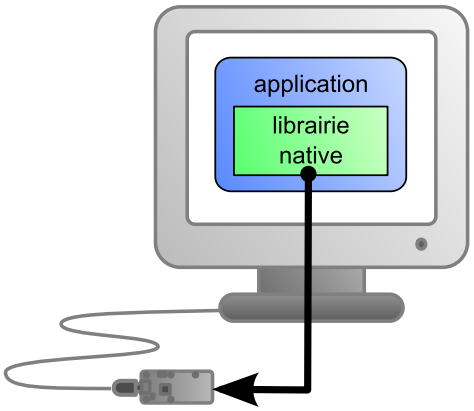
Attribut magique permettant de configurer les paramètres physiques du capteur de température.

6.8. Quelle interface: Native, DLL ou Service?

Il y existe plusieurs méthodes pour contrôler un module USB Yoctopuce depuis un programme.

Contrôle natif

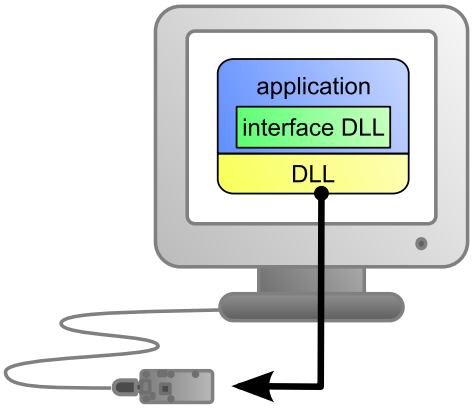
Dans ce cas de figure le programme pilotant votre projet est directement compilé avec une librairie qui offre le contrôle des modules. C'est objectivement la solution la plus simple et la plus élégante pour l'utilisateur final. Il lui suffira de brancher le câble USB et de lancer votre programme pour que tout fonctionne. Malheureusement, cette technique n'est pas toujours disponible ou même possible.



L'application utilise la librairie native pour contrôler le module connecté en local

Contrôle natif par DLL

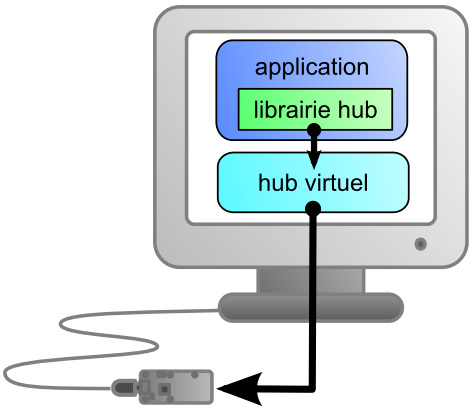
Ici l'essentiel du code permettant de contrôler les modules se trouve dans une DLL, et le programme est compilé avec une petite librairie permettant de contrôler cette DLL. C'est la manière la plus rapide pour coder le support des modules dans un language particulier. En effet la partie "utile" du code de contrôle se trouve dans la DLL qui est la même pour tous les langages, offrir le support pour un nouveau langage se limite à coder la petite librairie qui contrôle la DLL. Du point de de l'utilisateur final, il y a peu de différence: il faut simplement être sur que la DLL sera installée sur son ordinateur en même temps que le programme principal.



L'application utilise la DLL pour contrôler nativement le module connecté en local

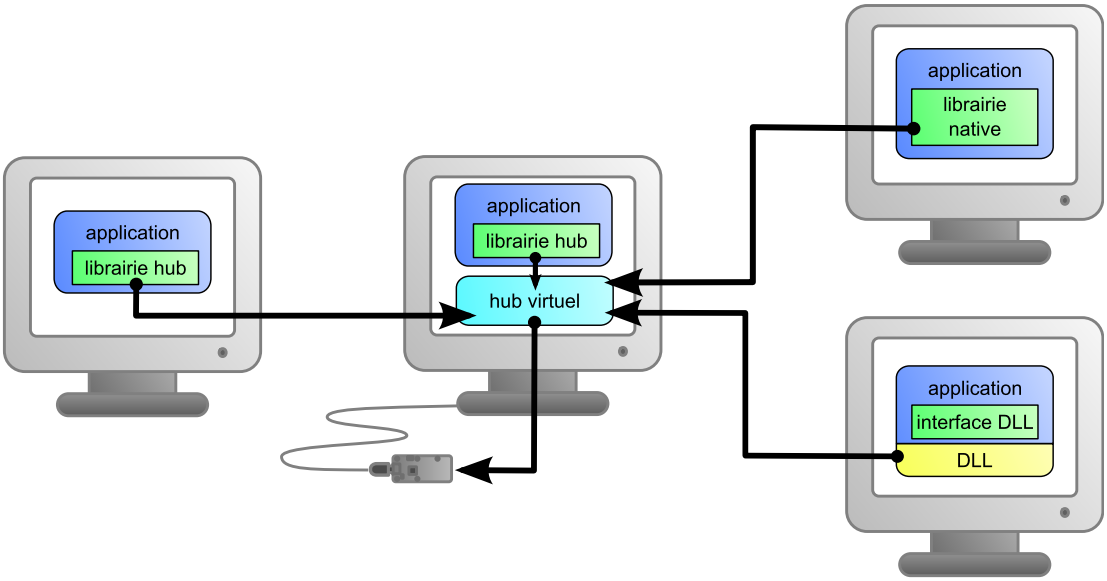
Contrôle par un service

Certain langages ne permettent tout simplement pas d'accéder facilement au niveau matériel de la machine. C'est le cas de Javascript par exemple. Pour gérer ce cas Yoctopuce offre la solution sous la forme d'un petit service, appelé VirtualHub qui lui est capable d'accéder aux modules, et votre application n'a plus qu'à utiliser une librairie qui offrira toutes les fonctions nécessaires au contrôle des modules en passant par l'intermédiaire de ce VirtualHub. L'utilisateur final se verra obligé de lancer le VirtualHub avant de lancer le programme de contrôle du projet proprement dit, à moins qu'il ne décide d'installer le VirtualHub sous la forme d'un service/démon, auquel cas le VirtualHub se lancera automatiquement au démarrage de la machine..



L'application se connecte au service VirtualHub pour connecter le module.

En revanche la méthode de contrôle par un service offre un avantage non négligeable: l'application n'est pas n'obligé de tourner sur la machine où se trouvent les modules: elle peut parfaitement se trouver sur un autre machine qui se connectera au service pour piloter les module. De plus les librairie natives et DLL évoquées plus haut sont aussi capables de se connecter à distance à un ou plusieurs VirtualHub.



Lorsqu'on utilise un VirtualHub, l'application de contrôle n'a plus besoin d'être sur la même machine que le module.

Quel que soit langage de programmation choisi et le paradigme de contrôle utilisé; la programmation reste strictement identique. D'un langage à l'autre les fonctions ont exactement le même nom, prennent les mêmes paramètres. Les seules différences sont liées aux contraintes des langages eux-mêmes.

Language	Natif	Natif avec .DLL/.so	VirtualHub
Ligne de commande	✔	-	✔
Python	-	✔	✔
C++	✔	✔	✔
C# .Net	-	✔	✔
C# UWP	✔	-	✔
LabVIEW	-	✔	✔
Java	-	✔	✔
Java pour Android	✔	-	✔
TypeScript	-	-	✔
JavaScript / ECMAScript	-	-	✔
PHP	-	-	✔
VisualBasic .Net	-	✔	✔
Delphi	-	✔	✔
Objective-C	✔	-	✔

Méthode de support pour les différents langages.

Limitation des librairies Yoctopuce

Les librairies Natives et DLL ont une limitation technique. Sur une même machine, vous ne pouvez pas faire tourner en même temps plusieurs applications qui accèdent nativement aux modules Yoctopuce. Si vous désirez contrôler plusieurs projets depuis la même machine, codez vos applications pour qu'elle accèdent aux modules via un *VirtualHub* plutôt que nativement. Le changement de mode de fonctionnement est trivial: il suffit de changer un paramètre dans l'appel à `yRegisterHub()`.

6.9. Programmation, par où commencer?

Arrivé à ce point du manuel, vous devriez connaître l'essentiel de la théorie à propos de votre Yocto-Motor-DC. Il est temps de passer à la pratique. Il vous faut télécharger la librairie Yoctopuce pour votre language de programmation favori depuis le site web de Yoctopuce[1]. Puis sautez directement au chapitre correspondant au langage de programmation que vous avez choisi.

Tous les exemples décrits dans ce manuel sont présents dans les librairies de programmation. Dans certains langages, les librairies comprennent aussi quelques applications graphiques complètes avec leur code source.

Une fois que vous maîtriserez la programmation de base de votre module, vous pourrez vous intéresser au chapitre concernant la programmation avancée qui décrit certaines techniques qui vous permettront d'exploiter au mieux votre Yocto-Motor-DC.

[1] http://www.yoctopuce.com/FR/libraries.php

7. Utilisation du Yocto-Motor-DC en ligne de commande

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

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

7.1. Installation

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

```
C:\>YTemperature any get_currentValue
```

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

7.2. Utilisation: description générale

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

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

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

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

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

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

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

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

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

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

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

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

7.3. Contrôle de la fonction Motor

Pour contrôler la fonction Motor de votre Yocto-Motor-DC, vous avez besoin de l'exécutable YMotor.

Vous pouvez par exemple lancer:

```
C:\>YTemperature any get_currentValue
```

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

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

```
C:\>YMotor MOTORCTL-123456.motor describe

C:\>YMotor MOTORCTL-123456.MaFonction describe

C:\>YMotor MonModule.motor describe

C:\>YMotor MonModule.MaFonction describe

C:\>YMotor MaFonction describe
```

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

```
C:\>YMotor all describe
```

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

```
C:\>YMotor /help
```

7.4. Contrôle de la partie module

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

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

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

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

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

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

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

Modifications des réglages du module

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

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

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

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

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

7.5. Limitations

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

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

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

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

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

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

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

8. Utilisation du Yocto-Motor-DC en Python

Python est un langage interprété orienté objet développé par Guido van Rossum. Il offre l'avantage d'être gratuit et d'être disponible pour la plupart de plate-formes tant Windows qu'Unix. C'est un language idéal pour écrire des petits scripts sur un coin de table. La librairie Yoctopuce est compatible avec Python 2.6+ et 3+. Elle fonctionne sous Windows, Max OS X et Linux tant Intel qu'ARM. La librairie a été testée avec Python 2.6 et Python 3.2. Les interpréteurs Python sont disponibles sur le site de Python [1].

8.1. Fichiers sources

Les classes de la librairie Yoctopuce[2] pour Python que vous utiliserez vous sont fournies au format source. Copiez tout le contenu du répertoire *Sources* dans le répertoire de votre choix et ajoutez ce répertoire à la variable d'environnement *PYTHONPATH*. Si vous utilisez un IDE pour programmer en Python, référez-vous à sa documentation afin le configurer de manière à ce qu'il retrouve automatiquement les fichiers sources de l'API.

8.2. Librairie dynamique

Une partie de la librairie de bas-niveau est écrite en C, mais vous n'aurez a priori pas besoin d'interagir directement avec elle: cette partie est fournie sous forme de DLL sous Windows, de fichier *.so* sous Unix et de fichier *.dylib* sous Mac OS X. Tout a été fait pour que l'interaction avec cette librairie se fasse aussi simplement que possible depuis Python: les différentes versions de la librairie dynamique correspondant aux différents systèmes d'exploitation et architectures sont stockées dans le répertoire *cdll*. L'API va charger automatiquement le bon fichier lors de son initialisation. Vous n'aurez donc pas à vous en soucier.

Si un jour vous deviez vouloir recompiler la librairie dynamique, vous trouverez tout son code source dans la librairie Yoctopuce pour le C++.

Afin de les garder simples, tous les exemples fournis dans cette documentation sont des applications consoles. Il va de soit que que le fonctionnement des librairies est strictement identiques si vous les intégrez dans une application dotée d'une interface graphique.

8.3. Contrôle de la fonction Motor

[1] http://www.python.org/download/
[2] www.yoctopuce.com/FR/libraries.php

Il suffit de quelques lignes de code pour piloter un Yocto-Motor-DC. Voici le squelette d'un fragment de code Python qui utilise la fonction Motor.

```
[...]
# On active la détection des modules sur USB
errmsg=YRefParam()
YAPI.RegisterHub("usb",errmsg)
[...]

# On récupère l'objet permettant d'intéragir avec le module
motor = YMotor.FindMotor("MOTORCTL-123456.motor")

# Pour gérer le hot-plug, on vérifie que le module est là
if motor.isOnline():
    # use motor.set_drivingForce()
    [...]

[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

YAPI.RegisterHub

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

YMotor.FindMotor

La fonction `YMotor.FindMotor` permet de retrouver un moteur 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-Motor-DC avec le numéros de série *MOTORCTL-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *motor* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
motor = YMotor.FindMotor("MOTORCTL-123456.motor")
motor = YMotor.FindMotor("MOTORCTL-123456.MaFonction")
motor = YMotor.FindMotor("MonModule.motor")
motor = YMotor.FindMotor("MonModule.MaFonction")
motor = YMotor.FindMotor("MaFonction")
```

`YMotor.FindMotor` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le moteur.

isOnline

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

set_drivingForce

La méthode `set_drivingForce()` de l'objet renvoyé par `YMotor.FindMotor` permet de régler la puissance, de -100 à +100% envoyée au moteur.

Un exemple réel

Lancez votre interpréteur Python et ouvrez le script correspondant, fourni dans le répertoire **Examples/Doc-GettingStarted-Yocto-Motor-DC** de la librairie Yoctopuce.

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

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os, sys
```

```
from yocto_api import *
from yocto_motor import *
from yocto_current import *
from yocto_voltage import *
from yocto_temperature import *


def usage():
    scriptname = os.path.basename(sys.argv[0])
    print("Usage:")
    print(scriptname + ' <serial_number> power')
    print(scriptname + ' <logical_name>power')
    print(scriptname + ' any <channel> power')
    print('power is an integer between -100 and 100%')
    print('Example:')
    print(scriptname + ' any 75')
    sys.exit()


def die(msg):
    sys.exit(msg + ' (check USB cable)')


# parse the command line


if len(sys.argv) < 3:
    usage()
target = sys.argv[1].upper()
power = int(sys.argv[2])

# Setup the API to use local USB devices
errmsg = YRefParam()
if YAPI.RegisterHub("usb", errmsg) != YAPI.SUCCESS:
    sys.exit("init error" + errmsg.value)

if target == 'ANY':
    # find any motor then retreive its serial #
    motor = YMotor.FirstMotor()
    if motor is None:
        die('No module connected')
    m = motor.get_module()
    target = m.get_serialNumber()
    print('using ' + target)

motor = YMotor.FindMotor(target + '.motor')
current = YCurrent.FindCurrent(target + '.current')
voltage = YVoltage.FindVoltage(target + '.voltage')
temperature = YTemperature.FindTemperature(target + '.temperature')

if motor.isOnline():
    # if the motor is in error state, reset it.
    if motor.get_motorStatus() >= YMotor.MOTORSTATUS_LOVOLT:  motor.resetStatus()
    motor.drivingForceMove(power, 2000)  # ramp up to power in 2 seconds
    while motor.isOnline():
        print("Status :  " + motor.get_advertisedValue() +
              " Current : " + "%2.1f" % (current.get_currentValue() / 1000) + "A  " + \
              "Voltage : " + "%2.1f" % (voltage.get_currentValue()) + "V  " + \
              "Temperature : " + "%2.1f" % (temperature.get_currentValue()) + "deg C")
        YAPI.Sleep(1000, errmsg)
else:
    die('device not connected')
YAPI.FreeAPI()
```

8.4. Contrôle de la partie module

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

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

```
import os, sys

from yocto_api import *


def usage():
    sys.exit("usage: demo <serial or logical name> [ON/OFF]")


errmsg = YRefParam()
if YAPI.RegisterHub("usb", errmsg) != YAPI.SUCCESS:
    sys.exit("RegisterHub error: " + str(errmsg))

if len(sys.argv) < 2:
    usage()

m = YModule.FindModule(sys.argv[1])  # # use serial or logical name

if m.isOnline():
    if len(sys.argv) > 2:
        if sys.argv[2].upper() == "ON":
            m.set_beacon(YModule.BEACON_ON)
        if sys.argv[2].upper() == "OFF":
            m.set_beacon(YModule.BEACON_OFF)

    print("serial:       " + m.get_serialNumber())
    print("logical name: " + m.get_logicalName())
    print("luminosity:   " + str(m.get_luminosity()))
    if m.get_beacon() == YModule.BEACON_ON:
        print("beacon:       ON")
    else:
        print("beacon:       OFF")
    print("upTime:       " + str(m.get_upTime() / 1000) + " sec")
    print("USB current:  " + str(m.get_usbCurrent()) + " mA")
    print("logs:\n" + m.get_lastLogs())
else:
    print(sys.argv[1] + " not connected (check identification and USB cable)")
YAPI.FreeAPI()
```

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

Modifications des réglages du module

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

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os, sys

from yocto_api import *


def usage():
    sys.exit("usage: demo <serial or logical name> <new logical name>")


if len(sys.argv) != 3:
    usage()

errmsg = YRefParam()
if YAPI.RegisterHub("usb", errmsg) != YAPI.SUCCESS:
    sys.exit("RegisterHub error: " + str(errmsg))
```

```
m = YModule.FindModule(sys.argv[1])  # use serial or logical name
if m.isOnline():
    newname = sys.argv[2]
    if not YAPI.CheckLogicalName(newname):
        sys.exit("Invalid name (" + newname + ")")
    m.set_logicalName(newname)
    m.saveToFlash()  # do not forget this
    print("Module: serial= " + m.get_serialNumber() + " / name= " + m.get_logicalName())
else:
    sys.exit("not connected (check identification and USB cable")
YAPI.FreeAPI()
```

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 `YModule.saveToFlash()` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

Enumeration des modules

Obtenir la liste des modules connectés se fait à l'aide de la fonction `YModule.yFirstModule()` qui renvoie le premier module trouvé, il suffit ensuite d'appeler la mehode `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

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os, sys


from yocto_api import *

errmsg = YRefParam()

# Setup the API to use local USB devices
if YAPI.RegisterHub("usb", errmsg) != YAPI.SUCCESS:
    sys.exit("init error" + str(errmsg))

print('Device list')

module = YModule.FirstModule()
while module is not None:
    print(module.get_serialNumber() + ' (' + module.get_productName() + ')')
    module = module.nextModule()
YAPI.FreeAPI()
```

8.5. Gestion des erreurs

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

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

La méthode recommandée par la plupart des langages de programmation pour la gestion des erreurs imprévisibles est l'utilisation d'exceptions. C'est le comportement par défaut de la librairie

Yoctopuce. Si une erreur se produit alors qu'on essaie d'accéder à un module, la librairie va lancer une exception. Dans ce cas, de trois choses l'une:

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

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `YAPI.DisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur `NomDeClasse.STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `NomDeClasse.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.

9. Utilisation du Yocto-Motor-DC 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.

9.1. Contrôle de la fonction Motor

Il suffit de quelques lignes de code pour piloter un Yocto-Motor-DC. Voici le squelette d'un fragment de code C++ qui utilise la fonction Motor.

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

[...]
// On active la détection des modules sur USB
String errmsg;
```

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

```
YAPI::RegisterHub("usb", errmsg);
[...]

// On récupère l'objet permettant d'intéragir avec le module
YMotor *motor;
motor = YMotor::FindMotor("MOTORCTL-123456.motor");

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.h et yocto_motor.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_motor.h` est nécessaire pour gérer les modules contenant un moteur, comme le Yocto-Motor-DC.

YAPI::RegisterHub

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

YMotor::FindMotor

La fonction `YMotor::FindMotor` permet de retrouver un moteur 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-Motor-DC avec le numéros de série *MOTORCTL-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *motor* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YMotor *motor = YMotor::FindMotor("MOTORCTL-123456.motor");
YMotor *motor = YMotor::FindMotor("MOTORCTL-123456.MaFonction");
YMotor *motor = YMotor::FindMotor("MonModule.motor");
YMotor *motor = YMotor::FindMotor("MonModule.MaFonction");
YMotor *motor = YMotor::FindMotor("MaFonction");
```

`YMotor::FindMotor` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le moteur.

isOnline

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

set_drivingForce

La méthode `set_drivingForce()` de l'objet renvoyé par `yFindMotor` permet de régler la puissance, de -100 à +100% envoyée au moteur.

Un exemple réel

Lancez votre environnement C++ et ouvrez le projet exemple correspondant, fourni dans le répertoire **Examples/Doc-GettingStarted-Yocto-Motor-DC** 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_motor.h"
#include "yocto_temperature.h"
#include "yocto_current.h"
#include "yocto_voltage.h"
#include <iostream>
#include <ctype.h>
#include <stdlib.h>

using namespace std;

static void usage(void)
{
  cout << "usage: demo <serial_number> power" << endl;
  cout << "       demo <logical_name>  power" << endl;
  cout << "       demo any power       (use any discovered device)" << endl;
  cout << "       power is a integer between -100 and 100%%" << endl;
  cout << "Example:" << endl;
  cout << "       demo any 75" << endl;

  u64 now = YAPI::GetTickCount();
  while (YAPI::GetTickCount() - now < 3000) {
    // wait 3 sec to show the message
  }
  exit(1);
}

int main(int argc, const char * argv[])
{
  string  target, errmsg;
  int power;
  YMotor *motor;
  YCurrent *current;
  YVoltage *voltage;
  YTemperature *temperature;

  // parse command line
  if (argc < 3) {
    usage();
  }
  target = (string) argv[1];
  power = atoi(argv[2]);

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

  if (target == "any") {
    // find the serial# of the first available motor
    motor =  YMotor::FirstMotor();
    if (motor == NULL) {
      cout << "No module connected (check USB cable)" << endl;
      return 1;
    }
    target = motor->get_module()->get_serialNumber();
  }

  // retreive motor, current, voltage and temperature features from the device
  motor       = YMotor::FindMotor(target + ".motor");
  current     = YCurrent::FindCurrent(target + ".current");
  voltage     = YVoltage::FindVoltage(target + ".voltage");
  temperature = YTemperature::FindTemperature(target + ".temperature");

  // lets start the motor
  if (motor->isOnline()) {
    // if motor is in error state, reset it.
    if (motor->get_motorStatus() >= YMotor::MOTORSTATUS_LOVOLT) {
      motor->resetStatus();
    }
    motor->drivingForceMove(power, 2000); // ramp up to power in 2 seconds
    while (motor->isOnline()) {
      // display motor status
      cout << "Status=" << motor->get_advertisedValue() << "  "
           << "Voltage=" << voltage->get_currentValue() << "V  "
           << "Current=" << current->get_currentValue() / 1000 << "A  "
           << "Temp=" << temperature->get_currentValue() << "deg C" << endl;
```

```
      YAPI::Sleep(1000, errmsg); // wait for one second
    }
  } else {
    cout << "Module not connected (check identification and USB cable)" << endl;
  }
  YAPI::FreeAPI();
  return 0;
}
```

9.2. 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(YAPI::RegisterHub("usb", errmsg) != YAPI::SUCCESS) {
    cerr << "RegisterHub error: " << errmsg << endl;
    return 1;
  }

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

  YModule *module = YModule::FindModule(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;
    cout << "Logs:" << endl << module->get_lastLogs() << endl;
  } else {
    cout << argv[1] << " not connected (check identification and USB cable)"
         << endl;
  }
  YAPI::FreeAPI();
  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(YAPI::RegisterHub("usb", errmsg) != YAPI::SUCCESS) {
    cerr << "RegisterHub error: " << errmsg << endl;
    return 1;
  }

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

  YModule *module = YModule::FindModule(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;
  }
  YAPI::FreeAPI();
  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(YAPI::RegisterHub("usb", errmsg) != YAPI::SUCCESS) {
    cerr << "RegisterHub error: " << errmsg << endl;
    return 1;
  }

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

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

9.3. 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 `YAPI.DisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur `NomDeClasse.STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `NomDeClasse.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.

9.4. 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 (recommandé)

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 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 utiliser la librairie statique, il faut la compiler à l'aide du shell script `build.sh` sous UNIX, ou `build.bat` sous Windows. Ce script qui se situe à la racine de la librairie, détecte l'OS et recompile toutes les librairies ainsi que les exemples correspondants.

Ensuite, 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**.

Finalement, 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 utiliser la librairie dynamique, il faut la compiler à l'aide du shell script `build.sh` sous UNIX, ou `build.bat` sous Windows. Ce script qui se situe à la racine de la librairie, détecte l'OS et recompile toutes les librairies ainsi que les exemples correspondant.

Ensuite, 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**.

Finalement, 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++
```

10. Utilisation du Yocto-Motor-DC en C#

C# (prononcez C-Sharp) est un langage orienté objet promu par Microsoft qui n'est pas sans rappeller Java. Tout comme Visual Basic et Delphi, il permet de créer des applications Windows relativement facilement. Tous les exemples et les modèles de projet sont testés avec Microsoft C# 2010 Express, disponible gratuitement sur le site de Microsoft [1].

Notre librairie est aussi compatible avec *Mono*, la version open source de C# qui fonctionne sous Linux et MacOS. Vous trouverez sur notre site web différents articles qui décrivent comment indiquer à Mono comment accéder à notre librairie.

10.1. Installation

Téléchargez la librairie Yoctopuce pour Visual C# depuis le site web de Yoctopuce[2]. Il n'y a pas de programme d'installation, copiez simplement de contenu du fichier zip dans le répertoire de votre choix. Vous avez besoin essentiellement du contenu du répertoire `Sources`. Les autres répertoires contiennent la documentation et quelques programmes d'exemple. Les projets d'exemple sont des projets Visual C# 2010, si vous utilisez une version antérieure, il est possible que vous ayez à reconstruire la structure de ces projets.

10.2. Utilisation l'API yoctopuce dans un projet Visual C#

La librairie Yoctopuce pour Visual C# .NET se présente sous la forme d'une DLL et de fichiers sources en Visual C#. La DLL n'est pas une DLL .NET mais une DLL classique, écrite en C, qui gère les communications à bas niveau avec les modules[3]. Les fichiers sources en Visual C# gèrent la partie haut niveau de l'API. Vous avez donc besoin de cette DLL et des fichiers .cs du répertoire `Sources` pour créer un projet gérant des modules Yoctopuce.

Configuration d'un projet Visual C#

Les indications ci-dessous sont fournies pour Visual Studio express 2010, mais la procédure est semblable pour les autres versions.

Commencez par créer votre projet, puis depuis le panneau **Explorateur de solutions** effectuez un clic droit sur votre projet, et choisissez **Ajouter** puis **Elément existant**.

[1] http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-csharp-express
[2] www.yoctopuce.com/FR/libraries.php
[3] Les sources de cette DLL sont disponibles dans l'API C++

Une fenêtre de sélection de fichiers apparaît: sélectionnez le fichier `yocto_api.cs` et les fichiers correspondant aux fonctions des modules Yoctopuce que votre projet va gérer. Dans le doute, vous pouvez aussi sélectionner tous les fichiers.

Vous avez alors le choix entre simplement ajouter ces fichiers à votre projet, ou les ajouter en tant que lien (le bouton **Ajouter** est en fait un menu déroulant). Dans le premier cas, Visual Studio va copier les fichiers choisis dans votre projet, dans le second Visual Studio va simplement garder un lien sur les fichiers originaux. Il est recommandé d'utiliser des liens, une éventuelle mise à jour de la librairie sera ainsi beaucoup plus facile.

Ensuite, ajoutez de la même manière la dll `yapi.dll`, qui se trouve dans le répertoire `Sources/dll`[4]. Puis depuis la fenêtre **Explorateur de solutions**, effectuez un clic droit sur la DLL, choisissez **Propriété** et dans le panneau **Propriétés**, mettez l'option **Copier dans le répertoire de sortie** à **toujours copier**. Vous êtes maintenant prêt à utiliser vos modules Yoctopuce depuis votre environnement Visual Studio.

Afin de 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.

10.3. Contrôle de la fonction Motor

Il suffit de quelques lignes de code pour piloter un Yocto-Motor-DC. Voici le squelette d'un fragment de code C# qui utilise la fonction Motor.

```
[...]
// On active la détection des modules sur USB
string errmsg = "";
YAPI.RegisterHub("usb", errmsg);
[...]

// On récupère l'objet permettant d'intéragir avec le module
YMotor motor = YMotor.FindMotor("MOTORCTL-123456.motor");

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

Voyons maintenant en détail ce que font ces quelques lignes.

YAPI.RegisterHub

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

YMotor.FindMotor

La fonction `YMotor.FindMotor` permet de retrouver un moteur 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-Motor-DC avec le numéros de série *MOTORCTL-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *motor* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
motor = YMotor.FindMotor("MOTORCTL-123456.motor");
motor = YMotor.FindMotor("MOTORCTL-123456.MaFonction");
motor = YMotor.FindMotor("MonModule.motor");
```

[4] Pensez à changer le filtre de la fenêtre de sélection de fichiers, sinon la DLL n'apparaîtra pas

```
motor = YMotor.FindMotor("MonModule.MaFonction");
motor = YMotor.FindMotor("MaFonction");
```

`YMotor.FindMotor` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le moteur.

isOnline

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

set_drivingForce

La méthode `set_drivingForce()` de l'objet renvoyé par `YMotor.FindMotor` permet de régler la puissance, de -100 à +100% envoyée au moteur.

Un exemple réel

Lancez Visual C# et ouvrez le projet exemple correspondant, fourni dans le répertoire **Examples/Doc-GettingStarted-Yocto-Motor-DC** de la librairie Yoctopuce.

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

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
  class Program
  {
    static void usage()
    {
      string execname = System.AppDomain.CurrentDomain.FriendlyName;
      Console.WriteLine("Usage:");
      Console.WriteLine(execname + "  <serial_number>  power");
      Console.WriteLine(execname + "  <logical_name>  power");
      Console.WriteLine(execname + "  any power");
      Console.WriteLine("  power is a integer between -100 and 100%");
      Console.WriteLine("Example:");
      Console.WriteLine(execname + "  any 75");
      System.Threading.Thread.Sleep(2500);
      Environment.Exit(0);
    }

    static void Main(string[] args)
    {
      string errmsg = "";
      string target;
      int power;
      YMotor motor;
      YCurrent current;
      YVoltage voltage;
      YTemperature temperature;

      // parse the  command line
      if (args.Length < 2) usage();
      target = args[0].ToUpper();
      power = Convert.ToInt32(args[1]);

      if (YAPI.RegisterHub("usb", ref errmsg) != YAPI.SUCCESS) {
        Console.WriteLine("RegisterHub error: " + errmsg);
        Environment.Exit(0);
      }
      if (target == "ANY") {
        // find the serial# of the first available motor
        motor = YMotor.FirstMotor();
        if (motor == null) {
          Console.WriteLine("No module connected (check USB cable) ");
          Environment.Exit(0);
        }
        target = motor.get_module().get_serialNumber();
```

```
      }

      motor = YMotor.FindMotor(target + ".motor");
      current = YCurrent.FindCurrent(target + ".current");
      voltage = YVoltage.FindVoltage(target + ".voltage");
      temperature = YTemperature.FindTemperature(target + ".temperature");

      // lets start the motor
      if (motor.isOnline()) {
        // if motor is in error state, reset it.
        if (motor.get_motorStatus() >= YMotor.MOTORSTATUS_LOVOLT) {
          motor.resetStatus();
        }
        motor.drivingForceMove(power, 2000);  // ramp up to power in 2 seconds
        while (motor.isOnline()) {
          // display motor status
          Console.WriteLine("Status=" + motor.get_advertisedValue() + "   " +
                            "Voltage=" + voltage.get_currentValue() + "V   " +
                            "Current=" + current.get_currentValue() / 1000 + "A   " +
                            "Temp=" + temperature.get_currentValue() + "deg C");
          YAPI.Sleep(1000, ref errmsg); // wait for one second
        }
      }
      YAPI.FreeAPI();
    }
  }
}
```

10.4. Contrôle de la partie module

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

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;


namespace ConsoleApplication1
{
  class Program
  {
    static void usage()
    {
      string execname = System.AppDomain.CurrentDomain.FriendlyName;
      Console.WriteLine("Usage:");
      Console.WriteLine(execname + " <serial or logical name> [ON/OFF]");
      System.Threading.Thread.Sleep(2500);
      Environment.Exit(0);
    }

    static void Main(string[] args)
    {
      YModule m;
      string errmsg = "";

      if (YAPI.RegisterHub("usb", ref errmsg) !=  YAPI.SUCCESS) {
        Console.WriteLine("RegisterHub error: " + errmsg);
        Environment.Exit(0);
      }


      if (args.Length < 1)  usage();

      m = YModule.FindModule(args[0]); // use serial or logical name

      if (m.isOnline()) {
        if (args.Length >= 2) {
          if (args[1].ToUpper() == "ON") {
            m.set_beacon(YModule.BEACON_ON);
          }
```

```
          if (args[1].ToUpper() == "OFF") {
            m.set_beacon(YModule.BEACON_OFF);
          }
        }

        Console.WriteLine("serial:       " + m.get_serialNumber());
        Console.WriteLine("logical name: " + m.get_logicalName());
        Console.WriteLine("luminosity:   " + m.get_luminosity().ToString());
        Console.Write("beacon:       ");
        if (m.get_beacon() == YModule.BEACON_ON)
          Console.WriteLine("ON");
        else
          Console.WriteLine("OFF");
        Console.WriteLine("upTime:       " + (m.get_upTime() / 1000 ).ToString() + " sec");
        Console.WriteLine("USB current:  " + m.get_usbCurrent().ToString() + " mA");
        Console.WriteLine("Logs:\r\n" + m.get_lastLogs());

      } else {
        Console.WriteLine(args[0] + " not connected (check identification and USB cable)");
      }
      YAPI.FreeAPI();
    }
  }
}
```

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

Modifications des réglages du module

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

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
  class Program
  {
    static void usage()
    {
      string execname = System.AppDomain.CurrentDomain.FriendlyName;
      Console.WriteLine("Usage:");
      Console.WriteLine("usage: demo <serial or logical name> <new logical name>");
      System.Threading.Thread.Sleep(2500);
      Environment.Exit(0);
    }

    static void Main(string[] args)
    {
      YModule m;
      string errmsg = "";
      string newname;

      if (args.Length != 2) usage();

      if (YAPI.RegisterHub("usb", ref errmsg) !=  YAPI.SUCCESS) {
        Console.WriteLine("RegisterHub error: " + errmsg);
        Environment.Exit(0);
      }

      m = YModule.FindModule(args[0]); // use serial or logical name
```

```
      if (m.isOnline()) {
        newname = args[1];
        if (!YAPI.CheckLogicalName(newname)) {
          Console.WriteLine("Invalid name (" + newname + ")");
          Environment.Exit(0);
        }

        m.set_logicalName(newname);
        m.saveToFlash(); // do not forget this

        Console.Write("Module: serial= " + m.get_serialNumber());
        Console.WriteLine(" / name= " + m.get_logicalName());
      } else {
        Console.Write("not connected (check identification and USB cable");
      }
      YAPI.FreeAPI();
    }
  }
}
```

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 `YModule.saveToFlash()` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

Enumeration des modules

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

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      YModule m;
      string errmsg = "";

      if (YAPI.RegisterHub("usb", ref errmsg) !=  YAPI.SUCCESS) {
        Console.WriteLine("RegisterHub error: " + errmsg);
        Environment.Exit(0);
      }

      Console.WriteLine("Device list");
      m = YModule.FirstModule();
      while (m != null) {
        Console.WriteLine(m.get_serialNumber() + " (" + m.get_productName() + ")");
        m = m.nextModule();
      }
      YAPI.FreeAPI();
    }
  }
}
```

10.5. Gestion des erreurs

Lorsque vous implémentez un programme qui doit interagir avec des modules USB, vous ne pouvez pas faire abstraction de la gestion des erreurs. Il y aura forcément une occasion où un utilisateur

aura débranché le périphérique, soit avant de lancer le programme, soit même en pleine opération. La librairie Yoctopuce est prévue pour vous aider à supporter ce genre de comportements, mais votre code doit néanmoins être fait pour se comporter au mieux pour interpréter les erreurs signalées par la librairie.

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

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

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

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `YAPI.DisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur `NomDeClasse.STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `NomDeClasse.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.

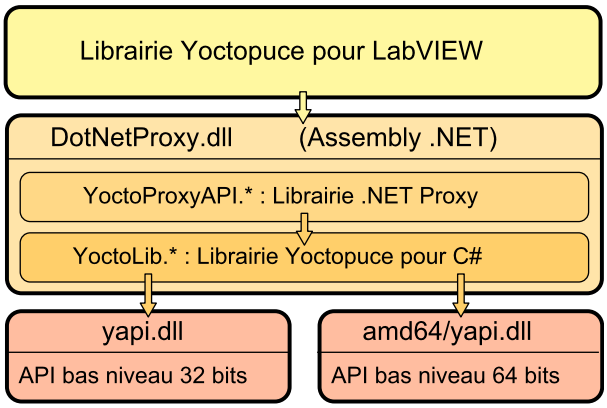
Quand vous travaillez sans les exceptions, il est possible d'obtenir un code d'erreur et un message expliquant l'origine de l'erreur en le demandant à l'objet qui a retourné une erreur à l'aide des méthodes `errType()` et `errMessage()`. Ce sont les même informations qui auraient été associées à l'exception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

11. Utilisation du Yocto-Motor-DC avec LabVIEW

LabVIEW est édité par National Instruments depuis 1986. C'est un environnement de développement graphique: plutôt que d'écrire des lignes de code, l'utilisateur dessine son programme, un peu comme un organigramme. LabVIEW est surtout pensé pour interfacer des instruments de mesures d'où le nom *Virtual Instruments* (VI) des programmes LabVIEW. Avec la programmation visuelle, dessiner des algorithmes complexes devient très vite fastidieux, c'est pourquoi la librairie Yoctopuce pour LabVIEW a été pensée pour être aussi simple de possible à utiliser. Autrement dit, LabVIEW étant un environnement extrêmement différent des autres langages supportés par l'API Yoctopuce, vous rencontrerez des différences majeures entre l'API LabVIEW et les autres API.

11.1. Architecture

La librairie LabVIEW est basée sur la librairie Yoctopuce DotNetProxy contenue dans la DLL DotNetProxyLibrary.dll. C'est en fait cette librairie DotNetProxy qui se charge du gros du travail en s'appuyant sur la librairie Yoctopuce C# qui, elle, utilise l'API bas niveau codée dans yapi.dll (32bits) et amd64\yapi.dll (64bits).



Architecture de l'API Yoctopuce pour LabVIEW

Vos applications LabVIEW utilisant l'API Yoctopuce devront donc impérativement être distribuées avec les DLL *DotNetProxyLibrary.dll*, *yapi.dll* et *amd64\yapi.dll*

Si besoin est, vous trouverez les sources de l'API bas niveau dans la librairie C# et les sources de *DotNetProxyLibrary.dll* dans la librairie *DotNetProxy*.

11.2. Compatibilité

Firmwares

Pour que la librairie Yoctopuce pour LabVIEW fonctionne convenablement avec vos modules Yoctopuce, ces derniers doivent avoir au moins le firmware 37120

LabVIEW pour Linux et MacOS

Au moment de l'écriture de ce manuel, l'API Yoctopuce pour LabVIEW n'a été testée que sous Windows. Il y a donc de fortes chances pour qu'elle ne fonctionne tout simplement pas avec les versions Linux et MacOS de LabVIEW.

LabVIEW NXG

La librairie Yoctopuce pour LabVIEW faisant appel à de nombreuses techniques qui ne sont pas encore disponibles dans la nouvelle génération de LabVIEW, elle n'est absolument pas compatible avec LabVIEW NXG.

A propos de DotNetProxyLibrary.dll

Afin d'être compatible avec un maximum de version de Windows, y compris Windows XP, la librairie *DotNetProxyLibrary.dll* est compilée en .NET 3.5, qui est disponible par défaut sur toutes les versions de Windows depuis XP.

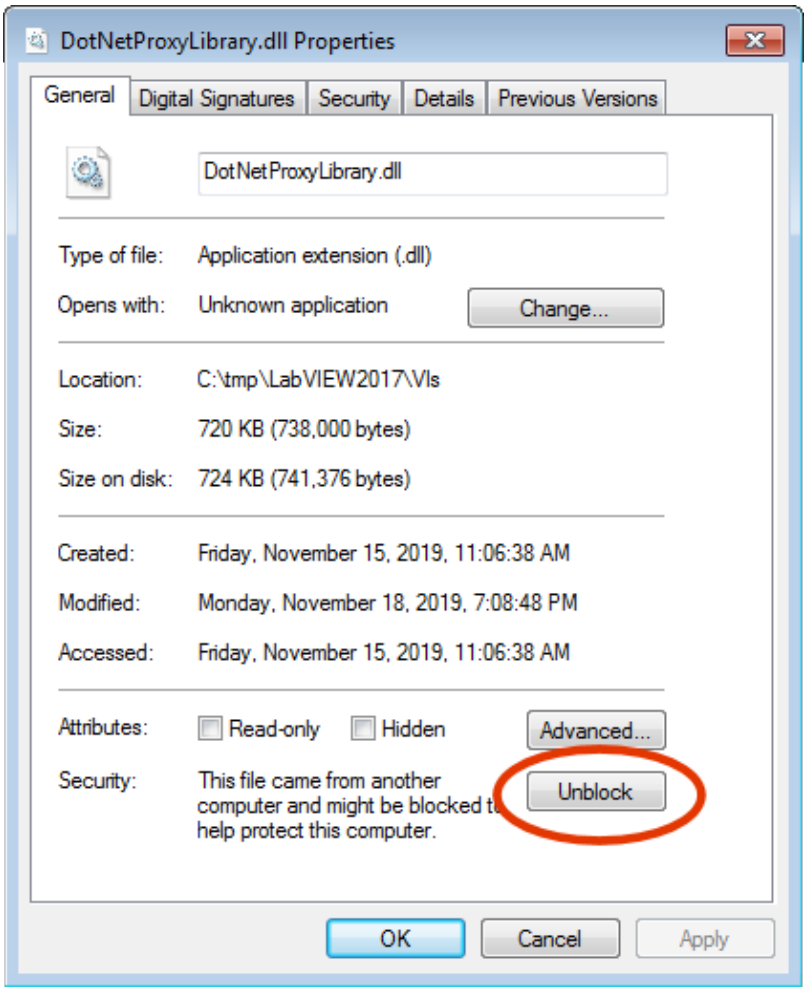
11.3. Installation

Téléchargez la librairie pour LabVIEW depuis le site web de Yoctopuce[1]. Il s'agit d'un fichier ZIP dans lequel vous trouverez un répertoire par version de LabVIEW. Chacun de ses répertoires contient deux sous-répertoires. Le premier contient des exemples de programmation pour chaque produit Yoctopuce; le second, nommé *VIs*, contient tous les VI de l'API et les DLL nécessaires.

Suivant la configuration de Windows et la méthode utilisée pour la copier, la DLL *DotNetProxyLibrary.dll* peut se faire bloquer par Windows parce que ce dernier aura détecté qu'elle provient d'une autre machine. Un cas typique est la décompression de l'archive de la librairie avec l'explorateur de fichier de Windows. Si la DLL est bloquée, LabVIEW ne pourra pas la charger, ce qui entrainera une erreur 1386 lors de l'exécution de n'importe quel VI de la librairie Yoctopuce.

Il y a deux manières de corriger le problème. La plus simple consiste à utiliser l'explorateur de fichier de Windows pour afficher les propriétés de la DLL et la débloquer. Mais cette manipulation devra être répété à chaque fois qu'une nouvelle version de la DLL sera copiée sur votre système.

[1] http://www.yoctopuce.com/FR/libraries.php



Débloquer la DLL DotNetProxyLibrary.dll.

La seconde méthode consiste à créer dans le même répertoire que l'exécutable labview.exe un fichier XML nommé *labview.exe.config* et contenant le code suivant :

```
<?xml version ="1.0"?>
<configuration>
 <runtime>
 <loadFromRemoteSources enabled="true" />
 </runtime>
</configuration>
```

Veillez à choisir le bon répertoire en fonction de la version de LabVIEW que vous utilisez (32 bits vs 64 bits). Vous trouverez plus d'information à propos de ce fichier sur le site web de National Instrument[2].

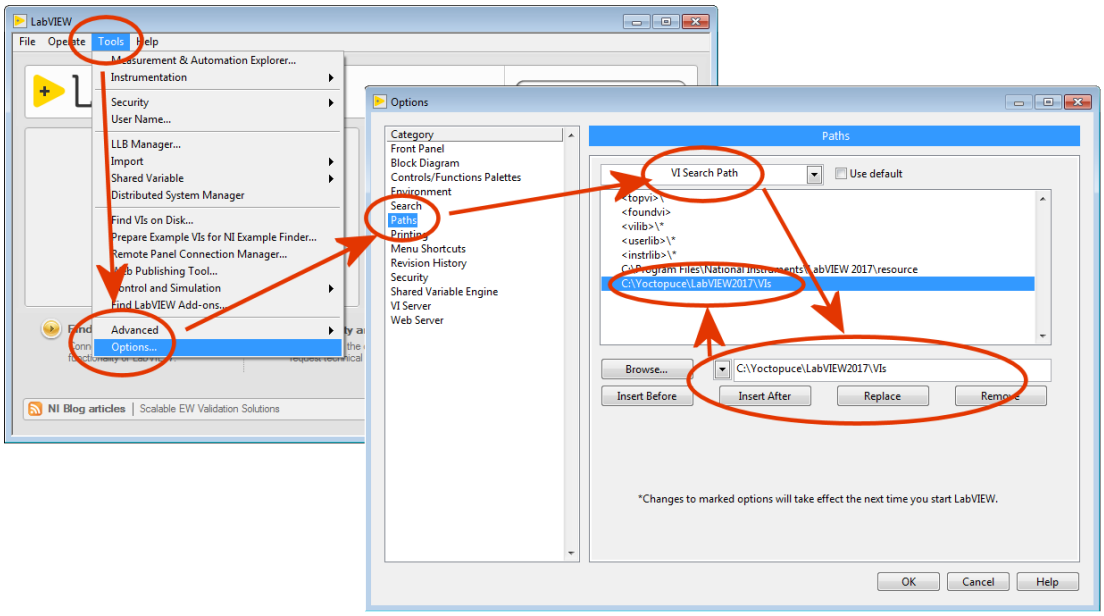
Pour installer l'API Yoctopuce pour LabVIEW vous avez plusieurs méthodes à votre disposition.

Méthode 1 : Installation "à l'emporter"

La manière la plus simple pour installer la librairie Yoctopuce consiste à copier le contenu du répertoire *VIs* où bon vous semble, et à utiliser les VIs dans LabVIEW avec une simple opération de *Drag and Drop*.

Pour pouvoir utiliser les exemples fournis avec l'API, vous aurez avantage à ajouter le répertoire des VIs Yoctopuce dans la liste des répertoires où LabVIEW doit chercher les VIs qu'il n'a pas trouvé. Cette liste est accessible via le menu *Tools* > *Options* > *Paths* > *VI Search Path*.

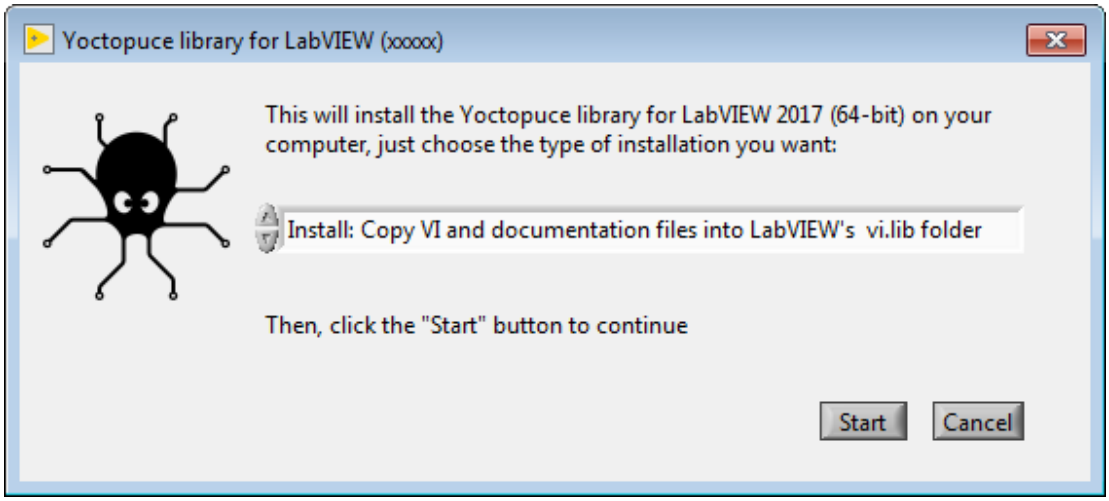
[2] https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P8XnSAK



Configuration du "VI Search Path"

Méthode 2 : Installeur fourni avec la librairie

Dans chaque répertoire LabVIEW200xx de la librairie, vous trouverez un VI appelé "*Install.vi*". Ouvrez simplement celui qui correspond à votre version de LabVIEW.



L'installeur fourni avec la librairie

Cet installeur offre trois options d'installation:

Install: Keep VI and documentation files where they are.
Avec cette option, les VI sont conservés à l'endroit où la librairie à été décompressée. Vous aurez donc à faire en sorte qu'ils ne soit pas effacés tant que vous en aurez besoin. Voici ce que fait exactement l'installeur quand cette option est choisie:

- Toute référence à des répertoires contenant une version quelconque de la librairie Yoctopuce sont supprimés de l'option *viSearchPath* dans le fichier *labview.ini*.
- Un fichier de palette *dir.mnu* référençant les VIs est créé dans le répertoire: *C:\Program Files xx\National Instruments\LabVIEW 20xx\vi.lib\addons\Yoctopuce*
- Une référence au répertoire contenants les VIs sera inséré dans l'option *viSearchPath* du fichier *labview.ini*.

Install: Copy VI and documentation files into LabVIEW's vi.lib folder
Dans ce cas, tous les fichiers nécessaires au bon fonctionnement de la librairie sont copiés dans le répertoire d'installation de LabVIEW. Vous pourrez donc effacer les fichiers originaux une fois

l'installation terminée. Notez cependant que les exemples de programmation ne sont pas copiés. Voici ce que fait l'installeur exactement:

- Toute référence à des répertoires contenant une version quelconque de la librairie Yoctopuce sont supprimés de l'option *viSearchPath* dans le fichier *labview.ini*.
- Tous les VIs, DLL et fichiers d'aide sont copiés dans:
  C:\Program Files xx\National Instruments\LabVIEW 20xx\vi.lib\Yoctopuce
- Les VIs sont modifiés pour que leur aide pointe sur les nouveaux fichiers d'aide.
- un fichier palette *dir.mnu* référençant les VIs copiés sera créé dans le répertoire:
  C:\Program Files xx\National Instruments\LabVIEW 20xx\vi.lib\addons\Yoctopuce

Uninstall Yoctopuce Library
Cette option supprime la Librairie Yoctopuce de votre installation LabVIEW:

- Toute référence à des répertoires contenant une version quelconque de la librairie Yoctopuce sont supprimés de l'option *viSearchPath* dans le fichier *labview.ini*.
- Les répertoires suivants seront supprimé s'ils existent:
  C:\Program Files xx\National Instruments\LabVIEW 20xx\vi.lib\addons\Yoctopuce
  C:\Program Files xx\National Instruments\LabVIEW 20xx\vi.lib\Yoctopuce

Dans tous les cas, si le fichier *labview.ini* a besoin d'être modifié, une copie de backup est automatiquement réalisée avant.

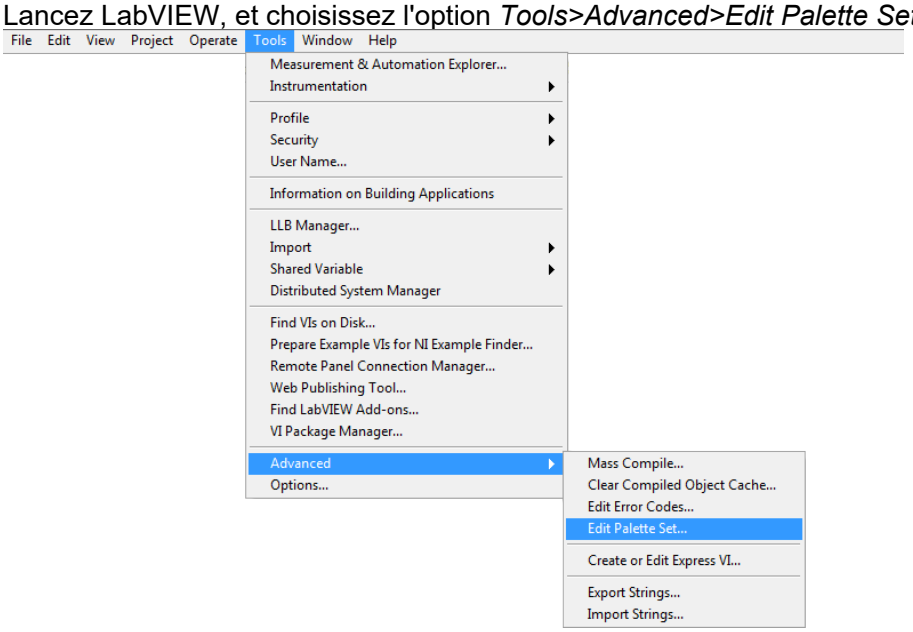
L'installeur reconnait les répertoires contenant la librairie Yoctopuce en testant l'existence du fichier *YRegisterHub.vi*.

Une fois l'installation terminée, vous trouverez une palette Yoctopuce dans le menu *Fonction/ Suppléments*.

Méthode 3 Installation manuelle dans la palette LabVIEW

Les étapes pour installer manuellement les VIs directement dans la Palette LabView sont un peu plus complexes, vous trouverez la procédure complète sur le site de National Instruments[3], mais voici un résumé:

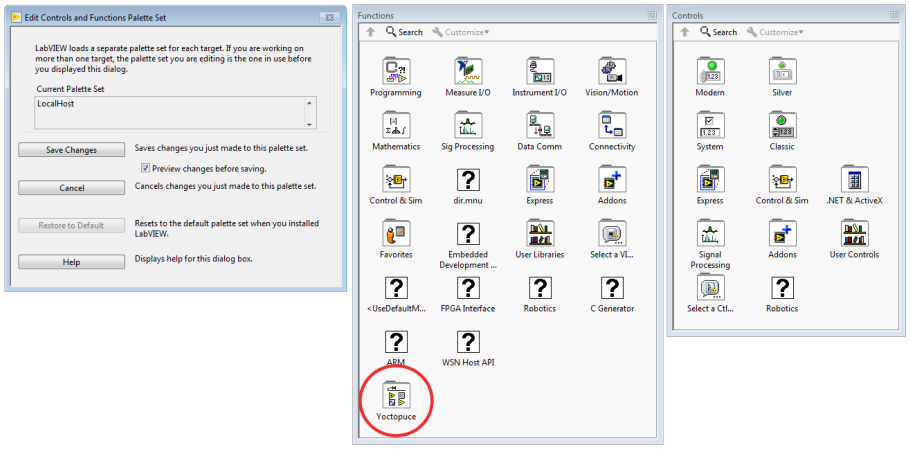
1. Créez un répertoire *Yoctopuce\API* dans le répertoire *C:\Program Files\National Instruments \LabVIEW xxxx\vi.lib*, et copiez tous les VIs et les DLL du répertoire *VIs* dedans.
2. Créez un répertoire *Yoctopuce* dans le répertoire *C:\Program Files\National Instruments \LabVIEW xxxx\menus\Categories*
3. Lancez LabVIEW, et choisissez l'option *Tools>Advanced>Edit Palette Set*



[3] https://forums.ni.com/t5/Developer-Center-Resources/Creating-a-LabVIEW-Palette/ta-p/3520557

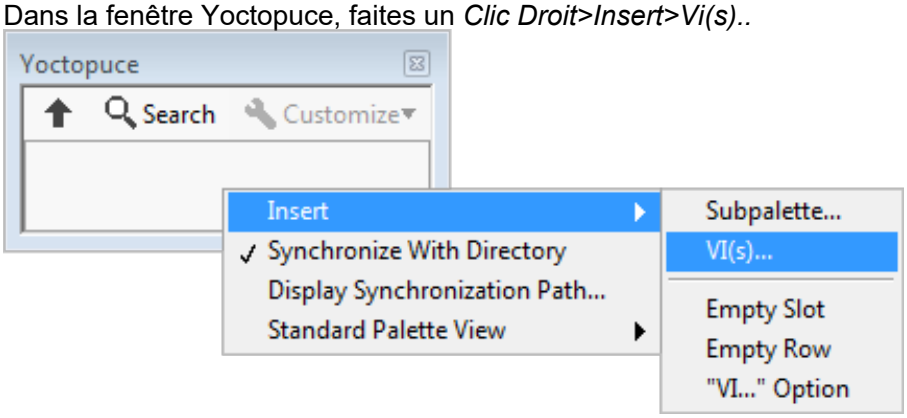
Trois fenêtres vont apparaître:

- "Edit Controls and Functions Palette Set"
- "Functions"
- "Controls"

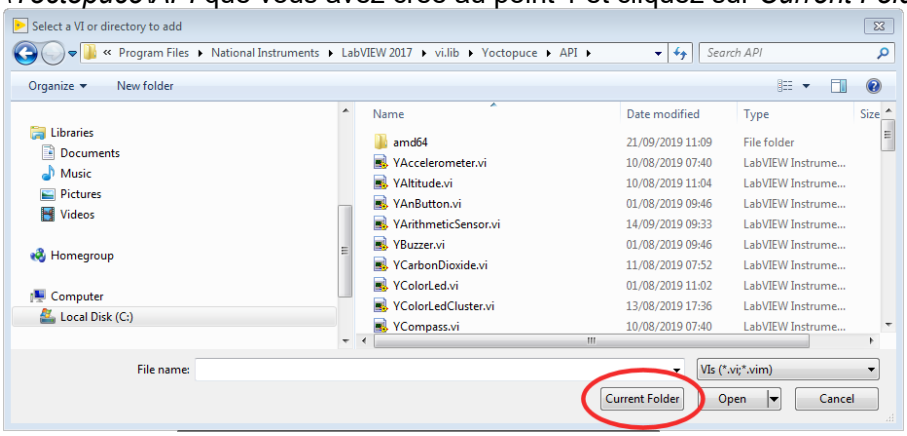


.
Dans la fenêtre *Function*, vous trouverez une icône *Yoctopuce*. Double-cliquez dessus, ce qui fera apparaitre une fenêtre "Yoctopuce" vide.

4. Dans la fenêtre Yoctopuce, faites un *Clic Droit>Insert>Vi(s)..*

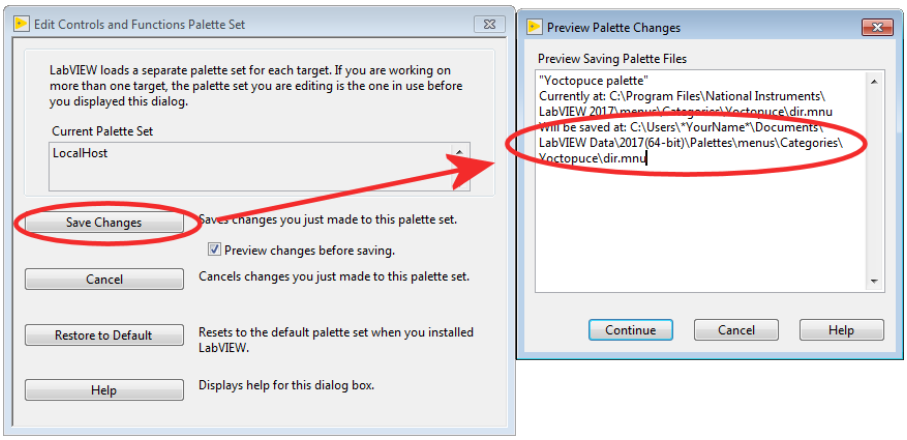


ce qui fera apparaître un sélecteur de fichier. Placer le sélecteur dans le répertoire *vi.lib \Yoctopuce\API* que vous avez créé au point 1 et cliquez sur *Current Folder*



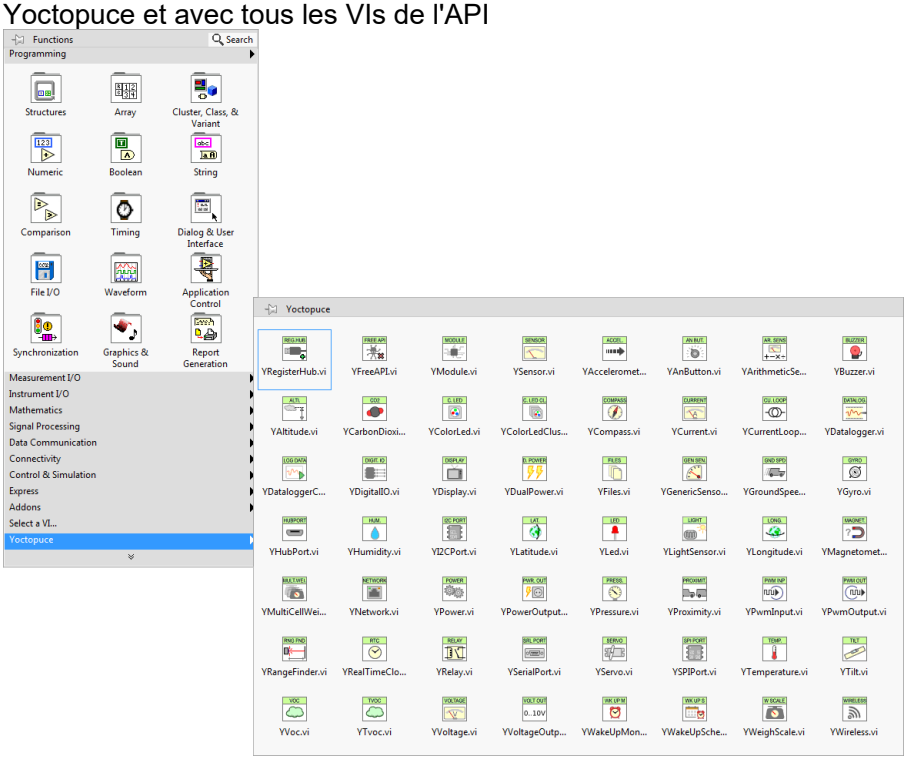
Tous les VIs Yoctopuce vont apparaitre dans la fenêtre Yoctopuce. Par défaut, ils sont triés dans l'ordre alphabétique, mais vous pouvez les arranger comme bon vous semble en les glissant avec la souris. Pour que la palette soit bien utilisable, nous vous suggérons de réorganiser les icônes sur 8 colonnes.

5. Dans la fenêtre "*Edit Controls and Functions Palette Set*", cliquez sur le bouton "Save Changes", la fenêtre va vous indiquer qu'elle a créé un fichier `dir.mnu` dans votre répertoire `Documents`.



Copiez ce fichier dans le répertoire "`menus\Categories\Yoctopuce`" que vous avez créé au point 2.

6. Redémarrez LabVIEW, la palette de LabVIEW contient maintenant une sous-palette Yoctopuce et avec tous les VIs de l'API

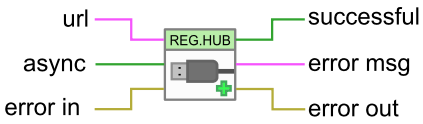


11.4. Présentation des VIs Yoctopuce

La librairie Yoctopuce pour LabVIEW comprend un VI par classe de l'API Yoctopuce, plus quelques VI spéciaux. Tous les VIs disposent des connecteurs traditionnels *Error IN* et *Error Out*.

YRegisterHub

Le VI `YRegisterHub` permet d'initialiser l'API. Ce VI doit impérativement être appelé une fois avant de faire quoi que ce soit qui soit en relation avec des modules Yoctopuce



Le VI YRegisterHub

Le VI `YRegisterHub` prend un paramètre *url* qui peut être soit:

- La chaîne de caractères `"usb"` pour indiquer que l'on souhaite travailler avec des modules locaux directement par USB
- Une addresse IP pour indiquer que l'on souhaite travailler avec des modules accessibles via une connexion réseau. Cette adresse IP peut être celle d'un YoctoHub[4] ou encore celle d'une machine sur laquelle tourne l'application VirtualHub[5].

Dans le cas d'une adresse IP, le VI `YRegisterHub` va essayer de contacter cette adresse et génèrera une erreur s'il n'y arrive pas, à moins que le paramètre *async* ne soit mis à TRUE. Si *async* est mis à TRUE, aucune erreur ne sera générée, et les modules Yoctopuce correspondant à cette adresse IP seront automatiquement mis à disposition dès que la machine concernée sera joignable.

Si tout s'est bien passé, la sortie *successful* contiendra la valeur TRUE. Dans le cas contraire elle contiendra la valeur FALSE et la sortie *error msg* contiendra une chaîne de caractères contenant une description de l'erreur

Vous pouvez utiliser plusieurs VI `YRegisterHub` avec des urls différentes si vous le souhaitez. En revanche, sur la même machine, il ne peut y avoir qu'un seul processus qui accède aux modules Yoctopuce locaux directement par USB (*url* mis à `"usb"`). Cette limitation peut facilement être contournée en faisant tourner le logiciel *VirtualHub* sur la machine locale et en utilisant l'url `"127.0.0.1"`.

YFreeAPI

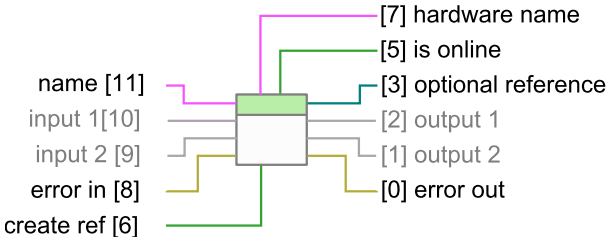
Le VI `YFreeAPI` permet de libérer les ressources allouée par l'API Yoctopuce.

Le VI YFreeAPI

Le VI `YFreeAPI` doit être appelé une fois que votre code en a fini avec l'API Yoctopuce, faute de quoi l'accès direct par USB (*url* mis à `"usb"`) pourrait rester bloqué une fois l'exécution de votre VI terminé, et ce tant que LabVIEW n'aura pas été complètement fermé.

Structure des VI correspondant à une classe

Les autres VIs correspondent à une fonction/classe de l'API Yoctopuce, ils ont tous la même structure:



Structure de la plupart des VIs de l'API.

- Connecteur [11]: *name* doit contenir le nom hardware ou le nom logique de la fonction visée.
- Connecteur [10] et [9]: paramètres d'entrée qui dépendent de la nature du VI
- Connecteur [8] et [0] : *error in* et *error out*.
- Connecteur [7] : Nom hardware unique de la fonction trouvée.
- Connecteur [5] : *is online* contient TRUE si la fonction est accessible, FALSE sinon.
- Connecteur [2] et [1]: valeurs de sortie qui dépendent de la nature du VI.
- Connecteur [6]: Si cette entré est mise à TRUE, le connecteur [3] contiendra une référence à l'objet *Proxy* implémenté par le VI[6]. Cette entrée est initialisée à FALSE par défaut.

[4] www.yoctopuce.com/FR/products/category/extensions-and-networking
[5] http://www.yoctopuce.com/EN/virtualhub.php
[6] voir section *Utilisation objets Proxy*

- Connecteur [3]: Référence sur l'objet *Proxy* implémenté par le VI si l'entrée [6] contient TRUE. Cet objet permet d'accéder à des fonctionnalités supplémentaires.

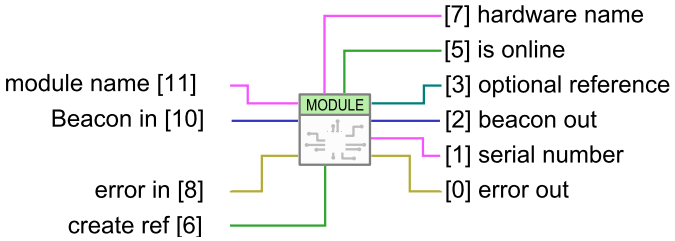
Vous trouverez la liste des fonctions disponibles sur votre Yocto-Motor-DC au chapitre *Programmation, concepts généraux*.

Si la fonction recherchée (paramètre *name*) n'est pas accessible, cela ne génèrera pas d'erreur mais la sortie *is online* contiendra FALSE et toutes les sorties contiendront les valeurs "N/A" quand c'est possible. Si la fonction recherchée devient disponible plus tard dans la vie de votre programme, *is online* passera à TRUE.

Si le paramètre *name* contient une chaîne vide, le VI ciblera la première fonction disponible du même type qu'il trouvera. Si aucune fonction n'est disponible, *is online* contiendra FALSE.

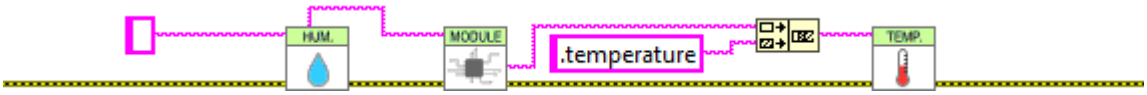
Le VI YModule

Le module `YModule` permet d'interfacer la partie "module" de chaque module Yoctopuce. Il permet de piloter la balise du module et de connaître le numéro de série d'un module.



Le VI YModule

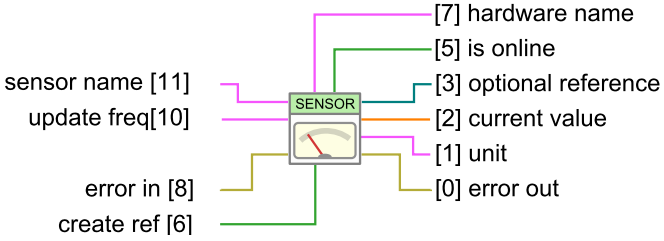
L'entrée *name* fonctionne de manière légèrement différente des autres VIs. S'il est appelé avec le paramètre *name* correspondant à un nom de fonction, le VI `YModule` trouvera la fonction *Module* du module hébergeant la fonction. Il est donc possible de trouver facilement le numéro de série du module d'une fonction quelconque. Cela permet de construire le nom d'autres fonctions qui se trouveraient sur le même module. L'exemple ci dessous trouve la première fonction *YHumidity* disponible et construit le nom de la fonction *YTemperature* qui se trouve sur le même module. Les exemples fournis avec l'API Yoctopuce font un usage extensif de cette technique.



Utilisation du VI YModule pour retrouver les fonctions hébergés sur le même module

Les VI senseurs

Tous les VI correspondant à des senseurs Yoctopuce ont exactement la même géométrie. Les deux sorties permettent de récupérer la valeur mesurée par le capteur correspondant ainsi que l'unité utilisée.



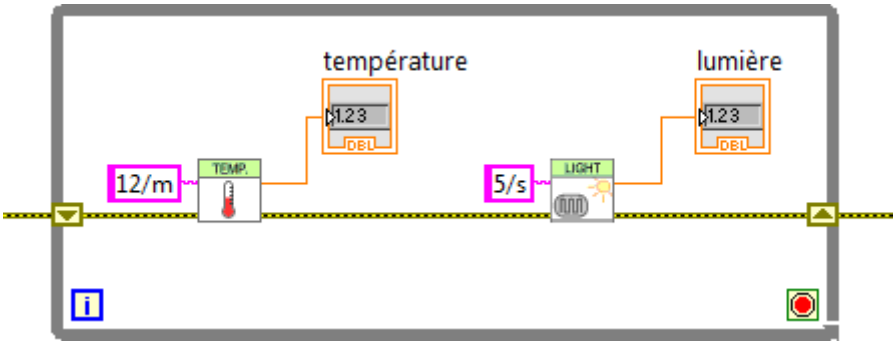
Les VI senseurs ont tous exactement la même géométrie

Le paramètre d'entrée *update freq* est une chaîne de caractères qui permet de configurer la façon dont la valeur de sortie est mis à jour:

- "auto" : la valeur du VI est mise à jour dès que le capteur détecte un changement significatif de valeur. C'est le fonctionnement par défaut.

- "x/s": la valeur du VI est mise à jour *x* fois par seconde avec la valeur instantanée du capteur.
- "x/m","x/h": la valeur du VI est mise à jour *x* fois par minute, (resp. heure) avec la valeur moyenne sur la dernière période. Attention les fréquences maximum sont (60/m) et (3600/h), pour des fréquence plus élevés utiliser la syntaxe (x/s).

La fréquence de mise à jour du VI est un paramètre géré par le module Yoctopuce physique. Si plusieurs VI essayent de changer la fréquence d'un même capteur, la configuration retenue sera celle du dernier appel. Par contre, il est tout à fait possible de configurer des fréquences différentes pour des capteurs du même module Yoctopuce.

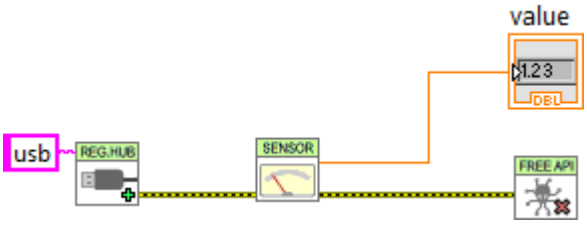


Changement de la fréquence de mise à jour du même module

La fréquence de mise à jour du VI est complètement indépendante de la fréquence d'échantillonnage du capteur qui n'est généralement pas modifiable. Il est inutile et contre-productif de définir une fréquence de mise à jour supérieure à la fréquence d'échantillonnage du capteur.

11.5. Fonctionnement et utilisation des VIs

Voici un exemple parmi les plus simples de VI utilisant l'API Yoctopuce.

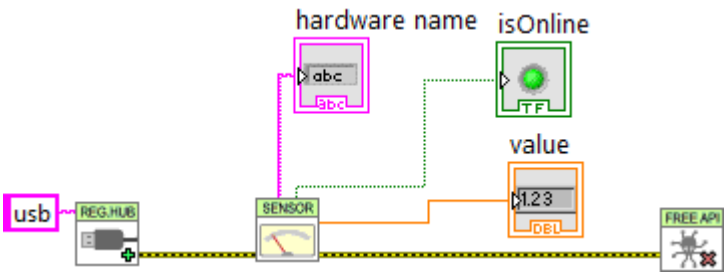


Exemple minimal d'utilisation de l'API Yoctopuce pour LabVIEW

Cet exemple s'appuie sur le VI `YSensor` qui est un VI générique qui permet d'interfacer n'importe quelle fonction senseur d'un module Yoctopuce. Vous pouvez remplacer ce VI par n'importe quel autre de l'API Yoctopuce, ils ont tous la même géométrie et fonctionnent tous de la même manière. Cet exemple se contente de faire trois choses:

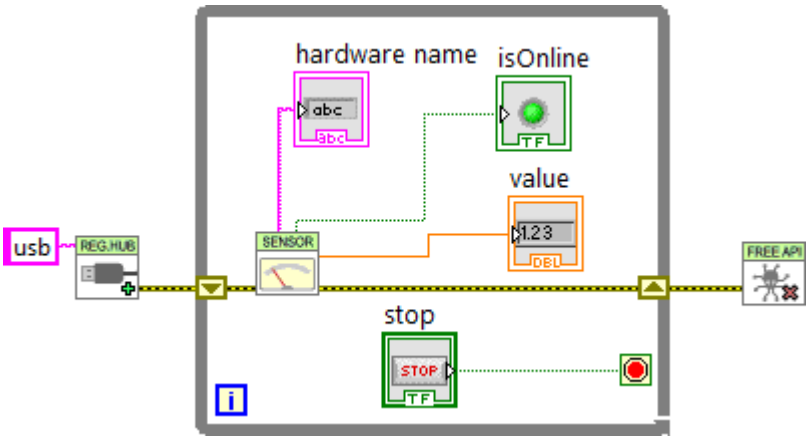
1. Il initialise l'API en mode natif ("usb") avec le VI `YRegisterHub`
2. Il affiche la valeur du premier capteur Yoctopuce qu'il trouve à l'aide du VI `YSensor`
3. Il libère l'API grâce au VI `YFreeAPI`

Cet exemple cherche automatiquement un senseur disponible, si un tel senseur est trouvé on pourra connaitre son nom via la sortie *hardware name* et la sortie *isOnline* sera à TRUE. Si aucun senseur n'est disponible, le VI ne génèrera pas d'erreur mais émulera un senseur fantôme qui sera "offline". Par contre si plus tard, dans la vie de l'application, un senseur devient disponible parce qu'il à été branché, *isOnline* passera à TRUE et le *hardware name* contiendra le nom du capteur. On peut donc facilement ajouter quelques indicateurs à l'exemple précédent pour savoir comment se passe l'exécution.



Utilisation des sorties hardware name et isOnline

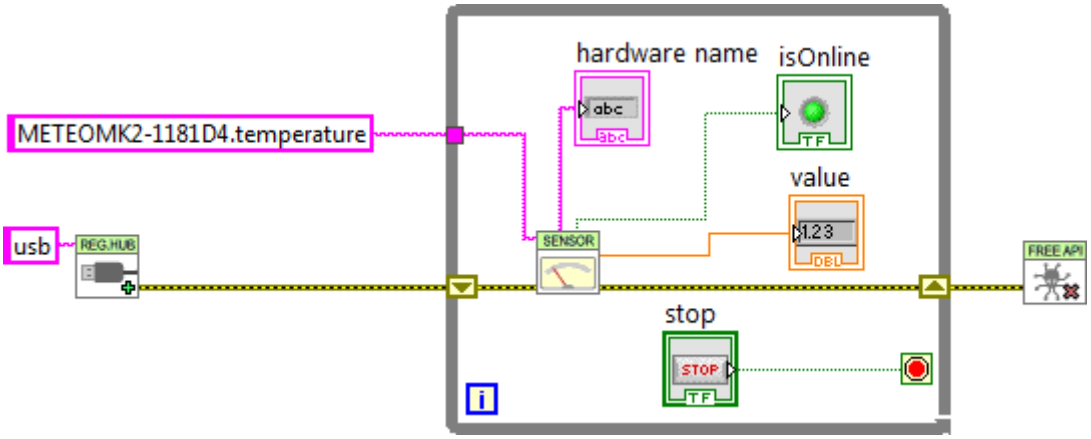
Les VIs de l'API Yoctopuce ne sont qu'une porte d'entrée sur la mécanique interne de la librairie Yoctopuce. Cette mécanique fonctionne indépendamment des VIs Yoctopuce. En effet, la plupart des communications avec les modules électroniques sont gérées automatiquement en arrière plan. C'est pourquoi vous n'avez pas forcément besoin de prendre de précaution particulière pour utiliser les VI Yoctopuce, vous pouvez par exemple les utiliser dans une boucle non temporisée sans que cela pose de problème particulier à l'API.



Les VIs Yoctopuce peuvent être utilisés dans une boucle non temporisée

Notez que le VI `YRegisterHub` n'est pas dans la boucle. Le VI `YRegisterHub` sert à l'initialiser l'API, donc à moins que vous n'ayez plusieurs url à enregistrer, il n'est pas souhaitable de l'appeler plusieurs fois.

Lorsque que le paramètre *name* est initialisé à une chaîne vide, les VI Yoctopuce recherchent automatiquement la fonction avec laquelle ils peuvent travailler, ce qui est très pratique lorsqu'on sait qu'il n'y a qu'une seule fonction du même type disponible que qu'on ne souhaite pas se soucier de gérer som nom. Si le paramètre *name* contient un nom matériel ou un nom logique, le VI cherchera la fonction correspondante, si il ne la trouve pas il émulera une fonction qui sera *offline* en attendant que la vraie fonction devienne disponible.

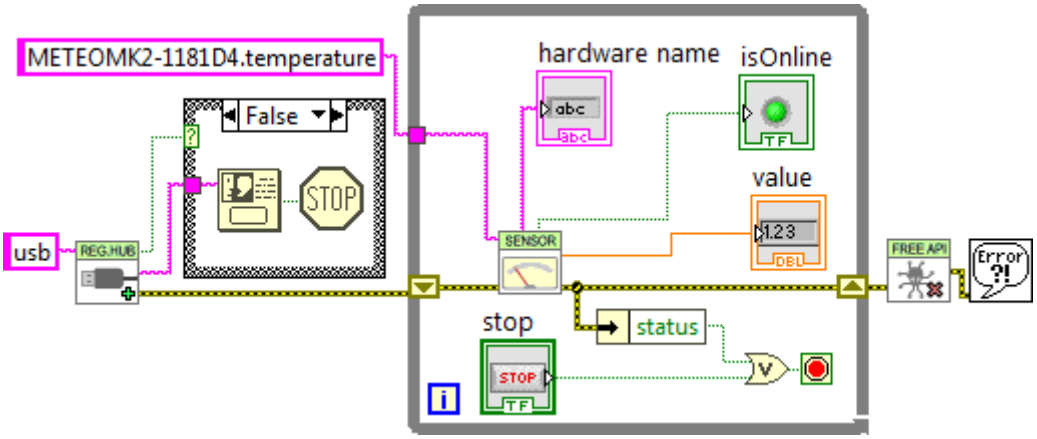


Utilisation de noms pour identifier les fonctions à utiliser

Gestion des erreurs

L'API Yoctopuce pour LabVIEW est codée pour gérer les erreurs d'une manière aussi gracieuse que possible: par exemple si vous utilisez un VI pour accéder à une fonction qui n'existe pas, sa sortie *isOnline* sera à FALSE, les autres sorties seront affecté à *NaN* et les entrées n'auront pas d'effet. Les erreurs fatales sont propagée à travers le canal traditionnel *error in*, *error out*.

Cependant, le VI `YRegisterHub` gère les erreurs de connexion de manière un peu différente. Afin de les rendre plus faciles à gérer, les erreurs de connexions sont signalées à l'aide de sorties *Success* et *error msg*. Si un problème apparait lors de l'appel au VI `YRegisterHub`, *success* contiendra FALSE et *error msg* contiendra une description de l'erreur.



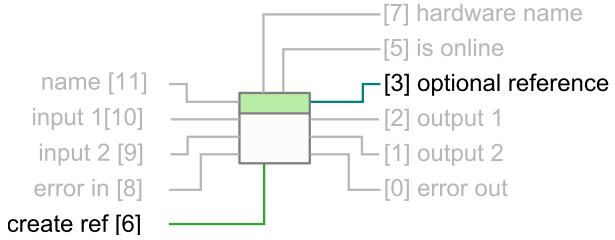
Gestion des erreurs

Le message d'erreur le plus courant est "*Another process is already using yAPI*". Il signifie qu'une autre application, LabVIEW ou autre, utilise déjà l'API en module USB natif. En effet, pour des raison techniques, l'API USB native ne peut être utilisée que par une seule application à la fois sur la même machine. Cette limitation peut être facilement contourné en utilisant le mode réseau.

11.6. Utilisation des objets *Proxy*

L'API Yoctopuce contient des centaines de méthodes, fonctions et propriétés. Il n'était ni possible, ni souhaitable de créer un VI pour chacune d'entre elles. C'est pourquoi il y a un VI par classe qui expose les deux propriétés que Yoctopuce a jugé les plus utiles, mais cela ne veut pas dire que les autres ne sont pas accessibles.

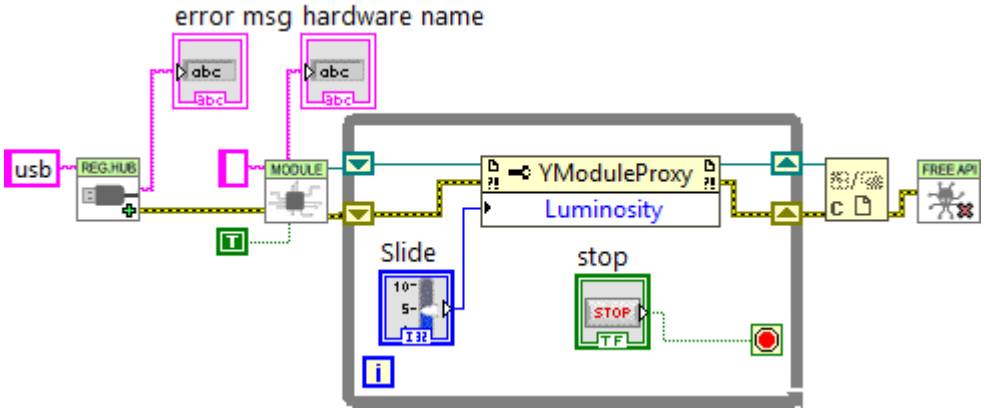
Chaque VI correspondant à une classe dispose de deux connecteurs *create ref* et *optional ref* qui permettent d'obtenir une référence sur l'objet *Proxy* de l'API *.NET Proxy* sur laquelle est construite la librairie LabVIEW.



Les connecteurs pour obtenir une référence sur l'objet Proxy correspondant au VI

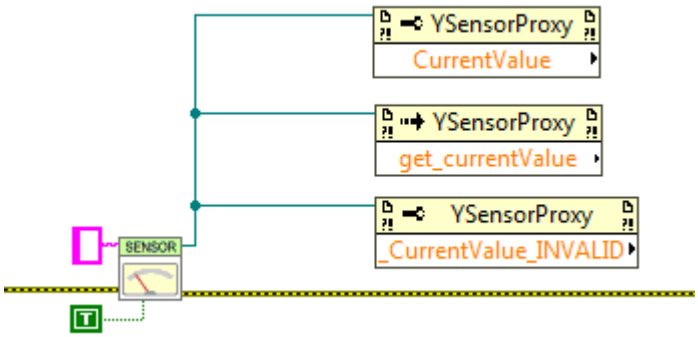
Pour obtenir cette référence, il suffit de mettre *optional ref* à TRUE. Attention, il est impératif de fermer toute référence créée de cette manière, sous peine de saturer rapidement la mémoire de l'ordinateur.

Voici un exemple qui utilise cette technique pour modifier la luminosité des LEDs d'un module Yoctopuce



Contrôle de la luminosité des LEDs d'un module

Notez que chaque référence permet d'obtenir aussi bien des propriétés (noeud *property*) que des méthodes (noeud *invoke*). Par convention, les propriétés sont optimisées pour générer un minimum de communication avec les modules, c'est pourquoi il est recommandé de les utiliser plutôt les méthodes *get_xxx* et *set_xxx* correspondantes qui pourraient sembler équivalentes mais qui ne sont pas optimisées. Les propriétés permettent aussi récupérer les différentes constantes de l'API, qui sont préfixées avec le caractère "_". Pour des raisons techniques, les méthodes *get_xxx* et *set_xxx* ne sont pas toutes disponibles sous forme de propriétés.

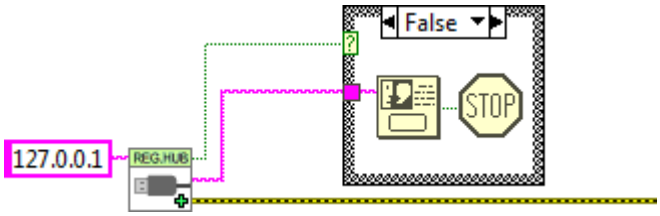


Noeuds Property et Invoke: Utilisation de propriétés, méthodes et constantes

Vous trouverez la description de toutes les propriétés, fonctions et méthodes disponibles dans la documentation de l'API *.NET Proxy*.

Utilisation en réseau

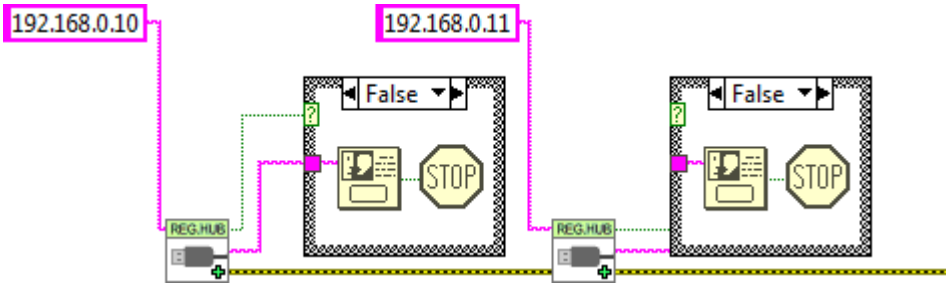
Sur une même machine, il ne peut y avoir qu'un seul processus qui accède aux modules Yoctopuce locaux directement par USB (url mis à `"usb"`). Par contre, plusieurs processus peuvent se connecter en parallèle à des YoctoHubs[7] ou à une machine sur laquelle tourne le logiciel *VirtualHub*[8], y compris la machine locale. Si vous utilisez l'adresse réseau locale de votre machine (127.0.0.1) et qu'un VirtualHub tourne dessus, vous pourrez ainsi contourner la limitation qui empêche l'utilisation en parallèle de l'API native USB.



Utilisation en mode réseau

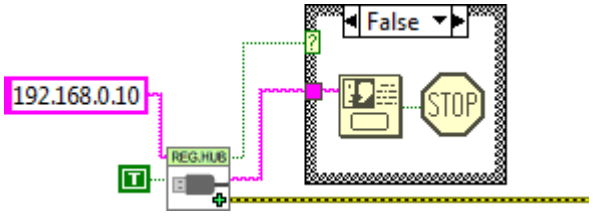
[7] www.yoctopuce.com/FR/products/category/extensions-et-reseau
[8] www.yoctopuce.com/FR/virtualhub.php

Il n'y a pas non plus de limitation sur le nombre d'interfaces réseau auxquels l'API peut se connecter en parallèle. Autrement dit, il est tout à fait possible de faire des appels multiples au VI `YRegisterHub`. C'est le seul cas où il y a un intérêt à appeler le VI `YRegisterHub` plusieurs fois au cours de la vie de l'application.



Les connexions réseau multiples sont possibles

Par défaut, le VI `YRegisterHub` essaye se connecter sur l'adresse donnée en paramètre et génère une erreur (*success*=FALSE) s'il n'y arrive pas parce que personne ne répond. Mais si le paramètre *async* est initialisé à TRUE, aucune erreur ne sera générée en cas d'erreur de connexion, mais si la connexion devient possible plus tard dans la vie de l'application, les modules correspondants seront automatiquement accessibles.



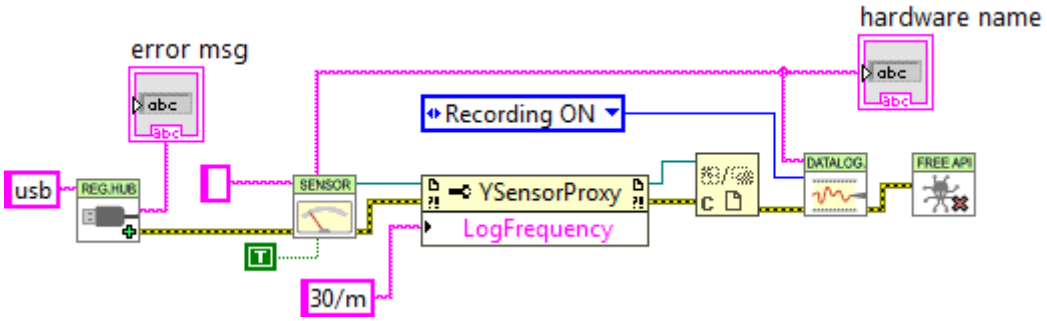
Connexion asynchrone

11.7. Gestion du datalogger

Quasiment tous les senseurs Yoctopuce disposent d'un enregistreur de données qui permet de stocker les mesures des senseurs dans la mémoire non volatile du module. La configuration de l'enregistreur de données peut être réalisée avec le VirtualHub, mais aussi à l'aide d'un peu de code LabVIEW

Enregistrement

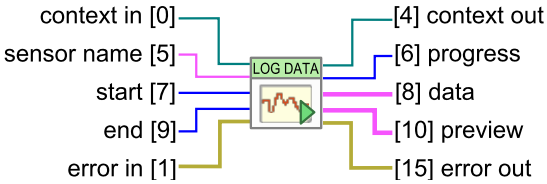
Pour ce faire, il faut configurer la fréquence d'enregistrement en utilisant la propriété "LogFrequency" que l'on atteint avec une référence sur l'objet *Proxy* du senseur utilisé, puis il faut mettre en marche l'enregistreur grâce au VI `YDataLogger`. Noter qu'à la manière du VI `YModule`, le VI `YDataLogger` correspondant à un module peut être obtenu avec son propre nom, mais aussi avec le nom de n'importe laquelle des fonctions présentes sur le même module.



Enclenchement de l'enregistrement de données dans le datalogger

Lecture

La récupération des données de l'enregistreur se fait l'aide du VI `YDataLoggerContents`.

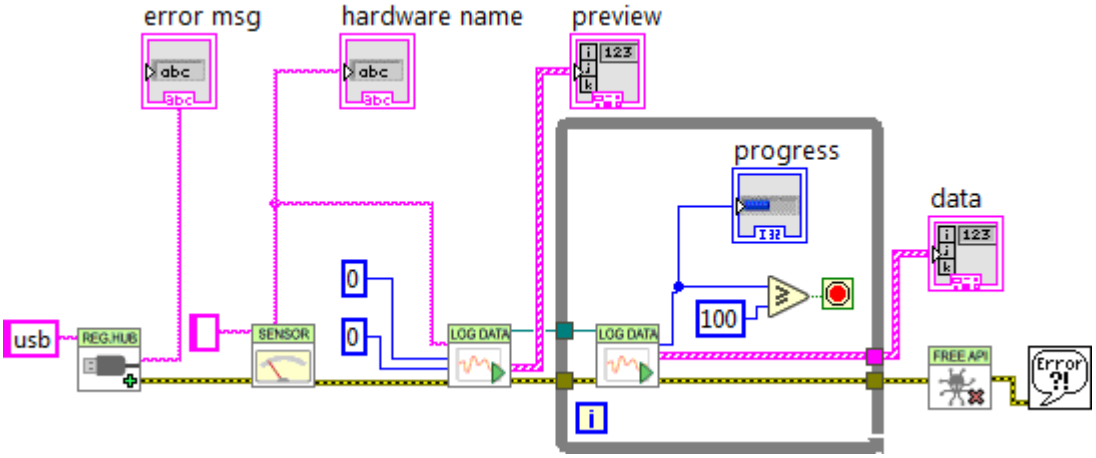


Le VI YDataLoggerContents

Extraire les données de l'enregistreur d'un module Yoctopuce est un processus lent qui peut prendre plusieurs dizaines de secondes. C'est pourquoi le VI qui permet cette opération a été conçu pour fonctionner de manière itérative.

Dans un premier temps le VI doit être appelé avec un nom de senseur, une date de début et une date de fin (timestamp UNIX en UTC). Le couple (0,0) permet d'obtenir la totalité du contenu de l'enregistreur. Ce premier appel permet d'obtenir un résumé du contenu du datalogger et un contexte.

Dans un deuxième temps, il faut rappeler le VI `YDataLoggerContents` en boucle avec le paramètre contexte, jusqu'à ce que la sortie *progress* atteigne la valeur 100. A ce moment la sortie data représente le contenu de l'enregistreur



Récupération du contenu de l'engistreur de données

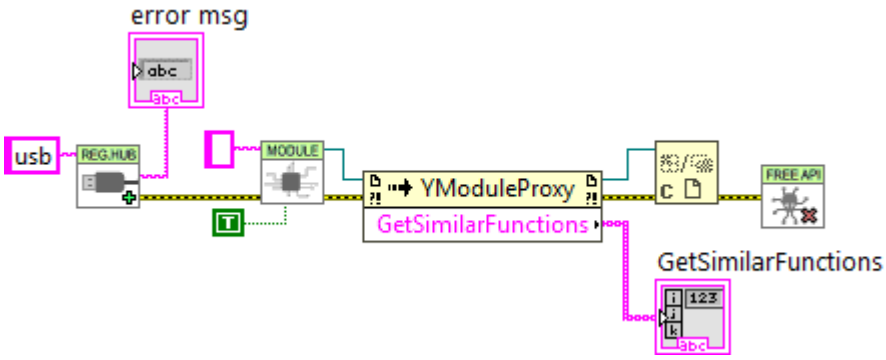
Les résultats et le résumé sont rendus sous la forme d'un tableau de structures qui contiennent les champs suivants:

- *startTime*: début de la période de mesure
- *endTime*: fin de la période de mesure
- *averageValue*: valeur moyenne pour la période
- *minValue*: valeur minimum sur la période
- *maxValue*: valeur maximum sur la période

Notez que si la fréquence d'enregistrement est supérieure à 1 Hz, l'enregistreur ne mémorise que des valeurs instantanées, dans ce cas *averageValue*, *minValue*, et *maxValue* auront la même valeur.

11.8. Énumération de fonctions

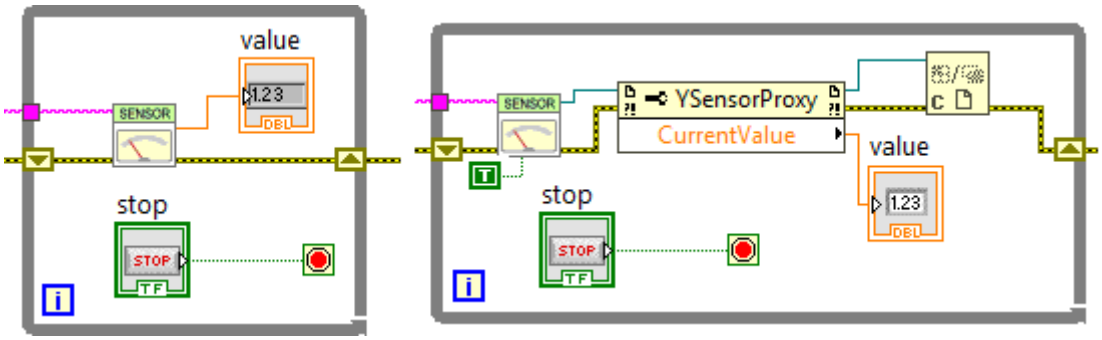
Chaque VI correspondant à un objet de l'API *.NET Proxy* permet de faire une énumération de toutes les fonctions de la même classe via la méthode *getSimilarfunctions()* de l'objet *Proxy* correspondant. Ainsi il est ainsi aisé de faire un inventaire de tous les modules connectés, de tous les capteurs connectés, de tous les relais connectés, etc....



Récupération de la liste de tous les modules connectés

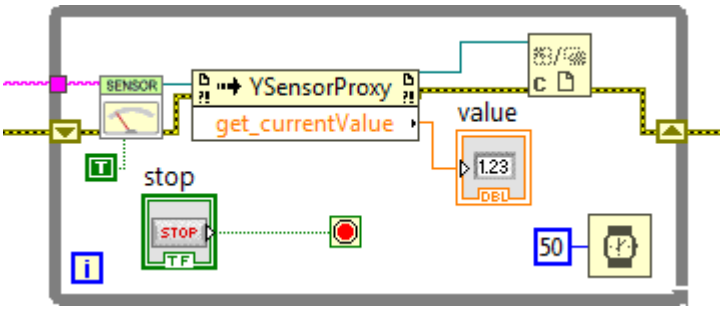
11.9. Un mot sur les performances

L'API Yoctopuce pour LabVIEW été optimisée de manière à ce que les tous les VIs et les propriétés de objets *Proxy* génèrent un minimum de communication avec les modules Yoctopuce. Ainsi vous pouvez les utiliser dans des boucles sans prendre de précaution particulière: vous n'êtes pas *obligés* de ralentir les boucles avec un timer.



Ces deux boucles génèrent peu de communications USB et n'ont pas besoin d'être ralenties

En revanche, presque toutes les méthodes des objets Proxy disponibles vont générer une communication avec les modules Yoctopuce à chaque fois qu'elles seront appelées, il conviendra donc d'éviter de les appeler trop souvent inutilement.



Cette boucle, qui utilise une méthode, doit être ralentie

11.10. Un exemple complet de programme LabVIEW

UNABLE TO INCLUDE
http://172.17.17.77/tu/projects/yoctomotorcontroler/public/doc-labview-example-FR.html

Si vous lisez cette documentation sur un écran, vous pouvez zoomer sur l'image ci-dessus. Vous pourrez aussi retrouver cet exemple dans la librairie Yoctopuce pour LabVIEW

11.11. Différences avec les autres API Yoctopuce

Yoctopuce fait tout son possible pour maintenir une forte cohérence entre les différentes librairies de programmation. Cependant, LabVIEW étant un environnement clairement à part, il en résulte des différences importantes avec les autres librairies.

Ces différences ont aussi été introduites pour rendre l'utilisation des modules aussi facile et intuitive que possible en nécessitant un minimum de code LabVIEW.

YFreeAPI

Contrairement aux autres langages, il est indispensable de libérer l'API native en appelant le VI `YFreeApi` lorsque votre code n'a plus besoin d'utiliser l'API. Si cet appel est omis, l'API native risque de rester bloquée pour les autres applications tant que LabVIEW ne sera pas complètement fermé.

Propriétés

Contrairement aux classes des autres API, les classes disponibles dans LabVIEW implémentent des *propriétés*. Par convention, ces propriétés sont optimisées pour générer un minimum de communication avec les modules tout en se rafraichissant automatiquement. En revanche, les méthodes de type *get_xxx* et *set_xxx* génèrent systématiquement des communications avec les modules Yoctopuce et doivent être appelées à bon escient.

Callback vs Propriétés

Il n'y a pas de callbacks dans l'API Yoctopuce pour LabVIEW, les VIs se rafraichissenti automatiquement: ils sont basés sur les propriétés des objets de l'API *.NET Proxy*.

12. Utilisation du Yocto-Motor-DC en Java

Java est un langage orienté objet développé par Sun Microsystem. Son principal avantage est la portabilité, mais cette portabilité a un coût. Java fait une telle abstraction des couches matérielles qu'il est très difficile d'interagir directement avec elles. C'est pourquoi l'API java standard de Yoctopuce ne fonctionne pas en natif: elle doit passer par l'intermédiaire d'un VirtualHub pour pouvoir communiquer avec les modules Yoctopuce.

12.1. Préparation

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

- La librairie de programmation pour Java[1]
- Le programme VirtualHub[2] pour Windows, Mac OS X ou Linux selon l'OS que vous utilisez

La librairie est disponible en fichier sources, mais elle aussi disponible sous la forme d'un fichier jar. Branchez vos modules, Décompressez les fichiers de la librairie dans un répertoire de votre choix. Lancez le programme VirtualHub, et vous pouvez commencer vos premiers test. Vous n'avez pas besoin d'installer de driver.

Afin de les garder simples, tous les exemples fournis dans cette documentation sont des applications consoles. Il va de soit que que le fonctionnement des librairies est strictement identiques si vous les intégrez dans une application dotée d'une interface graphique.

12.2. Contrôle de la fonction Motor

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

```
[...]
// On active l'accès aux modules locaux à travers le VirtualHub
YAPI.RegisterHub("127.0.0.1");
[...]

// On récupère l'objet permettant d'intéragir avec le module
motor = YMotor.FindMotor("MOTORCTL-123456.motor");

// Pour gérer le hot-plug, on vérifie que le module est là
if (motor.isOnline())
```

[1] www.yoctopuce.com/FR/libraries.php
[2] www.yoctopuce.com/FR/virtualhub.php

```
{
    // Utiliser motor.set_drivingForce()
    [...]
}

[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

YAPI.RegisterHub

La fonction `YAPI.RegisterHub` initialise l'API de Yoctopuce en indiquant où les modules doivent être recherchés. Le paramètre est l'adresse du virtual hub capable de voir les modules. Si l'initialisation se passe mal, une exception sera générée.

YMotor.FindMotor

La fonction `YMotor.FindMotor` permet de retrouver un moteur 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-Motor-DC avec le numéros de série *MOTORCTL-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *motor* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
motor = YMotor.FindMotor("MOTORCTL-123456.motor")
motor = YMotor.FindMotor("MOTORCTL-123456.MaFonction")
motor = YMotor.FindMotor("MonModule.motor")
motor = YMotor.FindMotor("MonModule.MaFonction")
motor = YMotor.FindMotor("MaFonction")
```

`YMotor.FindMotor` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le moteur.

isOnline

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

set_drivingForce

La méthode `set_drivingForce()` de l'objet renvoyé par `YMotor.FindMotor` permet de régler la puissance, de -100 à +100% envoyée au moteur.

Un exemple réel

Lancez votre environnement java et ouvrez le projet correspondant, fourni dans le répertoire **Examples/Doc-GettingStarted-Yocto-Motor-DC** de la librairie Yoctopuce.

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

```
import com.yoctopuce.YoctoAPI.*;

public class Demo {

    public static void main(String[] args)    {
        try {
            // setup the API to use local VirtualHub
            YAPI.RegisterHub("127.0.0.1");
        } catch (YAPI_Exception ex) {
            System.out.println("Cannot contact VirtualHub on 127.0.0.1 (" +
ex.getLocalizedMessage() + ")");
            System.out.println("Ensure that the VirtualHub application is running");
            System.exit(1);
        }

        String serial = "";
        if (args.length > 0 && !args[0].equalsIgnoreCase("any")) {
            serial = args[0];
```

```
        } else {
            YMotor tmp = YMotor.FirstMotor();
            if (tmp == null) {
                System.out.println("No module connected (check USB cable)");
                System.exit(1);
            }
            try {
                serial = tmp.module().get_serialNumber();
            } catch (YAPI_Exception ex) {
                System.out.println("No module connected (check USB cable)");
                System.exit(1);
            }
        }
        int power = 100;
        if (args.length>1) {
            power = Integer.valueOf(args[1]);
        }

        YMotor motor = YMotor.FindMotor(serial + ".motor");
        YCurrent current = YCurrent.FindCurrent(serial + ".current");
        YVoltage voltage = YVoltage.FindVoltage(serial + ".voltage");
        YTemperature temperature = YTemperature.FindTemperature(serial + ".temperature");

        // lets start the motor
        if (motor.isOnline()) {
            // if motor is in error state, reset it.
            try {
                if (motor.get_motorStatus() >= YMotor.MOTORSTATUS_LOVOLT)
                    motor.resetStatus();
                motor.drivingForceMove(power, 2000);  // ramp up to power in 2 seconds
                for (int i = 0 ; i < 60; i++) {
                    // display motor status
                    System.out.println("Status=" + motor.get_advertisedValue() +
                            "  Voltage=" + voltage.get_currentValue() + voltage.get_unit()
+
                            "  Current=" + current.get_currentValue()  + current.get_unit()
+
                            "  Temp=" + temperature.get_currentValue() +
temperature.get_unit());
                    YAPI.Sleep(1000); // wait for one second
                }
            } catch (YAPI_Exception e) {
                e.printStackTrace();
            }
        }

        YAPI.FreeAPI();
    }

    private static void usage()
    {
        System.err.println("Usage:");
        System.err.println("  demo <serial_number> power");
        System.err.println("  demo <logical_name> power");
        System.err.println("  demo any power");
        System.err.println("     power is a integer between -100 and 100%");
        System.err.println("Example:");
        System.err.println("  demo any 75");
        System.exit(0);
    }
}
```

12.3. Contrôle de la partie module

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

```
import com.yoctopuce.YoctoAPI.*;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```
public class Demo {

    public static void main(String[] args)
    {
        try {
            // setup the API to use local VirtualHub
            YAPI.RegisterHub("127.0.0.1");
        } catch (YAPI_Exception ex) {
            System.out.println("Cannot contact VirtualHub on 127.0.0.1 (" +
ex.getLocalizedMessage() + ")");
            System.out.println("Ensure that the VirtualHub application is running");
            System.exit(1);
        }
        System.out.println("usage: demo [serial or logical name] [ON/OFF]");

        YModule module;
        if (args.length == 0) {
            module = YModule.FirstModule();
            if (module == null) {
                System.out.println("No module connected (check USB cable)");
                System.exit(1);
            }
        } else {
            module = YModule.FindModule(args[0]);  // use serial or logical name
        }

        try {
            if (args.length > 1) {
                if (args[1].equalsIgnoreCase("ON")) {
                    module.setBeacon(YModule.BEACON_ON);
                } else {
                    module.setBeacon(YModule.BEACON_OFF);
                }
            }
            System.out.println("serial:       " + module.get_serialNumber());
            System.out.println("logical name: " + module.get_logicalName());
            System.out.println("luminosity:   " + module.get_luminosity());
            if (module.get_beacon() == YModule.BEACON_ON) {
                System.out.println("beacon:       ON");
            } else {
                System.out.println("beacon:       OFF");
            }
            System.out.println("upTime:       " + module.get_upTime() / 1000 + " sec");
            System.out.println("USB current:  " + module.get_usbCurrent() + " mA");
            System.out.println("logs:\n" + module.get_lastLogs());
        } catch (YAPI_Exception ex) {
            System.out.println(args[1] + " not connected (check identification and USB
cable)");
        }
        YAPI.FreeAPI();
    }
}
```

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

Modifications des réglages du module

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

```
import com.yoctopuce.YoctoAPI.*;

public class Demo {
```

```
    public static void main(String[] args)
    {
        try {
            // setup the API to use local VirtualHub
            YAPI.RegisterHub("127.0.0.1");
        } catch (YAPI_Exception ex) {
            System.out.println("Cannot contact VirtualHub on 127.0.0.1 (" +
ex.getLocalizedMessage() + ")");
            System.out.println("Ensure that the VirtualHub application is running");
            System.exit(1);
        }

        if (args.length != 2) {
            System.out.println("usage: demo <serial or logical name> <new logical name>");
            System.exit(1);
        }

        YModule m;
        String newname;

        m = YModule.FindModule(args[0]); // use serial or logical name

        try {
            newname = args[1];
            if (!YAPI.CheckLogicalName(newname))
                {
                    System.out.println("Invalid name (" + newname + ")");
                    System.exit(1);
                }

            m.set_logicalName(newname);
            m.saveToFlash(); // do not forget this

            System.out.println("Module: serial= " + m.get_serialNumber());
            System.out.println(" / name= " + m.get_logicalName());
        } catch (YAPI_Exception ex) {
            System.out.println("Module " + args[0] + "not connected (check identification
and USB cable)");
            System.out.println(ex.getMessage());
            System.exit(1);
        }

        YAPI.FreeAPI();
    }
}
```

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 `YModule.saveToFlash()` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

Enumeration des modules

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

```
import com.yoctopuce.YoctoAPI.*;

public class Demo {

    public static void main(String[] args)
    {
        try {
            // setup the API to use local VirtualHub
            YAPI.RegisterHub("127.0.0.1");
        } catch (YAPI_Exception ex) {
            System.out.println("Cannot contact VirtualHub on 127.0.0.1 (" +
ex.getLocalizedMessage() + ")");
            System.out.println("Ensure that the VirtualHub application is running");
            System.exit(1);
```

```
        }

        System.out.println("Device list");
        YModule module = YModule.FirstModule();
        while (module != null) {
            try {
                System.out.println(module.get_serialNumber() + " (" +
module.get_productName() + ")");
            } catch (YAPI_Exception ex) {
                break;
            }
            module = module.nextModule();
        }
        YAPI.FreeAPI();
    }
}
```

12.4. Gestion des erreurs

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

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

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

13. Utilisation du Yocto-Motor-DC avec Android

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

13.1. Accès Natif et Virtual Hub.

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

13.2. Préparation

Connectez-vous sur le site de Yoctopuce et téléchargez la librairie de programmation pour Java pour Android[1]. La librairie est disponible en fichiers sources, mais elle aussi disponible sous la forme d'un fichier jar. Branchez vos modules, décompressez les fichiers de la librairie dans le répertoire de votre choix. Et configurez votre environnement de programmation Android pour qu'il puisse les trouver.

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

13.3. Compatibilité

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

[1] www.yoctopuce.com/FR/libraries.php

Android 4.x

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

Support USB *host*

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

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

Matériel supporté

La librairie est testée et validée sur les machines suivantes:

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

Si votre machine Android n'est pas capable de faire fonctionner nativement des modules Yoctopuce, il vous reste tout de même la possibilité de contrôler à distance des modules pilotés par un VirtualHub sur un autre OS ou un YoctoHub[2].

13.4. Activer le port USB sous Android

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

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

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

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

```
</manifest>
```

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

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

Lancement automatique

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

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

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

</manifest>
```

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

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

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

13.5. Contrôle de la fonction Motor

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

```
[...]
// On active la détection des modules sur USB
YAPI.EnableUSBHost(this);
YAPI.RegisterHub("usb");
[...]
// On récupère l'objet permettant de communiquer avec le module
motor = YMotor.FindMotor("MOTORCTL-123456.motor");

// Pour gérer le hot-plug, on vérifie que le module est là
if (motor.isOnline())
{
    // Utilisez motor.set_drivingForce()
    [...]
}

[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

YAPI.EnableUSBHost

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

YAPI.RegisterHub

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

YMotor.FindMotor

La fonction `YMotor.FindMotor` permet de retrouver un moteur 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-Motor-DC avec le numéros de série *MOTORCTL-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *motor* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
motor = YMotor.FindMotor("MOTORCTL-123456.motor")
motor = YMotor.FindMotor("MOTORCTL-123456.MaFonction")
motor = YMotor.FindMotor("MonModule.motor")
motor = YMotor.FindMotor("MonModule.MaFonction")
motor = YMotor.FindMotor("MaFonction")
```

`YMotor.FindMotor` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le moteur.

isOnline

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

set_drivingForce

La méthode `set_drivingForce()` de l'objet renvoyé par `YMotor.FindMotor` permet de régler la puissance, de -100 à +100% envoyée au moteur.

Un exemple réel

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

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

```
package com.yoctopuce.doc_examples;

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

import com.yoctopuce.YoctoAPI.YAPI;
import com.yoctopuce.YoctoAPI.YAPI_Exception;
import com.yoctopuce.YoctoAPI.YCurrent;
import com.yoctopuce.YoctoAPI.YMotor;
import com.yoctopuce.YoctoAPI.YTemperature;
import com.yoctopuce.YoctoAPI.YVoltage;

public class GettingStarted_Yocto_Motor_DC extends Activity implements
OnItemSelectedListener
{

    private ArrayAdapter<String> aa;
    private Handler handler;
    private YMotor motor;
    private YCurrent current;
    private YTemperature temperature;
    private YVoltage voltage;

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

    public void updateMotor(View view)
    {
        if (motor!=null) {
            EditText control = (EditText)findViewById(R.id.controlfield);
            Integer power = Integer.valueOf(String.valueOf(control.getText()));
            try {
                // if motor is in error state, reset it.
                if (motor.get_motorStatus()>=YMotor.MOTORSTATUS_LOVOLT) {
                    motor.resetStatus();
                }
                motor.drivingForceMove(power, 2000);  // ramp up to power in 2 seconds
            } catch (YAPI_Exception e) {
                e.printStackTrace();
            }
        }
    }


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

        try {
```

```
            aa.clear();
            YAPI.EnableUSBHost(this);
            YAPI.RegisterHub("usb");
            YMotor m = YMotor.FirstMotor();
            while (m != null) {
                String serial = m.get_module().get_serialNumber();
                aa.add(serial);
                m = m.nextMotor();
            }
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
        // refresh Spinner with detected relay
        aa.notifyDataSetChanged();
        handler.postDelayed(r, 500);
    }

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

    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int pos, long id)
    {
        String serial = parent.getItemAtPosition(pos).toString();
        motor       = YMotor.FindMotor(serial + ".motor");
        current     = YCurrent.FindCurrent(serial + ".current");
        voltage     = YVoltage.FindVoltage(serial + ".voltage");
        temperature = YTemperature.FindTemperature(serial + ".temperature");

    }

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

    final Runnable r = new Runnable()
    {
        public void run()
        {
            if (motor != null && motor.isOnline()) {
                try {
                    TextView view = (TextView) findViewById(R.id.statefield);
                    view.setText(motor.get_advertisedValue());
                } catch (YAPI_Exception e) {
                    e.printStackTrace();

                }
            }
            if (current != null && current.isOnline()) {
                try {
                    TextView view = (TextView) findViewById(R.id.currentfield);
                    view.setText(String.format("%.1f %s", current.getCurrentValue(),
current.getUnit()));
                } catch (YAPI_Exception e) {
                    e.printStackTrace();

                }
            }
            if (voltage != null && voltage.isOnline()) {
                try {
                    TextView view = (TextView) findViewById(R.id.votltagefield);
                    view.setText(String.format("%.1f %s", voltage.getCurrentValue(),
voltage.getUnit()));
                } catch (YAPI_Exception e) {
                    e.printStackTrace();

                }
            }
            if (temperature != null && temperature.isOnline()) {
                try {
                    TextView view = (TextView) findViewById(R.id.tempfield);
                    view.setText(String.format("%.1f %s", temperature.getCurrentValue(),
temperature.getUnit()));
```

```
                } catch (YAPI_Exception e) {
                    e.printStackTrace();
                }
            }
            handler.postDelayed(this, 1000);
        }
    };


}
```

13.6. Contrôle de la partie module

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

```
package com.yoctopuce.doc_examples;

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

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

public class ModuleControl extends Activity implements OnItemSelectedListener
{

    private ArrayAdapter<String> aa;
    private YModule module = null;

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

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

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

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

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

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

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

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

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

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

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

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

Modifications des réglages du module

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

```
package com.yoctopuce.doc_examples;

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

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

public class SaveSettings extends Activity implements OnItemSelectedListener
{

    private ArrayAdapter<String> aa;
    private YModule module = null;

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

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

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

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

    private void DisplayModuleInfo()
```

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

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

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

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

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

}
```

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

Enumeration des modules

Obtenir la liste des modules connectés se fait à l'aide de la fonction `YModule.yFirstModule()` qui renvoie le premier module trouvé, il suffit ensuite d'appeler la mehode `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

```
package com.yoctopuce.doc_examples;

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

import com.yoctopuce.YoctoAPI.YAPI;
```

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

public class Inventory extends Activity
{

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

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

        try {
            YAPI.UpdateDeviceList();
            YModule module = YModule.FirstModule();
            while (module != null) {
                String line = module.get_serialNumber() + " (" + module.get_productName() +
")";
                TextView tx = new TextView(this);
                tx.setText(line);
                tx.setTextSize(TypedValue.COMPLEX_UNIT_SP, 20);
                layout.addView(tx);
                module = module.nextModule();
            }
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
    }

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

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

}
```

13.7. Gestion des erreurs

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

La manière la plus simple de contourner le problème est celle que nous avons employé pour les petits exemples précédents de ce chapitre: avant d'accéder à un module, on vérifie qu'il est en ligne avec la méthode `isOnline()` et on suppose ensuite qu'il va y rester pendant la fraction de seconde nécessaire à exécuter les lignes de code suivantes. Ce n'est pas parfait, mais ça peut

suffire dans certains cas. Il faut toutefois être conscient qu'on ne peut pas totalement exclure une erreur se produisant après le `isOnline()`, qui pourrait faire planter le programme.

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

14. Utilisation du Yocto-Motor-DC en TypeScript

TypeScript est une version améliorée du langage de programmation JavaScript. Il s'agit d'un sur-ensemble syntaxique avec typage fort, permettant d'améliorer la fiabilité du code, mais qui est transcompilé en JavaScript avant l'exécution, pour être ensuite interprêté par n'importe quel navigateur Web ou par Node.js.

Cette librairie de programmation Yoctopuce permet donc de coder des applications JavaScript tout en bénéficiant d'un typage fort. Comme notre librairie EcmaScript, elle utilise les fonctionnalités asynchrones introduites dans la version ECMAScript 2017 et qui sont maintenant disponibles nativement dans tous les environnements JavaScript modernes. Néanmoins, à ce jour, le code TypeScript n'est pas utilisable directement dans un navigateur Web ou Node.js, donc il est nécessaire de le compiler en JavaScript avant l'exécution.

La librairie peut travailler aussi bien dans un navigateur internet que dans un environnement Node.js. Pour satisfaire aux exigences de résolution statique des dépendances, et pour éviter les ambiguïtés qui surgiraient lors de l'utilisation d'environnements hybrides tels qu'Electron, la sélection de l'environnement doit être faite explicitement à l'import de la librairie, en important dans le projet soit `yocto_api_nodejs.js`, soit `yocto_api_html.js`.

La librairie peut être intégrée à vos projets de plusieurs manières, selon ce qui convient le mieux à votre projet:

- en copiant directement les fichiers sources TypeScript de notre librairie dans votre projet, et en les ajoutant à votre script de build. Il suffit en général de peu de fichiers pour couvrir la plupart des utilisations. Vous les trouverez dans le sous-répertoire `src` de notre librairie.
- en utilisant la résolution de modules CommonJS, supportée par TypeScript, avec un gestionnaire de packages comme `npm`. Vous trouverez une version transpilée au standard CommonJS dans le sous-répertoire `dist/cjs` de la librairie, y compris les fichiers de définition de type (extension `.d.ts`) et les fichiers de debug (extension `.js.map`) permettant le traçage des erreurs dans les fichiers sources TypeScript. Nous avons aussi publié ces fichiers sur `npmjs` sous le nom `yoctolib-cjs`.
- en utilisant la résolution de modules ECMAScript 2015, aussi supportée par TypeScript, et utilisable directement depuis une page HTML par un référencement relatif. Vous trouverez une version transpilée en module ECMAScript 2015 dans le sous-répertoire `dist/esm` de la librairie, y compris les fichiers de définition de type (extension `.d.ts`) et les fichiers de debug (extension `.js.map`) permettant le traçage des erreurs dans les fichiers sources TypeScript. Nous avons aussi publié ces fichiers sur `npmjs` sous le nom `yoctolib-esm`.

14.1. Utiliser la librairie Yoctopuce pour TypeScript

1. Commencez par installer TypeScript sur votre machine si cela n'est pas déjà fait. Pour cela:

- Installez sur votre machine de développement une version raisonnablement récente de Node.js (par exemple la version 10, ou une plus récente). Vous pouvez l'obtenir gratuitement sur le site officiel: http://nodejs.org. Assurez vous de l'installer entièrement, y compris `npm`, et de l'ajouter à votre system path.
- Installez ensuite TypeScript sur votre machine à l'aide de la commande:

```
npm install -g typescript
```

2. Connectez-vous ensuite sur le site Web de Yoctopuce et téléchargez les éléments suivants:

- La librairie de programmation pour TypeScript[1]
- Le programme VirtualHub[2] pour Windows, Mac OS X ou Linux selon l'OS que vous utilisez. En effet, TypeScript et JavaScript font partie de ces langages qui ne vous permettront pas d'accéder directement aux périphériques USB. C'est pourquoi si vous désirez travailler avec des modules branchés par USB, vous devrez faire tourner la passerelle de Yoctopuce appelée VirtualHub sur la machine à laquelle sont branchés les modules. Vous n'avez en revanche pas besoin d'installer de driver.

3. Décompressez les fichiers de la librairie dans un répertoire de votre choix, et ouvrez une fenêtre de commande dans le répertoire où vous l'avez installée. Lancez la commande suivante pour installer les quelques dépendances qui sont nécessaires au lancement des exemples:

```
npm install
```

Une fois cette commande terminée sans erreur, vous êtes prêt pour l'exploration des exemples. Ceux-ci sont fournis dans deux exemples différents, selon l'environnement d'exécution choisi: `example_html` pour l'exécution de la librairie Yoctopuce dans un navigateur Web, ou `example_nodejs` si vous provoyez d'utiliser la librairie dans un environnement Node.js.

La manière de lancer les exemples dépend de l'environnement choisi. Vous trouverez les instructions détaillées un peu plus loin.

14.2. Petit rappel sur les fonctions asynchrones en JavaScript

JavaScript a été conçu pour éviter toute situation de *concurrence* durant l'exécution. Il n'y a jamais qu'un seul *thread* en JavaScript. Pour gérer les attentes dans les entrées/sorties, JavaScript utilise les opérations asynchrones: lorsqu'une fonction potentiellement bloquante doit être appelée, l'opération est déclenchée mais le flot d'exécution est immédiatement suspendu. Le moteur JavaScript est alors libre pour exécuter d'autres tâches, comme la gestion de l'interface utilisateur par exemple. Lorsque l'opération bloquante se termine finalement, le système relance le code en appelant une fonction de callback, en passant en paramètre le résultat de l'opération, pour permettre de continuer la tâche originale.

L'utilisation d'opérations asynchrones avec des fonctions de callback a la fâcheuse tendance de rentre le code illisible puisqu'elle découpe systématiquement le flot du code en petites fonctions de callback déconnectées les unes des autres. Heureusement, le standard ECMAScript 2015 a apporté les objets *Promise* et la syntaxe `async` / `await` pour la gestion des appels asynchrones:

- une fonction déclarée *async* encapsule automatiquement son résultat dans une promesse

[1] www.yoctopuce.com/FR/libraries.php
[2] www.yoctopuce.com/FR/virtualhub.php

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

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

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

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

Dans la plupart des cas, le typage fort de TypeScript sera là pour vous rappeler d'utiliser `await` lors de l'appel d'une méthode asynchrone.

14.3. Contrôle de la fonction Motor

Il suffit de quelques lignes de code pour piloter un Yocto-Motor-DC. Voici le squelette d'un fragment de code TypeScript qui utilise la fonction Motor.

```
// En Node.js, on référence la librairie via son package NPM
// En HTML, on utiliserait plutôt un path relatif (selon l'environnement)
import { YAPI, YErrorMsg, YModule } from 'yoctolib-cjs/yocto_api_nodejs.js';
import { YMotor } from 'yoctolib-cjs/yocto_motor.js';

[...]
// On active l'accès aux modules locaux à travers le VirtualHub
await YAPI.RegisterHub('127.0.0.1');
[...]

// On récupère l'objet permettant d'intéragir avec le module
let motor: YMotor = YMotor.FindMotor("MOTORCTL-123456.motor");

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

Voyons maintenant en détail ce que font ces quelques lignes.

Import de yocto_api et yocto_motor

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

Pour que `yocto_api` soit correctement lié aux librairies réseau à utiliser pour établir la connexion (soit celles de Node.js, soit celles du navigateur dans le cas d'une application HTML), il faut que

vous référenciez au moins une fois dans votre projet soit la variante `yocto_api_nodejs.js`, soit `yocto_api_html.js`.

Notez que cet exemple importe la librairie au format CommonJS, le plus utilisé avec Node.JS à ce jour, mais si votre projet est construit pour utiliser les modules natifs EcmaScript, il suffit de remplace dans l'import le préfix `yoctolib-cjs` par `yoctolib-esm`.

YAPI.RegisterHub

La méthode `RegisterHub` permet d'indiquer sur quelle machine se trouvent les modules Yoctopuce, ou plus exactement la machine sur laquelle tourne le programme VirtualHub. Dans notre cas l'adresse `127.0.0.1:4444` indique la machine locale, en utilisant le port `4444` (le port standard utilisé par Yoctopuce). Vous pouvez parfaitement changer cette adresse, et mettre l'adresse d'une autre machine sur laquelle tournerait un autre VirtualHub, ou d'un YoctoHub. Si l'hôte n'est pas joignable, la fonction déclanche une exception.

Comme expliqué précédemment, il n'est pas possible d'utiliser directement `RegisterHub ("usb")` en TypeScript, car la machine virtuelle JavaScript n'a pas accès directement aux périphériques USB. Elle doit nécessairement passer par le programme VirtualHub via une connection par l'adresse 127.0.0.1.

YMotor.FindMotor

La méthode `FindMotor` permet de retrouver un moteur 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-Motor-DC avec le numéros de série *MOTORCTL-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *motor* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
motor = YMotor.FindMotor("MOTORCTL-123456.motor")
motor = YMotor.FindMotor("MOTORCTL-123456.MaFonction")
motor = YMotor.FindMotor("MonModule.motor")
motor = YMotor.FindMotor("MonModule.MaFonction")
motor = YMotor.FindMotor("MaFonction")
```

`YMotor.FindMotor` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le moteur.

isOnline

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

set_drivingForce

La méthode `set_drivingForce()` de l'objet renvoyé par `YMotor.FindMotor` permet de régler la puissance, de -100 à +100% envoyée au moteur.

Un exemple concret, en Node.js

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

Si le Yocto-Motor-DC n'est pas branché sur la machine où fonctionne le navigateur internet, remplacez dans l'exemple l'adresse `127.0.0.1` par l'adresse IP de la machine où est branché le Yocto-Motor-DC et où vous avez lancé le VirtualHub.

```
import { YAPI, YErrorMsg, YModule } from 'yoctolib-cjs/yocto_api_nodejs.js';
import { YMotor } from 'yoctolib-cjs/yocto_motor.js'
import { YCurrent } from 'yoctolib-cjs/yocto_current.js'
import { YVoltage } from 'yoctolib-cjs/yocto_voltage.js'
```

```
import { YTemperature } from 'yoctolib-cjs/yocto_temperature.js'

let motor: YMotor;
let current: YCurrent;
let voltage: YVoltage;
let temperature: YTemperature;

async function startDemo(): Promise<void>
{
    await YAPI.LogUnhandledPromiseRejections();

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

    // Select specified device, or use first available one
    let serial: string = process.argv[process.argv.length-1];
    if (serial[8] != '-') {
        // by default use any connected module suitable for the demo
        let anysensor = YMotor.FirstMotor();
        if (anysensor) {
            let module: YModule = await anysensor.get_module();
            serial = await module.get_serialNumber();
        } else {
            console.log('No matching Yocto-Motor-DC connected, check cable !');
            await YAPI.FreeAPI();
            return;
        }
    }
    console.log('Using device ' + serial);

    motor = YMotor.FindMotor(serial + ".motor");
    current = YCurrent.FindCurrent(serial + ".current");
    voltage = YVoltage.FindVoltage(serial + ".voltage");
    temperature = YTemperature.FindTemperature(serial + ".temperature");
    //power is a integer between -100 and 100%
    let power: number = 50;

    // if motor is in error state, reset it.
    if (await motor.get_motorStatus() >= YMotor.MOTORSTATUS_LOVOLT) {
        await motor.resetStatus();
    }
    await motor.drivingForceMove(power, 2000);  // ramp up to power in 2 seconds
    refresh();
}

async function refresh(): Promise<void>
{
    if (await motor.isOnline()) {
        // display motor status
        console.log("Status=" + await motor.get_advertisedValue() + "  " +
            "Voltage=" + await voltage.get_currentValue() + "V  " +
            "Current=" + await current.get_currentValue() / 1000 + "A  " +
            "Temp=" + await temperature.get_currentValue() + "deg C");
    } else {
        console.log('Module not connected');
    }
    setTimeout(refresh, 500);
}

startDemo();
```

Comme décrit au début de ce chapitre, vous devez avoir installé le complateur TypeScript sur votre machine pour essayer ces exemples, et installé les dépendances de la librairie TypeScript. Si vous l'avez fait, vous pouvez maintenant taper la commande suivantes dans le répertoire de l'exemple lui-même, pour finaliser la résolution de ses dépendances:

```
npm install
```

Vous êtes maintenant prêt pour lancer le code d'exemple dans Node.js. La manière la plus simple de le faire est d'utiliser la commande suivante, en remplaçant les [...] par les arguments que vous voulez passer au programme:

```
npm run demo [...]
```

Cette commande, définie dans le fichier `package.json`, a pour effet de compiler le code source TypeScript à l'aide de la simple commande `tsc`, puis de lancer le code compilé dans Node.js.

La compilation utilise les paramètres spécifiés dans le fichier `tsconfig.json`, et produit

- un fichier JavaScript `demo.js`, que Node.js pourra exécuter
- un fichier de debug `demo.js.map`, qui permettra le cas échéant à Node.js de signaler les erreurs en référançant leur origine dans le fichier d'origine en TypeScript.

Notez que le fichier `package.json` de nos exemples référence directement la version locale de la librairie par un path relatif, pour éviter de dupliquer la librairie dans chaque exemple. Bien sur, pour votre application de production, vous pourrez utiliser le package directement depuis le repository npm en l'ajoutant à votre projet à l'aide de la commande:

```
npm install yoctolib-cjs
```

Le même exemple, mais dans un navigateur

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

Aucune installation n'est nécessaire pout utiliser cet exemple HTML, puisqu'il référence la librairie TypeScript via un chemin relatif. Par contre, pour que le navigateur puisse exécuter le code, il faut que la page HTML soit publié par un serveur Web. Nous fournissons un petit serveur de test pour cet usage, que vous pouvez lancer avec la commande:

```
npm run app-server
```

Cette commande va compiler le code d'exemple TypeScript, le mettre à disposition via un serveur HTTP sur le port 3000 et ouvrir un navigateur sur cet exemple. Si vous modifiez le code d'exemple, il sera automatiquement recompilé et il vous suffira de recharger la page sur le navigateur pour retester.

Comme pour l'exemple Node.js, la compilation produit un fichier `.js.map` qui permet de debugger dans le navigateur directement sur le fichier source TypeScript. Notez qu'au moment où cette documentation est rédigée, le debug en format source dans le navigateur fonctionne pour les browsers basés sur Chromium, mais pas encore dans Firefox.

14.4. Contrôle de la partie module

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

```
import { YAPI, YErrorMsg, YModule } from 'yoctolib-cjs/yocto_api_nodejs.js';

async function startDemo(args: string[]): Promise<void>
{
    await YAPI.LogUnhandledPromiseRejections();

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

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

if(process.argv.length < 3) {
    console.log("usage: npm run demo <serial or logicalname> [ ON | OFF ]");
} else {
    startDemo(process.argv.slice(2));
}
```

Chaque propriété `xxx` du module peut être lue grâce à une méthode du type `get_xxxx()`, et les propriétés qui se sont pas en lecture seule peuvent être modifiées à l'aide de la méthode `set_xxx()` Pour plus de détails concernant ces méthodes 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 méthode `set_xxx()` correspondante, cependant cette modification n'a lieu que dans la mémoire vive du module: si le module redémarre, les modifications seront perdues. Pour qu'elle soient mémorisées de manière persistante, il est nécessaire de demander au module de sauvegarder sa configuration courante dans sa mémoire non volatile. Pour cela il faut utiliser la méthode `saveToFlash()`. Inversement il est possible de forcer le module à oublier ses réglages courants en utilisant la méthode `revertFromFlash()`. Ce petit exemple ci-dessous vous permet changer le nom logique d'un module.

```
import { YAPI, YErrorMsg, YModule } from 'yoctolib-cjs/yocto_api_nodejs.js';

async function startDemo(args: string[]): Promise<void>
{
    await YAPI.LogUnhandledPromiseRejections();

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

    // Select the device to use
    let module: YModule = YModule.FindModule(args[0]);
    if(await module.isOnline()) {
        if(args.length > 1) {
            let newname: string = args[1];
            if (!await YAPI.CheckLogicalName(newname)) {
                console.log("Invalid name (" + newname + ")");
                process.exit(1);
            }
            await module.set_logicalName(newname);
            await module.saveToFlash();
        }
        console.log('Current name: '+await module.get_logicalName());
```

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

if(process.argv.length < 3) {
    console.log("usage: npm run demo <serial> [newLogicalName]");
} else {
    startDemo(process.argv.slice(2));
}
```

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

Énumération des modules

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

```
import { YAPI, YErrorMsg, YModule } from 'yoctolib-cjs/yocto_api_nodejs.js';

async function startDemo(): Promise<void>
{
    await YAPI.LogUnhandledPromiseRejections();

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

async function refresh(): Promise<void>
{
    try {
        let errmsg: YErrorMsg = new YErrorMsg();
        await YAPI.UpdateDeviceList(errmsg);

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

startDemo();
```

14.5. Gestion des erreurs

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

votre code doit néanmoins être fait pour se comporter au mieux pour interpréter les erreurs signalées par la librairie.

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

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

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

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `YAPI.DisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur `NomDeClasse.STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `NomDeClasse.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.

15. Utilisation du Yocto-Motor-DC en JavaScript / EcmaScript

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

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

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

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

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

Nous ne recommendons plus l'utilisation de `jspm` dès lors que `async` / `await` sont standardisés.

15.1. Fonctions bloquantes et fonctions asynchrones en JavaScript

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

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

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

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

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

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

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

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

15.2. Utiliser la librairie Yoctopuce pour JavaScript / EcmaScript 2017

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

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

- La librairie de programmation pour Javascript / EcmaScript 2017[1]
- Le programme VirtualHub[2] pour Windows, Mac OS X ou Linux selon l'OS que vous utilisez

Décompressez les fichiers de la librairie dans un répertoire de votre choix, branchez vos modules et lancez le programme VirtualHub. Vous n'avez pas besoin d'installer de driver.

Utiliser la librairie Yoctopuce officielle pour node.js

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

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

```
npm install
```

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

```
node demo.js
```

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

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

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

Utiliser la librairie Yoctopuce dans un navigateur (HTML)

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

Utiliser la librairie Yoctopuce avec des anciennes version de JavaScript

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

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

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

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

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

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

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

Compatibilité avec l'ancienne librairie JavaScript

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

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

par

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

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

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

par

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

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

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

...
```

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

on peut utiliser:

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

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

Ce dernier appel asynchrone peut aussi être formulé comme:

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

15.3. Contrôle de la fonction Motor

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

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

[...]
// On active l'accès aux modules locaux à travers le VirtualHub
await YAPI.RegisterHub('127.0.0.1');
[...]

// On récupère l'objet permettant d'intéragir avec le module
let motor = YMotor.FindMotor("MOTORCTL-123456.motor");

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

Voyons maintenant en détail ce que font ces quelques lignes.

Require de yocto_api et yocto_motor

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

YAPI.RegisterHub

La méthode `RegisterHub` permet d'indiquer sur quelle machine se trouvent les modules Yoctopuce, ou plus exactement la machine sur laquelle tourne le programme VirtualHub. Dans notre cas l'adresse `127.0.0.1:4444` indique la machine locale, en utilisant le port `4444` (le port standard utilisé par Yoctopuce). Vous pouvez parfaitement changer cette adresse, et mettre l'adresse d'une autre machine sur laquelle tournerait un autre VirtualHub, ou d'un YoctoHub. Si l'hôte n'est pas joignable, la fonction déclanche une exception.

YMotor.FindMotor

La méthode `FindMotor` permet de retrouver un moteur 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-Motor-DC avec le numéros de série *MOTORCTL-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *motor* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
motor = YMotor.FindMotor("MOTORCTL-123456.motor")
motor = YMotor.FindMotor("MOTORCTL-123456.MaFonction")
motor = YMotor.FindMotor("MonModule.motor")
motor = YMotor.FindMotor("MonModule.MaFonction")
motor = YMotor.FindMotor("MaFonction")
```

`YMotor.FindMotor` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le moteur.

isOnline

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

set_drivingForce

La méthode `set_drivingForce()` de l'objet renvoyé par `YMotor.FindMotor` permet de régler la puissance, de -100 à +100% envoyée au moteur.

Un exemple concret, en Node.js

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

Si le Yocto-Motor-DC n'est pas branché sur la machine où fonctionne le navigateur internet, remplacez dans l'exemple l'adresse `127.0.0.1` par l'adresse IP de la machine où est branché le Yocto-Motor-DC et où vous avez lancé le VirtualHub.

```
"use strict";

require('yoctolib-es2017/yocto_api.js');
require('yoctolib-es2017/yocto_motor.js');
require('yoctolib-es2017/yocto_current.js');
require('yoctolib-es2017/yocto_voltage.js');
require('yoctolib-es2017/yocto_temperature.js');

let motor, current, voltage, temperature;

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

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

    // Select specified device, or use first available one
    let serial = process.argv[process.argv.length - 1];
    if (serial[8] != '-') {
        // by default use any connected module suitable for the demo
        let anysensor = YMotor.FirstMotor();
        if (anysensor) {
            let module = await anysensor.module();
            serial = await module.get_serialNumber();
        } else {
            console.log('No matching Yocto-Motor-DC connected, check cable !');
            return;
        }
    }
```

```
    console.log('Using device ' + serial);

    motor = YMotor.FindMotor(serial + ".motor");
    current = YCurrent.FindCurrent(serial + ".current");
    voltage = YVoltage.FindVoltage(serial + ".voltage");
    temperature = YTemperature.FindTemperature(serial + ".temperature");
    //power is a integer between -100 and 100%
    let power = 50;

    // if motor is in error state, reset it.
    if (await motor.get_motorStatus() >= YMotor.MOTORSTATUS_LOVOLT) {
        await motor.resetStatus();
    }
    await motor.drivingForceMove(power, 2000);  // ramp up to power in 2 seconds
    refresh();
}

async function refresh() {
    if (await motor.isOnline()) {
        // display motor status
        console.log("Status=" + await motor.get_advertisedValue() + "  " +
            "Voltage=" + await voltage.get_currentValue() + "V  " +
            "Current=" + await current.get_currentValue() / 1000 + "A  " +
            "Temp=" + await temperature.get_currentValue() + "deg C");
    } else {
        console.log('Module not connected');
    }
    setTimeout(refresh, 500);
}

startDemo();
```

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

```
npm install
```

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

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

Le même exemple, mais dans un navigateur

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

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Hello World</title>
  <script src="../../lib/yocto_api.js"></script>
  <script src="../../lib/yocto_motor.js"></script>
  <script src="../../lib/yocto_current.js"></script>
  <script src="../../lib/yocto_voltage.js"></script>
  <script src="../../lib/yocto_temperature.js"></script>
  <script>
    let motor, current, voltage, temperature;

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

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

```
      }
      refresh();
    }

    async function refresh() {
      // Select specified device, or use first available one
      let serial = document.getElementById('serial').value;
      if(serial == '') {
        // by default use any connected module suitable for the demo
        let anyMotor = YMotor.FirstMotor();
        if (anyMotor) {
          let module = await anyMotor.module();
          serial = await module.get_serialNumber();
          document.getElementById('serial').value = serial;
        }
      }

      motor = YMotor.FindMotor(serial + ".motor");
      current = YCurrent.FindCurrent(serial + ".current");
      voltage = YVoltage.FindVoltage(serial + ".voltage");
      temperature = YTemperature.FindTemperature(serial + ".temperature");
      //power is a integer between -100 and 100%

      // if motor is in error state, reset it.
      if (await motor.get_motorStatus() >= YMotor.MOTORSTATUS_LOVOLT) {
        await motor.resetStatus();
      }


      if (await motor.isOnline()) {
        // display motor status
        document.getElementById('msg').value = '';
        document.getElementById("status").value = await motor.get_advertisedValue();
        document.getElementById("current").value = await current.get_currentValue();
        document.getElementById("voltage").value = await voltage.get_currentValue();
        document.getElementById("temperature").value = await temperature.get_currentValue
();
      } else {
        document.getElementById('msg').value = 'Module not connected';
      }
      setTimeout(refresh, 500);
    }

    window.SetPower = function(power)
    {
      motor.drivingForceMove(power, 2000);  // ramp up to power in 2 seconds
    };


    startDemo();
  </script>
</head>
<body>
 Module to use: <input id='serial'>
 <input id='msg' style='color:red;border:none;' readonly><br>
 Status: <input id='status' style='border:none;'readonly><br>
 Current: <input id='current' style='border:none;'readonly><br>
 Voltage: <input id='voltage' style='border:none;'readonly><br>
 Temperature: <input id='temperature' style='border:none;'readonly><br>
 SetPower To :
 <a href='javascript:SetPower(0);'>0 %</a> 
 <a href='javascript:SetPower(25);'>25 %</a> 
 <a href='javascript:SetPower(50);'>50 %</a> 
 <a href='javascript:SetPower(75);'>75 %</a> 
 <a href='javascript:SetPower(100);'>100 %</a> 
</body>
</html>
```

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

15.4. Contrôle de la partie module

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

```
"use strict";

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

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

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

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

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

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

Modifications des réglages du module

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

```
"use strict";

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

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

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

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

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

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

Énumération des modules

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

```
"use strict";

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

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

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

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

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

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

15.5. Gestion des erreurs

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

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

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

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

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `YAPI.DisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur `NomDeClasse.STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `NomDeClasse.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.

16. Utilisation du Yocto-Motor-DC en PHP

PHP est, tout comme Javascript, un langage assez atypique lorsqu'il s'agit de discuter avec du hardware. Néanmoins, utiliser PHP avec des modules Yoctopuce offre l'opportunité de construire très facilement des sites web capables d'interagir avec leur environnement physique, ce qui n'est pas donné à tous les serveurs web. Cette technique trouve une application directe dans la domotique: quelques modules Yoctopuce, un serveur PHP et vous pourrez interagir avec votre maison depuis n'importe ou dans le monde. Pour autant que vous ayez une connexion internet.

PHP fait lui aussi partie de ces langages qui ne vous permettront pas d'accéder directement aux couches matérielles de votre ordinateur. C'est pourquoi vous devrez faire tourner un hub virtuel sur la machine à laquelle sont branchés les modules

Pour démarrer vos essais en PHP, vous allez avoir besoin d'un serveur PHP 5.3 ou plus [1] de préférence en local sur votre machine. Si vous souhaiter utiliser celui qui se trouve chez votre provider internet, c'est possible, mais vous devrez probablement configurer votre routeur ADSL pour qu'il accepte et forwarde les requêtes TCP sur le port 4444.

16.1. Préparation

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

- La librairie de programmation pour PHP[2]
- Le programme VirtualHub[3] pour Windows, Mac OS X ou Linux selon l'OS que vous utilisez

Décompressez les fichiers de la librairie dans un répertoire de votre choix accessible à votre serveur web, branchez vos modules, lancez le programme VirtualHub, et vous pouvez commencer vos premiers test. Vous n'avez pas besoin d'installer de driver.

16.2. Contrôle de la fonction Motor

Il suffit de quelques lignes de code pour piloter un Yocto-Motor-DC. Voici le squelette d'un fragment de code PHP qui utilise la fonction Motor.

```
include('yocto_api.php');
include('yocto_motor.php');
```

[1] Quelques serveurs PHP gratuits: easyPHP pour windows, MAMP pour Mac Os X
[2] www.yoctopuce.com/FR/libraries.php
[3] www.yoctopuce.com/FR/virtualhub.php

```
[...]
// On active l'accès aux modules locaux à travers le VirtualHub
YAPI::RegisterHub('http://127.0.0.1:4444/',$errmsg);
[...]

// On récupère l'objet permettant d'intéragir avec le module
$motor = YMotor::FindMotor("MOTORCTL-123456.motor");

// Pour gérer le hot-plug, on vérifie que le module est là
if($motor->isOnline())
{
    // Utiliser motor->set_drivingForce()
    [...]
}
```

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.php et yocto_motor.php

Ces deux includes PHP permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api.php` doit toujours être inclus, `yocto_motor.php` est nécessaire pour gérer les modules contenant un moteur, comme le Yocto-Motor-DC.

YAPI::RegisterHub

La fonction `YAPI::RegisterHub` permet d'indiquer sur quelle machine se trouve les modules Yoctopuce, ou plus exactemenent sur quelle machine tourne le programme VirtualHub. Dans notre cas l'adresse `127.0.0.1:4444` indique la machine locale, en utilisant le port `4444` (le port standard utilisé par Yoctopuce). Vous pouvez parfaitement changer cette adresse, et mettre l'adresse d'une autre machine sur laquelle tournerait un autre VirtualHub.

YMotor::FindMotor

La fonction `YMotor::FindMotor` permet de retrouver un moteur 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-Motor-DC avec le numéros de série *MOTORCTL-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *motor* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
$motor = YMotor::FindMotor("MOTORCTL-123456.motor");
$motor = YMotor::FindMotor("MOTORCTL-123456.MaFonction");
$motor = YMotor::FindMotor("MonModule.motor");
$motor = YMotor::FindMotor("MonModule.MaFonction");
$motor = YMotor::FindMotor("MaFonction");
```

`YMotor::FindMotor` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le moteur.

isOnline

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

set_drivingForce

La méthode `set_drivingForce()` de l'objet renvoyé par `yFindMotor` permet de régler la puissance, de -100 à +100% envoyée au moteur.

Un exemple réel

Ouvrez votre éditeur de texte préféré[4], recopiez le code ci dessous, sauvez-le dans un répertoire accessible par votre serveur web/PHP avec les fichiers de la librairie, et ouvrez-la page avec votre

[4] Si vous n'avez pas d'éditeur de texte, utilisez Notepad plutôt que Microsoft Word.

browser favori. Vous trouverez aussi ce code dans le répertoire **Examples/Doc-GettingStarted-Yocto-Motor-DC** de la librairie Yoctopuce.

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

```
<HTML>
<HEAD>
 <TITLE>Hello World</TITLE>
</HEAD>
<BODY>
<FORM method='get'>
<?php

  include('yocto_api.php');
  include('yocto_motor.php');
  include('yocto_current.php');
  include('yocto_voltage.php');
  include('yocto_temperature.php');

  // Use explicit error handling rather than exceptions
  YAPI::DisableExceptions();

  // Setup the API to use the VirtualHub on local machine
  if(YAPI::RegisterHub('http://127.0.0.1:4444/',$errmsg) != YAPI::SUCCESS) {
      die("Cannot contact VirtualHub on 127.0.0.1");
  }
  @$power = $_GET['power'];
  @$serial = $_GET['serial'];
  if ($serial == '') {
    $motor = YMotor::FirstMotor();
    if(is_null($motor)) {
      die("No module connected (check USB cable)");
    }
    $serial = $motor->get_module()->get_serialNumber();
  }

  $motor    = YMotor::FindMotor($serial.".motor");
  $current  = YCurrent::FindCurrent($serial.".current");
  $voltage  = YVoltage::FindVoltage($serial.".voltage");
  $temperature = YTemperature::FindTemperature($serial.".temperature");
  Print("Serial:<input name='serial' value='$serial'><br>");
  // lets start the motor
  if ($motor->isOnline()) {
    // if the motor is in error state, reset it.
    if ($motor->get_motorStatus()>=YMotor::MOTORSTATUS_LOVOLT)
      $motor->resetStatus();
    $motor->drivingForceMove($power,2000);  // ramp up to power in 2 seconds
    Printf("status=".$motor->get_advertisedValue().'<br>');
    Printf("current=".($current->get_currentValue()/1000).'A<br>');
    Printf("voltage=".$voltage->get_currentValue().'V<br>');
    Printf("temperature=".$temperature->get_currentValue().'&deg;C<br>');
    Print("Power:<input name='power' value='$power'><br>");
  } else {
    Printf("Module is not connected, check cable.<br>");
  }
  YAPI::FreeAPI();

?>

<input type='submit'>
</FORM>
</BODY>
</HTML>
```

16.3. 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.

```
<HTML>
<HEAD>
 <TITLE>Module Control</TITLE>
</HEAD>
<BODY>
<FORM method='get'>
<?php
  include('yocto_api.php');

  // Use explicit error handling rather than exceptions
  YAPI::DisableExceptions();

  // Setup the API to use the VirtualHub on local machine
  if(YAPI::RegisterHub('http://127.0.0.1:4444/',$errmsg) != YAPI::SUCCESS) {
      die("Cannot contact VirtualHub on 127.0.0.1 : ".$errmsg);
  }

  @$serial = $_GET['serial'];
  if ($serial != '') {
      // Check if a specified module is available online
      $module = YModule::FindModule("$serial");
      if (!$module->isOnline()) {
          die("Module not connected (check serial and USB cable)");
      }
  } else {
      // or use any connected module suitable for the demo
      $module = YModule::FirstModule();
      if($module) { // skip VirtualHub
          $module = $module->nextModule();
      }
      if(is_null($module)) {
          die("No module connected (check USB cable)");
      } else {
          $serial = $module->get_serialnumber();
      }
  }
  Print("Module to use: <input name='serial' value='$serial'><br>");

  if (isset($_GET['beacon'])) {
      if ($_GET['beacon']=='ON')
          $module->set_beacon(Y_BEACON_ON);
      else
          $module->set_beacon(Y_BEACON_OFF);
  }
  printf('serial: %s<br>',$module->get_serialNumber());
  printf('logical name: %s<br>',$module->get_logicalName());
  printf('luminosity: %s<br>',$module->get_luminosity());
  print('beacon: ');
  if($module->get_beacon() == Y_BEACON_ON) {
      printf("<input type='radio' name='beacon' value='ON' checked>ON ");
      printf("<input type='radio' name='beacon' value='OFF'>OFF<br>");
  } else {
      printf("<input type='radio' name='beacon' value='ON'>ON ");
      printf("<input type='radio' name='beacon' value='OFF' checked>OFF<br>");
  }
  printf('upTime: %s sec<br>',intVal($module->get_upTime()/1000));
  printf('USB current: %smA<br>',$module->get_usbCurrent());
  printf('logs:<br><pre>%s</pre>',$module->get_lastLogs());
  YAPI::FreeAPI();
?>
<input type='submit' value='refresh'>
</FORM>
</BODY>
</HTML>
```

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.

```
<HTML>
<HEAD>
 <TITLE>save settings</TITLE>
<BODY>
<FORM method='get'>
<?php
  include('yocto_api.php');

  // Use explicit error handling rather than exceptions
  YAPI::DisableExceptions();

  // Setup the API to use the VirtualHub on local machine
  if(YAPI::RegisterHub('http://127.0.0.1:4444/',$errmsg) != YAPI::SUCCESS) {
      die("Cannot contact VirtualHub on 127.0.0.1");
  }

  @$serial = $_GET['serial'];
  if ($serial != '') {
      // Check if a specified module is available online
      $module = YModule::FindModule("$serial");
      if (!$module->isOnline()) {
          die("Module not connected (check serial and USB cable)");
      }
  } else {
      // or use any connected module suitable for the demo
      $module = YModule::FirstModule();
      if($module) { // skip VirtualHub
          $module = $module->nextModule();
      }
      if(is_null($module)) {
          die("No module connected (check USB cable)");
      } else {
          $serial = $module->get_serialnumber();
      }
  }
  Print("Module to use: <input name='serial' value='$serial'><br>");

  if (isset($_GET['newname'])){
      $newname = $_GET['newname'];
      if (!yCheckLogicalName($newname))
          die('Invalid name');
      $module->set_logicalName($newname);
      $module->saveToFlash();
  }
  printf("Current name: %s<br>", $module->get_logicalName());
  print("New name: <input name='newname' value='' maxlength=19><br>");
  YAPI::FreeAPI();
?>
<input type='submit'>
</FORM>
</BODY>
</HTML>
```

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

Enumération 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

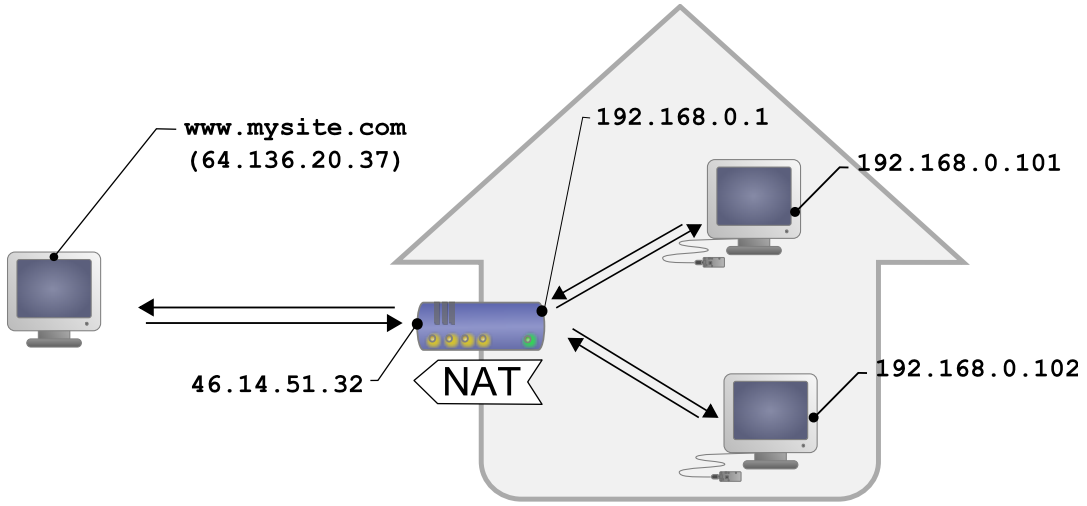
```
<HTML>
<HEAD>
 <TITLE>inventory</TITLE>
</HEAD>
<BODY>
<H1>Device list</H1>
<TT>
<?php
    include('yocto_api.php');
    YAPI::RegisterHub("http://127.0.0.1:4444/");
    $module    = YModule::FirstModule();
    while (!is_null($module)) {
        printf("%s (%s)<br>", $module->get_serialNumber(),
               $module->get_productName());
        $module=$module->nextModule();
    }
    YAPI::FreeAPI();
?>
</TT>
</BODY>
</HTML>
```

16.4. API par callback HTTP et filtres NAT

La librairie PHP est capable de fonctionner dans un mode spécial appelé *Yocto-API par callback HTTP*. Ce mode permet de contrôler des modules Yoctopuce installés derrière un filtre NAT tel qu'un routeur DSL par exemple, et ce sans avoir à un ouvrir un port. L'application typique est le contrôle de modules Yoctopuce situés sur réseau privé depuis un site Web publique.

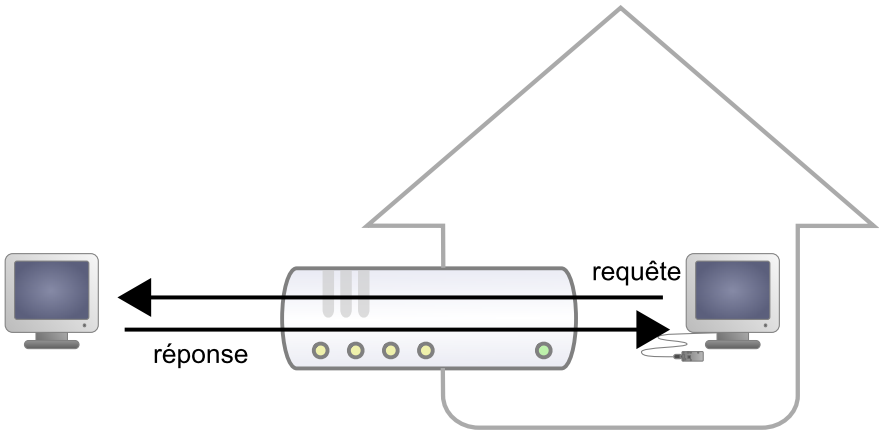
Le filtre NAT, avantages et inconvénients

Un routeur DSL qui effectue de la traduction d'adresse réseau (NAT) fonctionne un peu comme un petit central téléphonique privé: les postes internes peuvent s'appeler l'un l'autre ainsi que faire des appels vers l'extérieur, mais vu de l'extérieur, il n'existe qu'un numéro de téléphone officiel, attribué au central téléphonique lui-même. Les postes internes ne sont pas atteignables depuis l'extérieur.

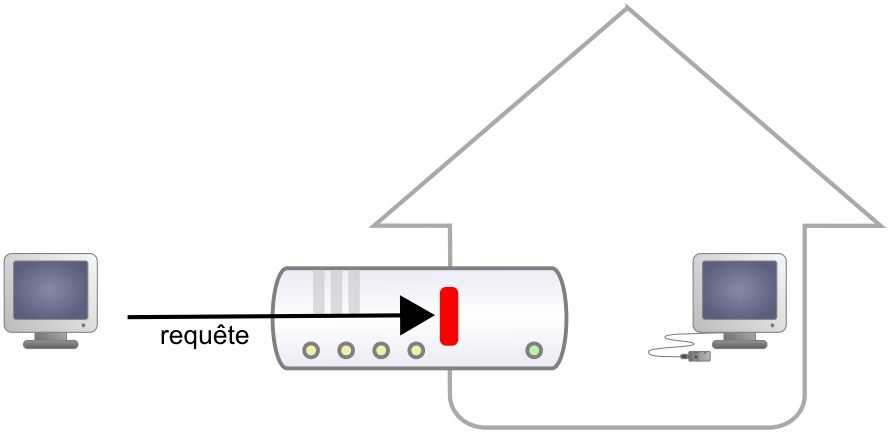


Configuration DSL typique, les machines du LAN sont isolées de l'extérieur par le router DSL

Ce qui, transposé en terme de réseau, donne : les appareils connectés sur un réseau domestique peuvent communiquer entre eux en utilisant une adresse IP locale (du genre 192.168.xxx.yyy), et contacter des serveurs sur Internet par leur adresse publique, mais vu de l'extérieur, il n'y a qu'une seule adresse IP officielle, attribuée au routeur DSL exclusivement. Les différents appareils réseau ne sont pas directement atteignables depuis l'extérieur. C'est assez contraignant, mais c'est une protection relativement efficace contre les intrusions.



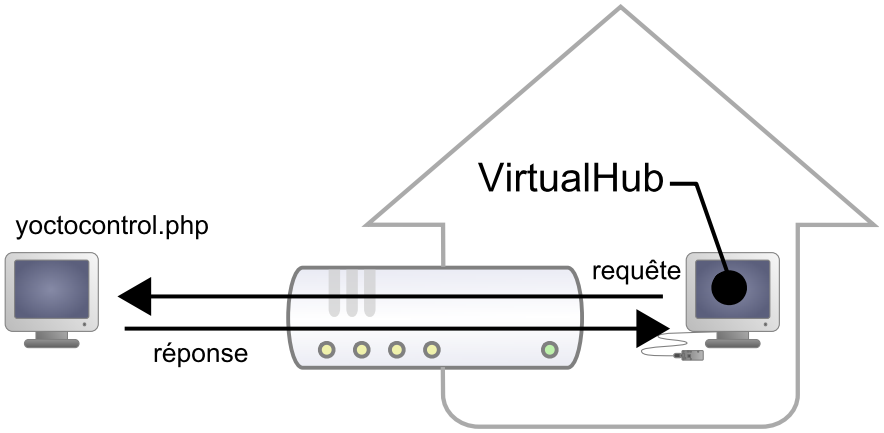
Les réponses aux requêtes venant des machines du LAN sont routées.



Mais les requêtes venant de l'extérieur sont bloquées.

Voir Internet sans être vu représente un avantage de sécurité énorme. Cependant, cela signifie qu'a priori, on ne peut pas simplement monter son propre serveur Web publique chez soi pour une installation domotique et offrir un accès depuis l'extérieur. Une solution à ce problème, préconisée par de nombreux vendeurs de domotique, consiste à donner une visibilité externe au serveur de domotique lui-même, en ouvrant un port et en ajoutant une règle de routage dans la configuration NAT du routeur DSL. Le problème de cette solution est qu'il expose le serveur de domotique aux attaques externes.

L'API par callback HTTP résoud ce problème sans qu'il soit nécessaire de modifier la configuration du routeur DSL. Le script de contrôle des modules est placé sur un site externe, et c'est le *Virtual Hub* qui est chargé de l'appeler à intervalle régulier.



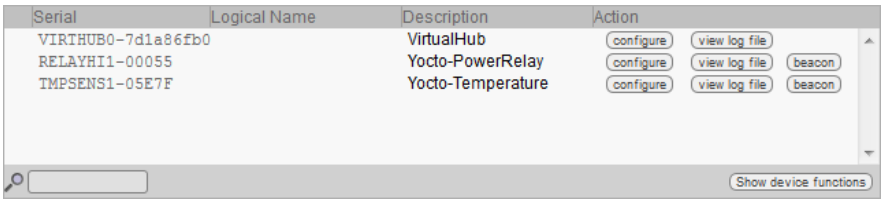
L'API par callback HTTP utilise le VirtualHub, et c'est lui qui initie les requêtes.

Configuration

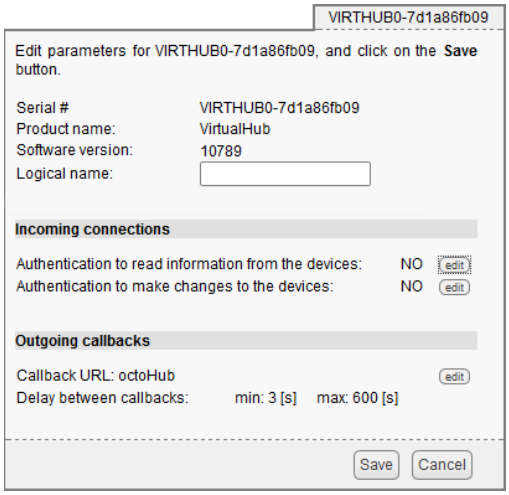
L'API callback se sert donc du *Virtual Hub* comme passerelle. Toutes les communications sont initiées par le *Virtual Hub*, ce sont donc des communication sortantes, et par conséquent parfaitement autorisée par le routeur DSL.

Il faut configurer le *VirtualHub* pour qu'il appelle le script PHP régulièrement. Pour cela il faut:

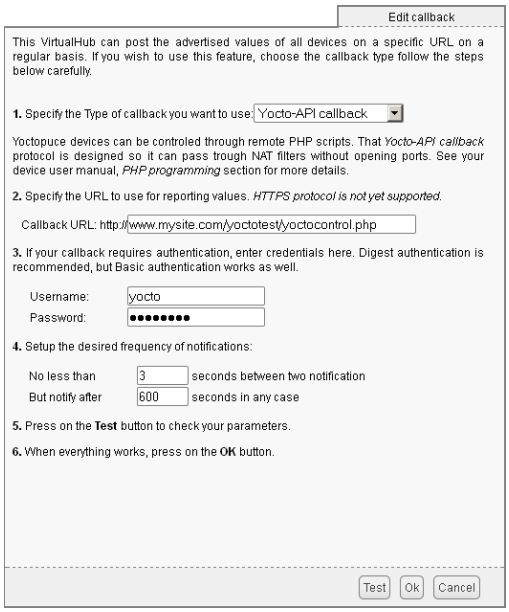
1. Lancer un *VirtualHub*
2. Accéder à son interface, généralement 127.0.0.1:4444
3. Cliquer sur le bouton **configure** de la ligne correspondant au *VirtualHub* lui-même
4. Cliquer sur le bouton **edit** de la section **Outgoing callbacks**



Cliquer sur le bouton "configure" de la première ligne



Cliquer sur le bouton "edit" de la section Outgoing callbacks.



Et choisir "Yocto-API callback".

Il suffit alors de définir l'URL du script PHP et, si nécessaire, le nom d'utilisateur et le mot de passe pour accéder à cette URL. Les méthodes d'authentification supportées sont *basic* et *digest*. La

seconde est plus sûre que la première car elle permet de ne pas transférer le mot de passe sur le réseau.

Utilisation

Du point de vue du programmeur, la seule différence se trouve au niveau de l'appel à la fonction *yRegisterHub*; au lieu d'utiliser une adresse IP, il faut utiliser la chaîne *callback* (ou *http://callback*, qui est équivalent).

```
include("yocto_api.php");
yRegisterHub("callback");
```

La suite du code reste strictement identique. Sur l'interface du *VirtualHub*, il y a en bas de la fenêtre de configuration de l'API par callback HTTP un bouton qui permet de tester l'appel au script PHP.

Il est à noter que le script PHP qui contrôle les modules à distance via l'API par callback HTTP ne peut être appelé que par le *VirtualHub*. En effet, il a besoin des informations postées par le *VirtualHub* pour fonctionner. Pour coder un site Web qui contrôle des modules Yoctopuce de manière interactive, il faudra créer une interface utilisateur qui stockera dans un fichier ou une base de données les actions à effectuer sur les modules Yoctopuce. Ces actions seront ensuite lues puis exécutés par le script de contrôle.

Problèmes courants

Pour que l'API par callback HTTP fonctionne, l'option de PHP *allow_url_fopen* doit être activée. Certains hébergeurs de site web ne l'activent pas par défaut. Le problème se manifeste alors avec l'erreur suivante:

```
error: URL file-access is disabled in the server configuration
```

Pour activer cette option, il suffit de créer dans le même répertoire que le script PHP de contrôle un fichier *.htaccess* contenant la ligne suivante:

```
php_flag "allow_url_fopen" "On"
```

Selon la politique de sécurité de l'hébergeur, il n'est parfois pas possible d'autoriser cette option à la racine du site web, où même d'installer des scripts PHP recevant des données par un POST HTTP. Dans ce cas il suffit de placer le script PHP dans un sous-répertoire.

Limitations

Cette méthode de fonctionnement qui permet de passer les filtres NAT à moindre frais a malgré tout un prix. Les communications étant initiées par le *Virtual Hub* à intervalle plus ou moins régulier, le temps de réaction à un événement est nettement plus grand que si les modules Yoctopuce étaient pilotés en direct. Vous pouvez configurer le temps de réaction dans la fenêtre ad-hoc du *Virtual Hub*, mais il sera nécessairement de quelques secondes dans le meilleur des cas.

Le mode *Yocto-API par callback HTTP* n'est pour l'instant disponible qu'en PHP, EcmaScript (Node.JS) et Java.

16.5. Gestion des erreurs

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

La manière la plus simple de contourner le problème est celle que nous avons employé pour les petits exemples précédents de ce chapitre: avant d'accéder à un module, on vérifie qu'il est en ligne

avec la méthode `isOnline()` et on suppose ensuite qu'il va y rester pendant la fraction de seconde nécessaire à exécuter les lignes de code suivantes. Ce n'est pas parfait, mais ça peut suffire dans certains cas. Il faut toutefois être conscient qu'on ne peut pas totalement exclure une erreur se produisant après le `isOnline()`, qui pourrait faire planter le programme. La seule manière de l'éviter est d'implémenter une des deux techniques de gestion des erreurs décrites ci-dessous.

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

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

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `YAPI.DisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur `NomDeClasse.STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `NomDeClasse.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.

17. Utilisation du Yocto-Motor-DC en VisualBasic .NET

VisualBasic a longtemps été la porte d'entrée privilégiée vers le monde Microsoft. Nous nous devions donc d'offrir notre interface pour ce langage, même si la nouvelle tendance est le C#. Tous les exemples et les modèles de projet sont testés avec Microsoft Visual Basic 2010 Express, disponible gratuitement sur le site de Microsoft [1].

17.1. Installation

Téléchargez la librairie Yoctopuce pour Visual Basic depuis le site web de Yoctopuce[2]. Il n'y a pas de programme d'installation, copiez simplement de contenu du fichier zip dans le répertoire de votre choix. Vous avez besoin essentiellement du contenu du répertoire `Sources`. Les autres répertoires contiennent la documentation et quelques programmes d'exemple. Les projets d'exemple sont des projets Visual Basic 2010, si vous utilisez une version antérieure, il est possible que vous ayez à reconstruire la structure de ces projets.

17.2. Utilisation l'API yoctopuce dans un projet Visual Basic

La librairie Yoctopuce pour Visual Basic .NET se présente sous la forme d'une DLL et de fichiers sources en Visual Basic. La DLL n'est pas une DLL .NET mais une DLL classique, écrite en C, qui gère les communications à bas niveau avec les modules[3]. Les fichiers sources en Visual Basic gèrent la partie haut niveau de l'API. Vous avez donc besoin de cette DLL et des fichiers .vb du répertoire `Sources` pour créer un projet gérant des modules Yoctopuce.

Configuration d'un projet Visual Basic

Les indications ci-dessous sont fournies pour Visual Studio express 2010, mais la procédure est semblable pour les autres versions.

Commencez par créer votre projet, puis depuis le panneau **Explorateur de solutions** effectuez un clic droit sur votre projet, et choisissez **Ajouter** puis **Elément existant**.

Une fenêtre de sélection de fichiers apparaît: sélectionnez le fichier `yocto_api.vb` et les fichiers correspondant aux fonctions des modules Yoctopuce que votre projet va gérer. Dans le doute, vous pouvez aussi sélectionner tous les fichiers.

[1] http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-basic-express
[2] www.yoctopuce.com/FR/libraries.php
[3] Les sources de cette DLL sont disponibles dans l'API C++

Vous avez alors le choix entre simplement ajouter ces fichiers à votre projet, ou les ajouter en tant que lien (le bouton **Ajouter** est en fait un menu déroulant). Dans le premier cas, Visual Studio va copier les fichiers choisis dans votre projet, dans le second Visual Studio va simplement garder un lien sur les fichiers originaux. Il est recommandé d'utiliser des liens, une éventuelle mise à jour de la librairie sera ainsi beaucoup plus facile.

Ensuite, ajoutez de la même manière la dll `yapi.dll`, qui se trouve dans le répertoire `Sources/dll`[4]. Puis depuis la fenêtre **Explorateur de solutions**, effectuez un clic droit sur la DLL, choisissez **Propriété** et dans le panneau **Propriétés**, mettez l'option **Copier dans le répertoire de sortie** à **toujours copier**. Vous êtes maintenant prêt à utiliser vos modules Yoctopuce depuis votre environnement Visual Studio.

Afin de 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.

17.3. Contrôle de la fonction Motor

Il suffit de quelques lignes de code pour piloter un Yocto-Motor-DC. Voici le squelette d'un fragment de code VisualBasic .NET qui utilise la fonction Motor.

```
[...]
' On active la détection des modules sur USB
Dim errmsg As String
YAPI.RegisterHub("usb", errmsg)
[...]

' On récupère l'objet permettant d'intéragir avec le module
Dim motor As YMotor
motor = YMotor.FindMotor("MOTORCTL-123456.motor")

' Pour gérer le hot-plug, on vérifie que le module est là
If (motor.isOnline()) Then
    ' Utiliser motor.set_drivingForce()
    [...]
End If

[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

YAPI.RegisterHub

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

YMotor.FindMotor

La fonction `YMotor.FindMotor` permet de retrouver un moteur 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-Motor-DC avec le numéros de série *MOTORCTL-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *motor* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
motor = YMotor.FindMotor("MOTORCTL-123456.motor")
motor = YMotor.FindMotor("MOTORCTL-123456.MaFonction")
motor = YMotor.FindMotor("MonModule.motor")
motor = YMotor.FindMotor("MonModule.MaFonction")
motor = YMotor.FindMotor("MaFonction")
```

[4] Pensez à changer le filtre de la fenêtre de sélection de fichiers, sinon la DLL n'apparaîtra pas

`YMotor.FindMotor` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le moteur.

isOnline

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

set_drivingForce

La méthode `set_drivingForce()` de l'objet renvoyé par `yFindMotor` permet de régler la puissance, de -100 à +100% envoyée au moteur.

Un exemple réel

Lancez Microsoft VisualBasic et ouvrez le projet exemple correspondant, fourni dans le répertoire **Examples/Doc-GettingStarted-Yocto-Motor-DC** de la librairie Yoctopuce.

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

```
Module Module1

  Private Sub Usage()
    Dim execname = System.AppDomain.CurrentDomain.FriendlyName
    Console.WriteLine("Usage:")
    Console.WriteLine(execname + "  <serial_number>  power")
    Console.WriteLine(execname + "  <logical_name>  power")
    Console.WriteLine(execname + "  any power")
    Console.WriteLine("  power is a integer between -100 and 100%")
    Console.WriteLine("Example:")
    Console.WriteLine(execname + "  any 75")
    System.Threading.Thread.Sleep(2500)
    End
  End Sub

  Sub Main()
    Dim argv() As String = System.Environment.GetCommandLineArgs()
    Dim errmsg As String = ""
    Dim target As String
    Dim power As Integer
    Dim motor As YMotor
    Dim current As YCurrent
    Dim voltage As YVoltage
    Dim temperature As YTemperature

    If argv.Length < 2 Then Usage()

    target = argv(1)
    power = Convert.ToInt32(argv(2))

    REM Setup the API to use local USB devices
    If (YAPI.RegisterHub("usb", errmsg) <> YAPI_SUCCESS) Then
      Console.WriteLine("RegisterHub error: " + errmsg)
      End
    End If

    If target = "any" Then
      motor = YMotor.FirstMotor()
      If motor Is Nothing Then
        Console.WriteLine("No module connected (check USB cable) ")
        End
      End If
      target = motor.get_module().get_serialNumber()
    End If

    motor = YMotor.FindMotor(target + ".motor")
    current = YCurrent.FindCurrent(target + ".current")
    voltage = YVoltage.FindVoltage(target + ".voltage")
    temperature = YTemperature.FindTemperature(target + ".temperature")

    If (motor.isOnline()) Then
      If (motor.get_motorStatus() >= Y_MOTORSTATUS_LOVOLT) Then motor.resetStatus()
      motor.drivingForceMove(power, 2000) REM ramp up to power in 2 seconds
```

```
      While (motor.isOnline())
        REM display motor status
        Console.WriteLine("Status=" + motor.get_advertisedValue() + "  " +
                 "Voltage=" + Str(voltage.get_currentValue()) + "V  " +
                 "Current=" + Str(current.get_currentValue() / 1000) + "A  " +
                 "Temp=" + Str(temperature.get_currentValue()) + "deg C")
        YAPI.Sleep(1000, errmsg) REM wait for one second
      End While

    Else
      Console.WriteLine("Module not connected (check USB cable) ")
    End If
    YAPI.FreeAPI()
  End Sub

End Module
```

17.4. Contrôle de la partie module

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

```
Imports System.IO
Imports System.Environment

Module Module1

  Sub usage()
    Console.WriteLine("usage: demo <serial or logical name> [ON/OFF]")
    End
  End Sub

  Sub Main()
    Dim argv() As String = System.Environment.GetCommandLineArgs()
    Dim errmsg As String = ""
    Dim m As ymodule

    If (YAPI.RegisterHub("usb", errmsg) <> YAPI_SUCCESS) Then
      Console.WriteLine("RegisterHub error:" + errmsg)
      End
    End If

    If argv.Length < 2 Then usage()

    m = YModule.FindModule(argv(1)) REM use serial or logical name
    If (m.isOnline()) Then
      If argv.Length > 2 Then
        If argv(2) = "ON" Then m.set_beacon(Y_BEACON_ON)
        If argv(2) = "OFF" Then m.set_beacon(Y_BEACON_OFF)
      End If
      Console.WriteLine("serial:       " + m.get_serialNumber())
      Console.WriteLine("logical name: " + m.get_logicalName())
      Console.WriteLine("luminosity:   " + Str(m.get_luminosity()))
      Console.Write("beacon:       ")
      If (m.get_beacon() = Y_BEACON_ON) Then
        Console.WriteLine("ON")
      Else
        Console.WriteLine("OFF")
      End If
      Console.WriteLine("upTime:       " + Str(m.get_upTime() / 1000) + " sec")
      Console.WriteLine("USB current:  " + Str(m.get_usbCurrent()) + " mA")
      Console.WriteLine("Logs:")
      Console.WriteLine(m.get_lastLogs())
    Else
      Console.WriteLine(argv(1) + " not connected (check identification and USB cable)")
    End If
    YAPI.FreeAPI()
  End Sub

End Module
```

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.

```
Module Module1


  Sub usage()

    Console.WriteLine("usage: demo <serial or logical name> <new logical name>")
    End
  End Sub

  Sub Main()
    Dim argv() As String = System.Environment.GetCommandLineArgs()
    Dim errmsg As String = ""
    Dim newname As String
    Dim m As YModule

    If (argv.Length <> 3) Then usage()

    REM Setup the API to use local USB devices
    If YAPI.RegisterHub("usb", errmsg) <> YAPI_SUCCESS Then
      Console.WriteLine("RegisterHub error: " + errmsg)
      End
    End If

    m = YModule.FindModule(argv(1)) REM use serial or logical name
    If m.isOnline() Then
      newname = argv(2)
      If (Not YAPI.CheckLogicalName(newname)) Then
        Console.WriteLine("Invalid name (" + newname + ")")
        End
      End If
      m.set_logicalName(newname)
      m.saveToFlash() REM do not forget this
      Console.Write("Module: serial= " + m.get_serialNumber)
      Console.Write(" / name= " + m.get_logicalName())
    Else
      Console.Write("not connected (check identification and USB cable")
    End If
    YAPI.FreeAPI()

  End Sub

End Module
```

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 `Nothing`. Ci-dessous un petit exemple listant les module connectés

```
Module Module1

  Sub Main()
    Dim M As ymodule
    Dim errmsg As String = ""

    REM Setup the API to use local USB devices
    If YAPI.RegisterHub("usb", errmsg) <> YAPI_SUCCESS Then
      Console.WriteLine("RegisterHub error: " + errmsg)
      End
    End If

    Console.WriteLine("Device list")
    M = YModule.FirstModule()
    While M IsNot Nothing
      Console.WriteLine(M.get_serialNumber() + " (" + M.get_productName() + ")")
      M = M.nextModule()
    End While
    YAPI.FreeAPI()
  End Sub

End Module
```

17.5. Gestion des erreurs

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

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

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

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

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `YAPI.DisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur `NomDeClasse.STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `NomDeClasse.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.

18. Utilisation du Yocto-Motor-DC en Delphi

Delphi est l'héritier de Turbo-Pascal. A l'origine, Delphi était produit par Borland, mais c'est maintenant Embarcadero qui l'édite. Sa force réside dans sa facilité d'utilisation, il permet à quiconque ayant des notions de Pascal de programmer une application Windows en deux temps trois mouvements. Son seul défaut est d'être payant[1].

Les librairies pour Delphi sont fournies non pas sous forme de composants VCL, mais directement sous forme de fichiers source. Ces fichiers sont compatibles avec la plupart des version de Delphi [2].

Afin des les garder simples, tous les exemples fournis dans cette documentation sont des applications consoles. Il va de soit que le fonctionnement des librairies est strictement identique avec des applications VCL.

Vous allez rapidement vous rendre compte que l'API Delphi défini 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.

18.1. Préparation

Connectez-vous sur le site de Yoctopuce et téléchargez la la librairie Yoctopuce pour Delphi[3]. Décompressez le tout dans le répertoire de votre choix, et ajoutez le sous-répertoire *sources* de l'archive dans la liste des répertoires des librairies de Delphi[4].

Par défaut la librairie Yoctopuce pour Delphi utilise une DLL *yapi.dll*, toutes les applications que vous créerez avec Delphi devront avoir accès à cette DLL. Le plus simple est de faire en sorte qu'elle soit présente dans le même répertoire que l'exécutable de votre application.

18.2. Contrôle de la fonction Motor

Il suffit de quelques lignes de code pour piloter un Yocto-Motor-DC. Voici le squelette d'un fragment de code Delphi qui utilise la fonction Motor.

```
uses yocto_api, yocto_motor;
```

[1] En fait, Borland a diffusé des versions gratuites (pour usage personnel) de Delphi 2006 et Delphi 2007, en cherchant un peu sur internet il est encore possible de les télécharger.
[2] Les librairies Delphi sont régulièrement testées avec Delphi 5 et Delphi XE2
[3] www.yoctopuce.com/FR/libraries.php
[4] Utilisez le menu **outils / options d'environement**

```
var errmsg: string;
    motor: TYMotor;

[...]
// On active la détection des modules sur USB
yRegisterHub('usb',errmsg)
[...]

// On récupère l'objet permettant d'intéragir avec le module
motor = yFindMotor("MOTORCTL-123456.motor")

// Pour gérer le hot-plug, on vérifie que le module est là
if motor.isOnline() then
    begin
        // use motor.set_drivingForce()
        [...]
    end;
[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api et yocto_motor

Ces deux unités permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api` doit toujours être utilisé, `yocto_motor` est nécessaire pour gérer les modules contenant un moteur, comme le Yocto-Motor-DC.

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.

yFindMotor

La fonction `yFindMotor` permet de retrouver un moteur 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-Motor-DC avec le numéros de série *MOTORCTL-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *motor* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
motor := yFindMotor("MOTORCTL-123456.motor");
motor := yFindMotor("MOTORCTL-123456.MaFonction");
motor := yFindMotor("MonModule.motor");
motor := yFindMotor("MonModule.MaFonction");
motor := yFindMotor("MaFonction");
```

`yFindMotor` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le moteur.

isOnline

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

set_drivingForce

La méthode `set_drivingForce()` de l'objet renvoyé par `yFindMotor` permet de régler la puissance, de -100 à +100% envoyée au moteur.

Un exemple réel

Lancez votre environnement Delphi, copiez la DLL yapi.dll dans un répertoire et créez une nouvelle application console dans ce même répertoire, et copiez-coller le code ci dessous.

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.

```
program helloworld;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Windows,
  yocto_api,
  yocto_motor,yocto_current,yocto_voltage,yocto_temperature;

procedure usage();
  var
    execname:string;
  begin
    execname := ExtractFileName(paramstr(0));
    WriteLn('Usage:');
    WriteLn(execname + ' <serial_number> power ');
    WriteLn(execname + ' <logical_name> power');
    WriteLn(execname + ' any <channel> power');
    WriteLn('power is a integer between -100 and 100%');
    WriteLn('Example:');
    WriteLn(execname + ' any 75');
    sleep(2500);
    halt;
  end;

var
 errmsg,target:string;
 power:integer;
 motor:TYMotor;
 temperature:TYTemperature;
 current:TYCurrent;
 voltage:TYVoltage;

 m : TYModule;

begin
  if (paramcount<2) then usage();

  // parse command line
  target  :=  UpperCase(paramstr(1));
  power   :=  strtoint(paramstr(2));
  writeln(power);

  // Setup the API to use local USB devices
  if (YRegisterHub('usb', errmsg) <> YAPI_SUCCESS)  then
    begin
      writeln('RegisterHub error: ' + errmsg);
      halt;
    end;

  if (target='ANY') then
    begin
      // find the serial# of the first available motor
      motor :=  YFirstMotor();
      if (motor =nil) then
       begin
         writeln('No module connected (check USB cable)');
         halt;
       end;
      // retreive the hosting device serial
      m :=  motor.get_module();
      target := m.get_serialNumber();
     end;

  Writeln('using ' + target);

  // retreive motor, current, voltage and temperature features from the device
  motor       := YFindMotor(target + '.motor');
  current     := YFindCurrent(target + '.current');
  temperature := YFindTemperature(target + '.temperature');
  voltage     := YFindVoltage(target + '.voltage');

  // lets start the motor
  if (motor.isOnline()) then
    begin
       // if motor is in error state, reset it.
       if ( motor.get_motorStatus>=Y_MOTORSTATUS_LOVOLT) then motor.resetStatus();
       motor.drivingForceMove(power,2000);  // ramp up to power in 2 seconds
       while motor.isOnline() do
```

```
        begin
          // display motor status
          Write('Status=',motor.get_advertisedValue(),'  ');
          Write('Voltage=',FloatToStrF(voltage.get_currentValue(),ffFixed,3,1),'V  ' );
          Write('Current=',FloatToStrF(current.get_currentValue()/1000,ffFixed,3,1),'A  ');
          Writeln('Temp=',FloatToStrF(temperature.get_currentValue(),ffFixed,3,1),'deg C');
          Ysleep(1000,errmsg); // wait for one second
        end;
  end else writeln('Module not connected (check identification and USB cable)');
  yFreeAPI();

end.
```

18.3. 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.

```
program modulecontrol;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  yocto_api;

const
  serial = 'MOTORCTL-123456'; // use serial number or logical name

procedure refresh(module:Tymodule) ;
  begin
    if (module.isOnline())  then
     begin
       Writeln('');
       Writeln('Serial       : ' + module.get_serialNumber());
       Writeln('Logical name : ' + module.get_logicalName());
       Writeln('Luminosity   : ' + intToStr(module.get_luminosity()));
       Write('Beacon    :');
       if  (module.get_beacon()=Y_BEACON_ON) then Writeln('on')
                                             else Writeln('off');
       Writeln('uptime       : ' + intToStr(module.get_upTime() div 1000)+'s');
       Writeln('USB current  : ' + intToStr(module.get_usbCurrent())+'mA');
       Writeln('Logs         : ');
       Writeln(module.get_lastlogs());
       Writeln('');
       Writeln('r : refresh / b:beacon ON / space : beacon off');
     end
    else Writeln('Module not connected (check identification and USB cable)');
  end;


procedure beacon(module:Tymodule;state:integer);
  begin
    module.set_beacon(state);
    refresh(module);
  end;

var
  module : TYModule;
  c      : char;
  errmsg : string;

begin
  // Setup the API to use local USB devices
  if yRegisterHub('usb', errmsg)<>YAPI_SUCCESS then
  begin
    Write('RegisterHub error: '+errmsg);
    exit;
  end;

  module := yFindModule(serial);
  refresh(module);

  repeat
```

```
    read(c);
    case c of
     'r': refresh(module);
     'b': beacon(module,Y_BEACON_ON);
     ' ': beacon(module,Y_BEACON_OFF);
    end;
  until  c = 'x';
  yFreeAPI();
end.
```

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.

```
program savesettings;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  yocto_api;

const
  serial = 'MOTORCTL-123456'; // use serial number or logical name

var
  module  : TYModule;
  errmsg  : string;
  newname : string;

begin
  // Setup the API to use local USB devices
  if yRegisterHub('usb', errmsg)<>YAPI_SUCCESS then
  begin
    Write('RegisterHub error: '+errmsg);
    exit;
  end;

  module := yFindModule(serial);
  if (not(module.isOnline)) then
   begin
     writeln('Module not connected (check identification and USB cable)');
     exit;
   end;

  Writeln('Current logical name : '+module.get_logicalName());
  Write('Enter new name : ');
  Readln(newname);
  if (not(yCheckLogicalName(newname))) then
   begin
     Writeln('invalid logical name');
     exit;
   end;
  module.set_logicalName(newname);
  module.saveToFlash();
  yFreeAPI();
  Writeln('logical name is now : '+module.get_logicalName());
end.
```

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.

Énumération 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 `nil`. Ci-dessous un petit exemple listant les module connectés

```
program inventory;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  yocto_api;

var
  module : TYModule;
  errmsg : string;

begin
  // Setup the API to use local USB devices
  if yRegisterHub('usb', errmsg)<>YAPI_SUCCESS then
  begin
    Write('RegisterHub error: '+errmsg);
    exit;
  end;

  Writeln('Device list');

  module := yFirstModule();
  while module<>nil  do
   begin
     Writeln( module.get_serialNumber()+' ('+module.get_productName()+')');
     module := module.nextModule();
   end;
  yFreeAPI();

end.
```

18.4. Gestion des erreurs

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

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

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

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

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

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `YAPI.DisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur `NomDeClasse.STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `NomDeClasse.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.

19. Utilisation du Yocto-Motor-DC avec Universal Windows Platform

Universal Windows Platform, abrégé UWP, est n'est pas un langage à proprememt parler mais une plate-forme logicielle créée par Micorosft. Cette platform permet d'executer un nouveau type d'applications : les application universelle Windows. Ces applicaiton peuvent fonctionner sur toutes les machines qui fonctione sous Windows 10. Cela comprend les PCs, les tablettes, les smartphones, la XBox One, mais aussi Windows IoT Core.

La librairie Yoctopuce UWP permet d'utiliser les modules Yoctopuce dans une application universelle Winodws et est entièrement écrite C#. Elle peut être ajoutée a un projet Visual Studio 2017[1].

19.1. Fonctions bloquantes et fonctions asynchrones

La librairie Universal Windows Platform n'utilise pas l'API win32 mais uniquement l'API Windows Runtime qui est disponible sur toutes les versions de Windows 10 et pour n'importe quelle architecture. Grâce à cela la librairie UWP peut être utilisé sur toutes les versions de Windows 10, y compris Windows 10 IoT Core.

Cependant, l'utilisation des nouvelles API UWP n'est pas sans conséquence: l'API Windows Runtime pour accéder aux ports USB est asynchrone, et par conséquent la librairie Yoctopuce doit aussi être asynchrone. Concrètement les méthodes asynchrones ne retournent pas directement le résultat mais un objet `Task` ou `Task<>` et le résultat peut être obtenu plus tard. Fort heureusement, le langage C# version 6 supporte les mots-clefs `async` et `await` qui simplifie beaucoup l'utilisation de ces fonctions. Il est ainsi possible d'utiliser les fonctions asynchrones de la même manière que les fonctions traditionnelles pour autant que les deux règles suivantes soient respectées:

- La méthode est déclarée comme asynchrone à l'aide du mot-clef `async`
- le mot-clef `await` est ajouté lors de l'utilisation d'une fonction asynchrone

Exemple:

```
async Task<int> MyFunction(int  val)
{
    // do some long computation
    ...

    return result;
}
```

[1] https://www.visualstudio.com/fr/vs/

```
int res = await MyFunction(1234);
```

Notre librairie suit ces deux règles et peut donc dutiliser la notation `await`.

Pour ne pas devoir vous poser la question pour chaque méthode de savoir si elle est asynchrone ou pas, la convention est la suivante: **toutes les méthodes publiques** de la librairie UWP **sont asyncrones**, c'est-à-dire qui faut les appeler en ajoutant le mot clef `await`, **sauf**:

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

19.2. Installation

Téléchargez la librairie Yoctopuce pour Universal Windows Platform depuis le site web de Yoctopuce [2]. Il n'y a pas de programme d'installation, copiez simplement de contenu du fichier zip dans le répertoire de votre choix. Vous avez besoin essentiellement du contenu du répertoire `Sources`. Les autres répertoires contiennent la documentation et quelques programmes d'exemple. Les projets d'exemple sont des projets Visual Studio 2017 qui est disponible sur le site de Microsoft [3].

19.3. Utilisation l'API Yoctopuce dans un projet Visual Studio

Commencez par créer votre projet , puis depuis le panneau **Explorateur de solutions** effectuez un clic droit sur votre projet, et choisissez **Ajouter** puis **Élément existant** .

Une fenêtre de sélection de fichiers apparaît: sélectionnez tous les fichiers du répertoire `Sources` de la librairie.

Vous avez alors le choix entre simplement ajouter ces fichiers à votre projet, ou les ajouter en tant que lien (le bouton **Ajouter** est en fait un menu déroulant). Dans le premier cas, Visual Studio va copier les fichiers choisis dans votre projet, dans le second Visual Studio va simplement garder un lien sur les fichiers originaux. Il est recommandé d'utiliser des liens, une éventuelle mise à jour de la librairie sera ainsi beaucoup plus facile.

Le fichier Package.appxmanifest

Par défaut, une application Universal Windows n'a pas le droit daccéder aux ports USB. Si l'on désire accéder à un périphérique USB, il faut impérativement le déclarer dans le fichier `Package.appxmanifest`.

Malheureusement, la fenêtre d'édition de ce fichier ne permet pas cette opération et il faut modifier le fichier `Package.appxmanifest` à la main. Dans le panneau "Solutions Explorer", faites un clic droit sur le fichier `Package.appxmanifest` et sélectionner "View Code".

Dans ce fichier XML, il faut rajouter un nud `DeviceCapability` dans le nud `Capabilities`. Ce nud doit avoir un attribut "Name" qui vaut "humaninterfacedevice".

A lintérieur de ce nud, il faut déclarer tous les modules qui peuvent être utilisés. Concrètement, pour chaque module, il faut ajouter un nud "Device" avec un attribut "Id" dont la valeur est une chaîne de caractères "vidpid:`USB_VENDORID USB_DEVICE_ID`". Le `USB_VENDORID` de Yoctopuce est 24e0 et le `USB_DEVICE_ID` de chaque module Yoctopuce peut être trouvé dans la

[2] www.yoctopuce.com/FR/libraries.php
[3] https://www.visualstudio.com/downloads/

documentation dans la section "Caractéristiques". Pour finir, le n u d "Device" doit contenir un n u d "Function" avec l'attribut "Type" dont la valeur est "usage:ff00 0001".

Pour le Yocto-Motor-DC voici ce qu'il faut ajouter dans le n u d "Capabilities":

```
<DeviceCapability Name="humaninterfacedevice">
    <!-- Yocto-Motor-DC -->
    <Device Id="vidpid:24e0 0016">
      <Function Type="usage:ff00 0001" />
    </Device>
  </DeviceCapability>
```

Malheureusement, il n'est pas possible d'écrire un règle qui autorise tous les modules Yoctopuce, par conséquent il faut impérativement ajouter chaque module que l'on désire utiliser.

19.4. Contrôle de la fonction Motor

Il suffit de quelques lignes de code pour piloter un Yocto-Motor-DC. Voici le squelette d'un fragment de code c# qui utilise la fonction Motor.

```
[...]
// On active la détection des modules sur USB
await YAPI.RegisterHub("usb");
[...]

// On récupère l'objet permettant d'intéragir avec le module
YMotor motor = YMotor.FindMotor("MOTORCTL-123456.motor");

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

[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

YAPI.RegisterHub

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

YMotor.FindMotor

La fonction `YMotor.FindMotor` permet de retrouver un moteur 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-Motor-DC avec le numéros de série *MOTORCTL-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *motor* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
motor = YMotor.FindMotor("MOTORCTL-123456.motor");
motor = YMotor.FindMotor("MOTORCTL-123456.MaFonction");
motor = YMotor.FindMotor("MonModule.motor");
motor = YMotor.FindMotor("MonModule.MaFonction");
motor = YMotor.FindMotor("MaFonction");
```

`YMotor.FindMotor` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le moteur.

isOnline

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

set_drivingForce

La méthode `set_drivingForce()` de l'objet renvoyé par `YMotor.FindMotor` permet de régler la puissance, de -100 à +100% envoyée au moteur.

19.5. Un exemple concret

Lancez Visual Studio et ouvrez le projet correspondant, fourni dans le répertoire **Examples/Doc-GettingStarted-Yocto-Motor-DC** de la librairie Yoctopuce.

Le projets Visual Studio contient de nombreux fichiers dont la plupart ne sont pas liés à l'utilisation de la librairie Yoctopuce. Pour simplifier la lecture du code nous avons regroupé tout le code qui utilise la librairie dans la classe `Demo` qui se trouve dans le fichier `demo.cs`. Les propriétés de cette classe correspondent aux différentes champs qui sont affichés à l'écran, et la méthode `Run()` contient le code qui est exécuté quand le bouton "Start" est pressé.

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.

```
using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.UI.Xaml.Controls;
using com.yoctopuce.YoctoAPI;

namespace Demo
{
  public class Demo : DemoBase
  {
    public string HubURL { get; set; }
    public string Target { get; set; }
    public string Power { get; set; }

    public override async Task<int> Run()
    {
      try {
        await YAPI.RegisterHub(HubURL);
        YMotor motor;
        YCurrent current;
        YVoltage voltage;
        YTemperature temperature;

        if (Target.ToLower() == "any") {
          // find the serial# of the first available motor
          motor = YMotor.FirstMotor();
          if (motor == null) {
            WriteLine("No module connected (check USB cable) ");
            return -1;
          }

          Target = await (await motor.get_module()).get_serialNumber();
        }

        int power = Convert.ToInt32(Power);
        motor = YMotor.FindMotor(Target + ".motor");
        current = YCurrent.FindCurrent(Target + ".current");
        voltage = YVoltage.FindVoltage(Target + ".voltage");
        temperature = YTemperature.FindTemperature(Target + ".temperature");

        // lets start the motor
        if (await motor.isOnline()) {
          // if motor is in error state, reset it.
          if (await motor.get_motorStatus() >= YMotor.MOTORSTATUS_LOVOLT) {
            await motor.resetStatus();
          }

          await motor.drivingForceMove(power, 2000); // ramp up to power in 2 seconds
```

```
          while (await motor.isOnline()) {
            // display motor status
            WriteLine("Status=" + await motor.get_advertisedValue() + "  "
                      + "Voltage=" + await voltage.get_currentValue() + "V  "
                      + "Current=" + await current.get_currentValue() / 1000 + "A  "
                      + "Temp=" + await temperature.get_currentValue() + "deg C");
            await YAPI.Sleep(1000); // wait for one second
          }
        }
      } catch (YAPI_Exception ex) {
        WriteLine("error: " + ex.Message);
      }

      YAPI.FreeAPI();
      return 0;
    }
  }
}
```

19.6. Contrôle de la partie module

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

```
using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.UI.Xaml.Controls;
using com.yoctopuce.YoctoAPI;

namespace Demo
{
  public class Demo : DemoBase
  {

    public string HubURL { get; set; }
    public string Target { get; set; }
    public bool Beacon { get; set; }

    public override async Task<int> Run()
    {
      YModule m;
      string errmsg = "";

      if (await YAPI.RegisterHub(HubURL) != YAPI.SUCCESS) {
        WriteLine("RegisterHub error: " + errmsg);
        return -1;
      }
      m = YModule.FindModule(Target + ".module"); // use serial or logical name
      if (await m.isOnline()) {
        if (Beacon) {
          await m.set_beacon(YModule.BEACON_ON);
        } else {
          await m.set_beacon(YModule.BEACON_OFF);
        }

        WriteLine("serial: " + await m.get_serialNumber());
        WriteLine("logical name: " + await m.get_logicalName());
        WriteLine("luminosity: " + await m.get_luminosity());
        Write("beacon: ");
        if (await m.get_beacon() == YModule.BEACON_ON)
          WriteLine("ON");
        else
          WriteLine("OFF");
        WriteLine("upTime: " + (await m.get_upTime() / 1000) + " sec");
        WriteLine("USB current: " + await m.get_usbCurrent() + " mA");
        WriteLine("Logs:\r\n" + await m.get_lastLogs());
      } else {
        WriteLine(Target + " not connected  on" + HubURL +
                  "(check identification and USB cable)");
      }
      YAPI.FreeAPI();
      return 0;
```

```
    }
  }
}
```

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

Modifications des réglages du module

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

```
using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.UI.Xaml.Controls;
using com.yoctopuce.YoctoAPI;

namespace Demo
{
  public class Demo : DemoBase
  {

    public string HubURL { get; set; }
    public string Target { get; set; }
    public string LogicalName { get; set; }

    public override async Task<int> Run()
    {
      try {
        YModule m;

        await YAPI.RegisterHub(HubURL);

        m = YModule.FindModule(Target); // use serial or logical name
        if (await m.isOnline()) {
          if (!YAPI.CheckLogicalName(LogicalName)) {
            WriteLine("Invalid name (" + LogicalName + ")");
            return -1;
          }

          await m.set_logicalName(LogicalName);
          await m.saveToFlash(); // do not forget this
          Write("Module: serial= " + await m.get_serialNumber());
          WriteLine(" / name= " + await m.get_logicalName());
        } else {
          Write("not connected (check identification and USB cable");
        }
      } catch (YAPI_Exception ex) {
        WriteLine("RegisterHub error: " + ex.Message);
      }
      YAPI.FreeAPI();
      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 `YModule.saveToFlash()` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

Enumeration des modules

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

```
using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.UI.Xaml.Controls;
using com.yoctopuce.YoctoAPI;

namespace Demo
{
  public class Demo : DemoBase
  {
    public string HubURL { get; set; }

    public override async Task<int> Run()
    {
      YModule m;
      try {
        await YAPI.RegisterHub(HubURL);

        WriteLine("Device list");
        m = YModule.FirstModule();
        while (m != null) {
          WriteLine(await m.get_serialNumber()
                    + " (" + await m.get_productName() + ")");
          m = m.nextModule();
        }
      } catch (YAPI_Exception ex) {
        WriteLine("Error:" + ex.Message);
      }
      YAPI.FreeAPI();
      return 0;
    }
  }
}
```

19.7. Gestion des erreurs

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

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

Dans la librairie Universal Windows Platform, le traitement d'erreur est implémenté au moyen d'exceptions. Vous devrez donc intercepter et traiter correctement ces exceptions si vous souhaitez avoir un projet fiable qui ne crashera pas des que vous débrancherez un module.

Les exceptions lancées de la librairie sont toujours de type YAPI_Exception, ce qui permet facilement de les séparer des autres exceptions dans un bloc `try{...} catch{...}`.

Exemple:

```
try {
```

```
        ....
} catch (YAPI_Exception ex) {
        Debug.WriteLine("Exception from Yoctopuce lib:" + ex.Message);
} catch (Exception ex) {
        Debug.WriteLine("Other exceptions :" + ex.Message);
}
```

20. Utilisation du Yocto-Motor-DC en Objective-C

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

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

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

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

20.1. Contrôle de la fonction Motor

Il suffit de quelques lignes de code pour piloter un Yocto-Motor-DC. Voici le squelette d'un fragment de code Objective-C qui utilise la fonction Motor.

```
#import "yocto_api.h"
#import "yocto_motor.h"

...
NSError *error;
[YAPI RegisterHub:@"usb": &error]
...
// On récupère l'objet représentant le module (ici connecté en local sur USB)
motor = [YMotor FindMotor:@"MOTORCTL-123456.motor"];
```

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

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.h et yocto_motor.h

Ces deux fichiers importés permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api.h` doit toujours être utilisé, `yocto_motor.h` est nécessaire pour gérer les modules contenant un moteur, comme le Yocto-Motor-DC.

[YAPI RegisterHub]

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

[Motor FindMotor]

La fonction `[Motor FindMotor]`, permet de retrouver un moteur 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-Motor-DC avec le numéros de série *MOTORCTL-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *motor* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YMotor *motor = [YMotor FindMotor:@"MOTORCTL-123456.motor"];
YMotor *motor = [YMotor FindMotor:@"MOTORCTL-123456.MaFonction"];
YMotor *motor = [YMotor FindMotor:@"MonModule.motor"];
YMotor *motor = [YMotor FindMotor:@"MonModule.MaFonction"];
YMotor *motor = [YMotor FindMotor:@"MaFonction"];
```

`[YMotor FindMotor]` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le moteur.

isOnline

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

set_drivingForce

La méthode `set_drivingForce()` de l'objet renvoyé par `YMotor.FindMotor` permet de régler la puissance, de -100 à +100% envoyée au moteur.

Un exemple réel

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

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

```
#import <Foundation/Foundation.h>
#import "yocto_api.h"
#import "yocto_motor.h"
#import "yocto_current.h"
#import "yocto_voltage.h"
#import "yocto_temperature.h"

static void usage(void)
```

```
{
  NSLog(@"usage: demo <serial_number> power");
  NSLog(@"       demo <logical_name>  power");
  NSLog(@"       demo any power       (use any discovered device)");
  NSLog(@"       power is a integer between -100 and 100%%");
  NSLog(@"Example:");
  NSLog(@"       demo any 75");
  exit(1);
}

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

  if (argc < 3) {
    usage();
  }

  @autoreleasepool {
    // Setup the API to use local USB devices
    if([YAPI RegisterHub:@"usb": &error] != YAPI_SUCCESS) {
      NSLog(@"RegisterHub error: %@", [error localizedDescription]);
      return 1;
    }
    NSString *target = [NSString stringWithUTF8String:argv[1]];
    int power  = atoi(argv[2]);

    if ([target isEqualToString:@"any"]) {
      // find the serial# of the first available motor
      motor = [YMotor FirstMotor];
      if (motor == nil) {
        NSLog(@"No module connected (check USB cable)");
        return 1;
      }
      target = [[motor get_module] get_serialNumber];
    }

    // retreive motor, current, voltage and temperature features from the device
    YMotor *motor = [YMotor FindMotor:[target stringByAppendingString:@".motor"]];
    YCurrent *current = [YCurrent FindCurrent:[target stringByAppendingString:@".current"]
];
    YVoltage *voltage = [YVoltage FindVoltage:[target stringByAppendingString:@".voltage"]
];
    YTemperature *temperature = [YTemperature
                                 FindTemperature:[target
stringByAppendingString:@".temperature"]];

    // we need to retreive both DC and AC current from the device.
    if ([motor isOnline])  {
      // if motor is in error state, reset it.
      if ([motor get_motorStatus] >= Y_MOTORSTATUS_LOVOLT) {
        [motor resetStatus];
      }
      [motor drivingForceMove:power :2000];  // ramp up to power in 2 seconds
      while ([motor isOnline]) {
        // display motor status
        NSLog(@"Status=%@  Voltage=%fV  Current=%f  Temp=%f",
              [motor get_advertisedValue], [voltage get_currentValue],
              [current get_currentValue] / 1000, [temperature get_currentValue]);
        [YAPI Sleep:1000 :NULL]; // wait for one second
      }
    } else {
      NSLog(@"No module connected (check USB cable)");
      return 1;
    }
    [YAPI FreeAPI];
  }
  return 0;
}
```

20.2. Contrôle de la partie module

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

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

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


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

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

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

Modifications des réglages du module

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

méthode `revertFromFlash`. Ce petit exemple ci-dessous vous permet changer le nom logique d'un module.

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

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


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

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

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

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

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

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

Enumeration des modules

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

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

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

  @autoreleasepool {
```

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

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

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

20.3. 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 `YAPI.DisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur `NomDeClasse.STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `NomDeClasse.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.

21. Utilisation avec des langages non supportés

Les modules Yoctopuce peuvent être contrôlés depuis la plupart des langages de programmation courants. De nouveaux langages sont ajoutés régulièrement en fonction de l'intérêt exprimé par les utilisateurs de produits Yoctopuce. Cependant, certains langages ne sont pas et ne seront jamais supportés par Yoctopuce, les raisons peuvent être diverses: compilateurs plus disponibles, environnements inadaptés, etc...

Il existe cependant des méthodes alternatives pour accéder à des modules Yoctopuce depuis un langage de programmation non supporté.

21.1. Utilisation en ligne de commande

Le moyen le plus simple pour contrôler des modules Yoctopuce depuis un langage non supporté consiste à utiliser l'API en ligne de commande à travers des appels système. L'API en ligne de commande se présente en effet sous la forme d'un ensemble de petits exécutables qu'il est facile d'appeler et dont la sortie est facile à analyser. La plupart des langages de programmation permettant d'effectuer des appels système, cela permet de résoudre le problème en quelques lignes.

Cependant, si l'API en ligne de commande est la solution la plus facile, ce n'est pas la plus rapide ni la plus efficace. A chaque appel, l'exécutable devra initialiser sa propre API et faire l'inventaire des modules USB connectés. Il faut compter environ une seconde par appel.

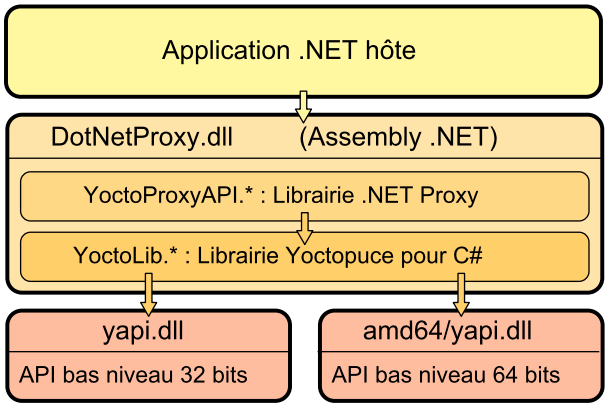
21.2. Assembly .NET

Un Assembly .NET permet de partager un ensemble de classes précompilées pour offrir un service, en annonçant des points d'entrées qui peuvent être utilisés par des applications tierces. Dans notre cas, c'est toute la librairie Yoctopuce qui est disponible dans l'Assembly .NET, de sorte à pouvoir être utilisée dans n'importe quel environnement qui supporte le chargement dynamique d'Assembly .NET.

La librairie Yoctopuce sous forme d'Assembly .NET ne contient pas uniquement la librairie Yoctopuce standard pour C#, car cela n'aurait pas permis une utilisation optimale dans tous les environnements. En effet, on ne peut pas attendre forcément des applications hôtes d'offrir un système de threads ou de callbacks, pourtant très utiles pour la gestion du plug-and-play et des capteurs à taux de rafraîchissements élevé. De même, on ne peut pas attendre des applications externes un comportement transparent dans le cas où un appel de fonction dans l'Assembly cause un délai en raison de communication réseau.

Nous y avons donc ajouté une surcouche, appelée librairie *.NET Proxy*. Cette surcouche offre une interface très similaire à la librairie standard mais un peu simplifiée, car elle gère en interne tous les

mécanismes de callbacks. A la place, cette librairie offre des objets miroirs, appelés *Proxys*, qui publient par le biais de *Propriétés* les principaux attributs des fonctions Yoctopuce tels que la mesure courante, les paramètres de configuration, l'état, etc.



Architecture de l'Assembly .NET

Les propriétés des objets *Proxys* sont automatiquement mises à jour en tâche de fond par le mécanisme de callbacks, sans que l'application hôte n'ait à s'en soucier. Celle-ci peut donc à tout moment et sans aucun risque de latence afficher la valeur de toutes les propriétés des objets Proxys Yoctopuce.

Notez bien que la librairie de communication de bas niveau `yapi.dll` n'est **pas** inclue dans l'Assembly .NET. Il faut donc bien penser à la garder toujours avec `DotNetProxyLibrary.dll`. La version 32 bits doit être dans le même répertoire que `DotNetProxyLibrary.dll`, tandis que la version 64 bits doit être dans un sous-répertoire nommé `amd64`.

Exemple d'utilisation avec MATLAB

Voici comment charger notre Assembly .NET Proxy dans MATLAB et lire la valeur du premier capteur branché par USB trouvé sur la machine :

```
NET.addAssembly("C:/Yoctopuce/DotNetProxyLibrary.dll");
import YoctoProxyAPI.*

errmsg = YAPIProxy.RegisterHub("usb");
sensor = YSensorProxy.FindSensor("");
measure = sprintf('%.3f %s', sensor.CurrentValue, sensor.Unit);
```

Exemple d'utilisation en PowerShell

Les commandes en PowerShell sont un peu plus étranges, mais on reconnaît le même schéma :

```
Add-Type -Path "C:/Yoctopuce/DotNetProxyLibrary.dll"

$errmsg = [YoctoProxyAPI.YAPIProxy]::RegisterHub("usb")
$sensor = [YoctoProxyAPI.YSensorProxy]::FindSensor("")
$measure = "{0:n3} {1}" -f $sensor.CurrentValue, $sensor.Unit
```

Particularités de la librairie .NET Proxy

Par rapport aux librairies Yoctopuce classiques, on notera en particulier les différences suivantes.

Pas de méthode FirstModule/nextModule

Pour obtenir un objet se référant au premier module trouvé, on appelle un `YModuleProxy.FindModule("")`. Si aucun module n'est connecté, cette méthode retournera un objet avec la propriété `module.IsOnline` à `False`. Dès le branchement d'un module, la propriété passera à `True` et l'identifiant matériel du module sera mis à jour.

Pour énumérer les modules, on peut appeler la méthode `module.GetSimilarFunctions()` qui retourne un tableau de chaînes de caractères contenant les identifiants de tous les module trouvés.

Pas de fonctions de callback
Les fonctions de callback sont implémentées en interne et mettent à jour les propriétés des objets. Vous pouvez donc simplement faire du polling sur les propriétés, sans pénalité significative de performance. Prenez garde au fait que si vous utilisez l'une des méthodes qui désactive les callbacks, le rafraichissement automatique des propriétés des objets en sera altéré.

Une nouvelle méthode `YAPIProxy.GetLog` permet de récupérer les logs de diagnostiques de bas niveau sans recourir à l'utilisation de callbacks.

Types énumérés
Pour maximiser la compatibilité avec les applications hôte, la librairie .NET Proxy n'utilise pas de véritables types énumérés .NET, mais des simples entiers. Pour chaque type énuméré, la librairie publie des constantes entières nommées correspondant aux valeurs possibles. Contrairement aux librairies Yoctopuce classiques, les valeurs utiles commencent toujours à `1`, la valeur `0` étant réservée pour signifier une valeur invalide, par exemple lorsque le module est débranché.

Valeurs numériques invalides
Pour toutes les grandeurs numériques, plutôt qu'une constante arbitraire, la valeur invalide retournée en cas d'erreur est *NaN*. Il faut donc utiliser la fonction *isNaN()* pour détecter cette valeur.

Utilisation de l'Assembly .NET sans la librairie Proxy

Si pour une raison ou une autre vous ne désirez pas utiliser la librairie Proxy, et que votre environnement le permet, vous pouvez utiliser l'API C# standard puisqu'elle se trouve dans l'Assembly, sous le namespace `YoctoLib`. Attention toutefois à ne pas mélanger les deux utilisations: soit vous passez par la librairie Proxy, soit vous utilisez directement la version `YoctoLib`, mais pas les deux !

Compatibilité

Pour que la librairie *.NET Proxy* fonctionne correctement avec vos modules Yoctopuce, ces derniers doivent avoir au moins le firmware 37120.

Afin d'être compatible avec un maximum de version de Windows, y compris Windows XP, la librairie *DotNetProxyLibrary.dll* est compilée en .NET 3.5, qui est disponible par défaut sur toutes les versions de Windows depuis XP. A ce jour nous n'avons pas trouvé d'environnement hormis Windows qui supporte le chargement d'Assemblys, donc seules les dll de bas niveau pour Windows sont distribuées avec l'Assembly.

21.3. Virtual Hub et HTTP GET

Le *Virtual Hub* est disponible pour presque toutes les plateformes actuelles, il sert généralement de passerelle pour permettre l'accès aux modules Yoctopuce depuis des langages qui interdisent l'accès direct aux couches matérielles d'un ordinateur (Javascript, PHP, Java...).

Il se trouve que le *Virtual Hub* est en fait un petit serveur Web qui est capable de router des requêtes HTTP vers les modules Yoctopuce. Ce qui signifie que si vous pouvez faire une requête HTTP depuis votre langage de programmation, vous pouvez contrôler des modules Yoctopuce, même si ce langage n'est pas officiellement supporté.

Interface REST

A bas niveau, les modules sont pilotés à l'aide d'une API REST. Ainsi pour contrôler un module, il suffit de faire les requêtes HTTP appropriées sur le *Virtual Hub*. Par défaut le port HTTP du *Virtual Hub* est 4444.

Un des gros avantages de cette technique est que les tests préliminaires sont très faciles à mettre en uvre, il suffit d'un *Virtual Hub* et d'un simple browser Web. Ainsi, si vous copiez l'URL suivante dans votre browser favori, alors que le *Virtual Hub* est en train de tourner, vous obtiendrez la liste des modules présents.

```
http://127.0.0.1:4444/api/services/whitePages.txt
```

Remarquez que le résultat est présenté sous forme texte, mais en demandant *whitePages.xml* vous auriez obtenu le résultat en XML. De même, *whitePages.json* aurait permis d'obtenir le résultat en JSON. L'extension *html* vous permet même d'afficher une interface sommaire vous permettant de changer les valeurs en direct. Toute l'API REST est disponible dans ces différents formats.

Contrôle d'un module par l'interface REST

Chaque module Yoctopuce a sa propre interface REST disponible sous différentes formes. Imaginons un Yocto-Motor-DC avec le numéro de de série *MOTORCTL-12345* et le nom logique *monModule*. l'URL suivante permettra de connaître l'état du module.

```
http://127.0.0.1:4444/bySerial/MOTORCTL-12345/api/module.txt
```

Il est bien entendu possible d'utiliser le nom logique des modules plutôt que leur numéro de série.

```
http://127.0.0.1:4444/byName/monModule/api/module.txt
```

Vous pouvez retrouver la valeur d'une des propriétés d'un module, il suffit d'ajouter le nom de la propriété en dessous de *module*. Par exemple, si vous souhaitez connaître la luminosité des LEDs de signalisation, il vous suffit de faire la requête suivante:

```
http://127.0.0.1:4444/bySerial/MOTORCTL-12345/api/module/luminosity
```

Pour modifier la valeur d'une propriété, il vous suffit de modifier l'attribut correspondant. Ainsi, pour modifier la luminosité il vous suffit de faire la requête suivante:

```
http://127.0.0.1:4444/bySerial/MOTORCTL-12345/api/module?luminosity=100
```

Contrôle des différentes fonctions du module par l'interface REST

Les fonctionnalités des modules se manipulent de la même manière. Pour connaître l'état de la fonction motor, il suffit de construire l'URL suivante.

```
http://127.0.0.1:4444/bySerial/MOTORCTL-12345/api/motor.txt
```

En revanche, si vous pouvez utiliser le nom logique du module en lieu et place de son numéro de série, vous ne pouvez pas utiliser les noms logiques des fonctions, seuls les noms hardware sont autorisés pour les fonctions.

Vous pouvez retrouver un attribut d'une fonction d'un module d'une manière assez similaire à celle utilisée avec les modules, par exemple:

```
http://127.0.0.1:4444/bySerial/MOTORCTL-12345/api/motor/logicalName
```

Assez logiquement, les attributs peuvent être modifiés de la même manière.

```
http://127.0.0.1:4444/bySerial/MOTORCTL-12345/api/motor?logicalName=maFonction
```

Vous trouverez la liste des attributs disponibles pour votre Yocto-Motor-DC au début du chapitre *Programmation, concepts généraux*.

Accès aux données enregistrées sur le datalogger par l'interface REST

Cette section s'applique uniquement aux modules dotés d'un enregistreur de donnée.

La version résumée des données enregistrées dans le datalogger peut être obtenue au format JSON à l'aide de l'URL suivante:

```
http://127.0.0.1:4444/bySerial/MOTORCTL-12345/dataLogger.json
```

Le détail de chaque mesure pour un chaque tranche d'enregistrement peut être obtenu en ajoutant à l'URL l'identifiant de la fonction désirée et l'heure de départ de la tranche:

```
http://127.0.0.1:4444/bySerial/MOTORCTL-12345/dataLogger.json?id=motor&utc=1389801080
```

21.4. Utilisation des librairies dynamiques

L'API Yoctopuce bas niveau est disponible sous différents formats de librairie dynamiques écrites en C, dont les sources sont disponibles avec l'API C++. Utiliser une de ces librairies bas niveau vous permettra de vous passer du *Virtual Hub*.

Filename	Plateforme
libyapi.dylib	Max OS X
libyapi-amd64.so	Linux Intel (64 bits)
libyapi-armel.so	Linux ARM EL (32 bits)
libyapi-armhf.so	Linux ARM HL (32 bits)
libyapi-aarch64.so	Linux ARM (64 bits)
libyapi-i386.so	Linux Intel (32 bits)
yapi64.dll	Windows (64 bits)
yapi.dll	Windows (32 bits)

Ces librairies dynamiques contiennent toutes les fonctionnalités nécessaires pour reconstruire entièrement toute l'API haut niveau dans n'importe quel langage capable d'intégrer ces librairies. Ce chapitre se limite cependant à décrire une utilisation de base des modules.

Contrôle d'un module

Les trois fonctions essentielles de l'API bas niveau sont les suivantes:

```
int yapiInitAPI(int connection_type, char *errmsg);
int yapiUpdateDeviceList(int forceupdate, char *errmsg);
int yapiHTTPRequest(char *device, char *request, char* buffer,int buffsize,int *fullsize,
char *errmsg);
```

La fonction *yapiInitAPI* permet d'initialiser l'API et doit être appelée une fois en début du programme. Pour une connection de type USB, le paramètre *connection_type* doit prendre la valeur 1. *errmsg* est un pointeur sur un buffer de 255 caractères destiné à récupérer un éventuel message d'erreur. Ce pointeur peut être aussi mis à *NULL*. La fonction retourne un entier négatif en cas d'erreur, ou zéro dans le cas contraire.

La fonction *yapiUpdateDeviceList* gère l'inventaire des modules Yoctopuce connectés, elle doit être appelée au moins une fois. Pour pouvoir gérer le hot plug, et détecter d'éventuels nouveaux modules connectés, cette fonction devra être apellée à intervalles réguliers. Le paramètre *forceupdate* devra être à la valeur 1 pour forcer un scan matériel. Le paramètre *errmsg* devra pointer sur un buffer de 255 caractères pour récupérer un éventuel message d'erreur. Ce pointeur peut aussi être à *null*.Cette fonction retourne un entier négatif en cas d'erreur, ou zéro dans le cas contraire.

Enfin, la fonction *yapiHTTPRequest* permet d'envoyer des requêtes HTTP à l'API REST du module. Le paramètre *device* devra contenir le numéro de série ou le nom logique du module que vous cherchez à atteindre. Le paramètre *request* doit contenir la requête HTTP complète (y compris les sauts de ligne terminaux). *buffer* doit pointer sur un buffer de caractères suffisamment grand pour contenir la réponse. *buffsize* doit contenir la taille du buffer. *fullsize* est un pointeur sur un entier qui

sera affecté à la taille effective de la réponse. Le paramètre *errmsg* devra pointer sur un buffer de 255 caractères pour récupérer un éventuel message d'erreur. Ce pointeur peut aussi être à *null*. Cette fonction retourne un entier négatif en cas d'erreur, ou zéro dans le cas contraire.

Le format des requêtes est le même que celui décrit dans la section *Virtual Hub et HTTP GET*. Toutes les chaînes de caractères utilisées par l'API sont des chaînes constituées de caractères 8 bits: l'Unicode et l'UTF8 ne sont pas supportés.

Le résultat retourné dans la variable buffer respecte le protocole HTTP, il inclut donc un header HTTP . Ce header se termine par deux lignes vides, c'est-à-dire une séquence de quatre caractères ASCII 13, 10, 13, 10.

Voici un programme d'exemple écrit en pascal qui utilise la DLL *yapi.dll* pour lire puis changer la luminosité d'un module.

```
// Dll functions import
function  yapiInitAPI(mode:integer;
                      errmsg : pansichar):integer;cdecl;
                      external 'yapi.dll' name 'yapiInitAPI';
function  yapiUpdateDeviceList(force:integer;errmsg : pansichar):integer;cdecl;
                      external 'yapi.dll' name 'yapiUpdateDeviceList';
function  yapiHTTPRequest(device:pansichar;url:pansichar; buffer:pansichar;
                      buffsize:integer;var fullsize:integer;
                      errmsg : pansichar):integer;cdecl;
                      external 'yapi.dll' name 'yapiHTTPRequest';

var
 errmsgBuffer  : array [0..256] of ansichar;
 dataBuffer    : array [0..1024] of ansichar;
 errmsg,data   : pansichar;
 fullsize,p    : integer;

const
  serial      = 'MOTORCTL-12345';
  getValue = 'GET /api/module/luminosity HTTP/1.1'#13#10#13#10;
  setValue = 'GET /api/module?luminosity=100 HTTP/1.1'#13#10#13#10;

begin
  errmsg  :=  @errmsgBuffer;
  data    :=  @dataBuffer;
  // API  initialization
  if(yapiInitAPI(1,errmsg)<0) then
   begin
    writeln(errmsg);
    halt;
  end;

  // forces a device inventory
  if( yapiUpdateDeviceList(1,errmsg)<0) then
    begin
     writeln(errmsg);
     halt;
   end;

  // requests the  module luminosity
  if (yapiHTTPRequest(serial,getValue,data,sizeof(dataBuffer),fullsize,errmsg)<0) then
   begin
     writeln(errmsg);
     halt;
   end;

  // searches for the HTTP header end
  p := pos(#13#10#13#10,data);

  // displays the response minus the HTTP header
  writeln(copy(data,p+4,length(data)-p-3));

  // change the luminosity
  if (yapiHTTPRequest(serial,setValue,data,sizeof(dataBuffer),fullsize,errmsg)<0) then
   begin
     writeln(errmsg);
     halt;
   end;
```

```
end.
```

Inventaire des modules

Pour procéder à l'inventaire des modules Yoctopuce, deux fonctions de la librairie dynamique sont nécessaires

```
int yapiGetAllDevices(int *buffer,int maxsize,int *neededsize,char *errmsg);
int yapiGetDeviceInfo(int devdesc,yDeviceSt *infos, char *errmsg);
```

La fonction *yapiGetAllDevices* permet d'obtenir la liste des modules connectés sous la forme d'une liste de handles. *buffer* pointe sur un tableau d'entiers 32 bits qui contiendra les handles retournés. *Maxsize* est la taille en bytes du buffer. *neededsize* contiendra au retour la taille nécessaire pour stocker tous les handles. Cela permet d'en déduire le nombre de module connectés, ou si le buffer passé en entrée est trop petit. Le paramètre *errmsg* devra pointer sur un buffer de 255 caractères pour récupérer un éventuel message d'erreur. Ce pointeur peut aussi être à *null*. Cette fonction retourne un entier négatif en cas d'erreur, ou zéro dans le cas contraire.

La fonction *yapiGetDeviceInfo* permet de récupérer les informations relatives à un module à partir de son handle. *devdesc* est un entier 32bit qui représente le module, et qui a été obtenu grâce à *yapiGetAllDevices*. *infos* pointe sur une structure de données dans laquelle sera stocké le résultat. Le format de cette structure est le suivant:

Nom	Type	Taille (bytes)	Description
vendorid	int	4	ID USB de Yoctopuce
deviceid	int	4	ID USB du module
devrelease	int	4	Version du module
nbinbterfaces	int	4	Nombre d'interfaces USB utilisée par le module
manufacturer	char[]	20	Yoctopuce (null terminé)
productname	char[]	28	Modèle (null terminé)
serial	char[]	20	Numéro de série (null terminé)
logicalname	char[]	20	Nom logique (null terminé)
firmware	char[]	22	Version du firmware (null terminé)
beacon	byte	1	Etat de la balise de localisation (0/1)

Le paramètre *errmsg* devra pointer sur un buffer de 255 caractères pour récupérer un éventuel message d'erreur.

Voici un programme d'exemple écrit en pascal qui utilise la DLL *yapi.dll* pour lister les modules connectés.

```
// device description structure
type yDeviceSt = packed record
   vendorid        : word;
   deviceid        : word;
   devrelease      : word;
   nbinbterfaces   : word;
   manufacturer    : array [0..19] of ansichar;
   productname     : array [0..27] of ansichar;
   serial          : array [0..19] of ansichar;
   logicalname     : array [0..19] of ansichar;
   firmware        : array [0..21] of ansichar;
   beacon          : byte;
 end;

// Dll function import
function  yapiInitAPI(mode:integer;
                      errmsg : pansichar):integer;cdecl;
                      external 'yapi.dll' name 'yapiInitAPI';

function  yapiUpdateDeviceList(force:integer;errmsg : pansichar):integer;cdecl;
                      external 'yapi.dll' name 'yapiUpdateDeviceList';

function  yapiGetAllDevices( buffer:pointer;
                             maxsize:integer;
```

```
                              var neededsize:integer;
                              errmsg : pansichar):integer; cdecl;
                              external 'yapi.dll' name 'yapiGetAllDevices';

function  apiGetDeviceInfo(d:integer; var infos:yDeviceSt;
                              errmsg : pansichar):integer;  cdecl;
                              external 'yapi.dll' name 'yapiGetDeviceInfo';


var
 errmsgBuffer  : array [0..256] of ansichar;
 dataBuffer    : array [0..127] of integer;   // max of 128 USB devices
 errmsg,data   : pansichar;
 neededsize,i  : integer;
 devinfos      : yDeviceSt;

begin
  errmsg  :=  @errmsgBuffer;

  // API  initialisation
  if(yapiInitAPI(1,errmsg)<0) then
   begin
    writeln(errmsg);
    halt;
  end;

   // forces a device inventory
  if( yapiUpdateDeviceList(1,errmsg)<0) then
    begin
     writeln(errmsg);
     halt;
   end;

  // loads all device handles into dataBuffer
  if yapiGetAllDevices(@dataBuffer,sizeof(dataBuffer),neededsize,errmsg)<0 then
    begin
     writeln(errmsg);
     halt;
    end;

  // gets device info from each handle
  for i:=0 to  neededsize div sizeof(integer)-1 do
   begin
     if (apiGetDeviceInfo(dataBuffer[i], devinfos, errmsg)<0) then
       begin
         writeln(errmsg);
         halt;
       end;
     writeln(pansichar(@devinfos.serial)+' ('+pansichar(@devinfos.productname)+')');
   end;

end.
```

VB6 et yapi.dll

Chaque point d'entrée de la DLL yapi.dll est disponible en deux versions, une classique C-decl, et un seconde compatible avec Visual Basic 6 préfixée avec *vb6_*.

21.5. Port de la librairie haut niveau

Toutes les sources de l'API Yoctopuce étant fournies dans leur intégralité, vous pouvez parfaitement entreprendre le port complet de l'API dans le langage de votre choix. Sachez cependant qu'une grande partie du code source de l'API est généré automatiquement.

Ainsi, il n'est pas nécessaire de porter la totalité de l'API, il suffit de porter le fichier *yocto_api* et un de ceux correspondant à une fonctionnalité, par exemple *yocto_relay*. Moyennant un peu de travail supplémentaire, Yoctopuce sera alors en mesure de générer tous les autres fichiers. C'est pourquoi il est fortement recommandé de contacter le support Yoctopuce avant d'entreprendre le port de la librairie Yoctopuce dans un autre langage. Un travail collaboratif sera profitable aux deux parties.

22. Programmation avancée

Les chapitres précédents vous ont présenté dans chaque language disponible les fonctions de programmation de base utilisables avec votre module Yocto-Motor-DC. Ce chapitre présente de façon plus générale une utilisation plus avancée de votre module. Les exemples sont donnés dans le language le plus populaire auprès des clients de Yoctopuce, à savoir C#. Néanmoins, vous trouverez dans les librairies de programmation pour chaque language des exemples complets illustrant les concepts présentés ici.

Afin de rester le plus concis possible, les exemples donnés dans ce chapitre ne font aucune gestion d'erreur. Ne les copiez pas tels-quels dans une application de production.

22.1. Programmation par événements

Les méthodes de gestion des modules Yoctopuce qui vous ont été présentées dans les chapitres précédents sont des fonctions de polling, qui consistent à demander en permanence à l'API si quelque chose a changé. Facile à appréhender, cette technique de programmation est n'est pas la plus efficace ni la plus réactive. C'est pourquoi l'API de programmation Yoctopuce propose aussi un modèle de programmation par événements. Cette technique consiste à demander à l'API de signaler elle-même les changements importants dès qu'ils sont détectés. A chaque fois qu'un paramètre clé change, l'API appelle une fonction de callback que vous avez prédéfinie.

Détecter l'arrivée et le départ des modules

La gestion du *hot-plug* est importante lorsque l'on travaille avec des modules USB, car tôt ou tard vous serez amené à brancher et débrancher un module après le lancement de votre programme. L'API a été conçue pour gérer l'arrivée et le départ inopinés des modules de manière transparente, mais votre application doit en général en tenir compte si elle veut éviter de prétendre utiliser un module qui a été débranché.

La programmation par événements est particulièrement utile pour détecter les branchements/débranchements de modules. Il est en effet plus simple de se faire signaler les branchements, que de devoir lister en permanence les modules branchés pour en déduire ceux qui sont arrivés et ceux qui sont partis. Pour pouvoir être prévenu dès qu'un module arrive, vous avez besoin de trois morceaux de code.

Le callback

Le callback est la fonction qui sera appelée à chaque fois qu'un nouveau module Yoctopuce sera branché. Elle prend en paramètre le module concerné.

```
static void deviceArrival(YModule m)
```

```
{
    Console.WriteLine("Nouveau module  : " + m.get_serialNumber());
}
```

L'initialisation
Vous devez ensuite signaler à l'API qu'il faut appeler votre callback quand un nouveau module est branché.

```
YAPI.RegisterDeviceArrivalCallback(deviceArrival);
```

Notez que si des modules sont déjà branchés lorsque le callback est enregistré, le callback sera appelé pour chacun de ces modules déjà branchés.

Déclenchement des callbacks
Un problème classique de la programmation par callbacks est que ces callbacks peuvent être appelés n'importe quand, y compris à des moments où le programme principal n'est pas prêt à les recevoir, ce qui peut avoir des effets de bords indésirables comme des dead-locks et autres conditions de course. C'est pourquoi dans l'API Yoctopuce, les callbacks d'arrivée/départs de modules ne sont appelés que pendant l'exécution de la fonction `UpdateDeviceList()`. Il vous suffit d'appeler `UpdateDeviceList()` à intervalle régulier depuis un timer ou un thread spécifique pour controller précisément quand les appels à ces callbacks auront lieu:

```
// boucle d'attente gérant les callback
while (true)
{
    // callback d'arrivée / départ de modules
    YAPI.UpdateDeviceList(ref errmsg);
    // attente non active gérant les autres callbacks
    YAPI.Sleep(500, ref errmsg);
}
```

De manière similaire, il est possible d'avoir un callback quand un module est débranché. Vous trouverez un exemple concret démontrant toutes ces techniques dans la librairie de programmation Yoctopuce de chaque langage. L'exemple se trouve dans le répertoire *Examples/Prog-EventBased*.

Attention: dans la plupart des langages, les callbacks doivent être des procédures globales, et non pas des méthodes. Si vous souhaitez que le callback appelle une méthode d'un objet, définissez votre callback sous la forme d'une procédure globale qui ensuite appellera votre méthode.

23. Mise à jour du firmware

Il existe plusieurs moyens de mettre à jour le firmware des modules Yoctopuce.

23.1. Le VirtualHub ou le YoctoHub

Il est possible de mettre à jour un module directement depuis l'interface web du VirutalHub ou du YoctoHub. Il suffit d'accéder à la fenêtre de configuration du module que à mettre à jour et de cliquer sur le bouton "upgrade". Le VirtualHub démarre un assistant qui vous guidera durant la procédure de mise à jour.

Si pour une raison ou une autre, la mise à jour venait à échouer et que le module de fonctionnait plus, débranchez puis rebranchez le module en maintenant sur le Yocto-bouton appuyé. Le module va démarrer en mode "mise à jour" et sera listé en dessous des modules connectés.

23.2. La librairie ligne de commandes

Tous les outils en lignes de commandes ont la possibilité de mettre à jour les modules Yoctopuce grâce à la commande `downloadAndUpdate`. Le mécanisme de sélection des modules fonctionne comme pour une commande traditionnelle. La [cible] est le nom du module qui va être mis à jour. Vous pouvez aussi utiliser les alias "any" ou "all", ou encore une liste de noms, séparés par des virgules, sans espace.

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

L'exemple suivant met à jour tous les modules Yoctopuce connectés en USB.

```
C:\>YModule all downloadAndUpdate
ok: Yocto-PowerRelay RELAYHI1-266C8(rev=15430) is up to date.
ok: 0 / 0 hubs in 0.000000s.
ok: 0 / 0 shields in 0.000000s.
ok: 1 / 1 devices in 0.130000s 0.130000s per device.
ok: All devices are now up to date.
C:\>
```

23.3. L'application Android Yocto-Firmware

Il est possible de mettre à jour le firmware de vos modules depuis votre téléphone ou tablette Android avec l'application Yocto-Firmware. Cette application liste tous les modules Yoctopuce

branchés en USB et vérifie si un firmware plus récent est disponible sur www.yoctopuce.com. Si un firmware plus récent est disponible, il est possible de mettre à jour le module. L'application se charge de télécharger et d'installer le nouveau firmware en préservant les paramètres du module.

Attention, pendant la mise à jour du firmware, le module redémarre plusieurs fois. Android interprète le reboot d'un périphérique USB comme une déconnexion et reconnexion du périphérique USB, et demande à nouveau l'autorisation d'utiliser le port USB. L'utilisateur est obligé de cliquer sur **OK** pour que la procédure de mise à jour se termine correctement.

23.4. La librairie de programmation

Si vous avez besoin d'intégrer la mise à jour de firmware dans votre application, les librairies proposent une API pour mettre à jour vos modules.[1]

Sauvegarder et restaurer les paramètres

La méthode `get_allSettings()` retourne un buffer binaire qui permet de sauvegarder les paramètres persistants d'un module. Cette fonction est très utile pour sauvegarder la configuration réseau d'un YoctoHub par exemple.

```
YWireless wireless = YWireless.FindWireless("reference");
YModule m = wireless.get_module();
byte[] default_config =  m.get_allSettings();
saveFile("default.bin", default_config);
...
```

Ces paramètres peuvent être appliqués sur d'autres modules à l'aide de la méthode `set_allSettings()`.

```
byte[] default_config = loadFile("default.bin");
YModule m = YModule.FirstModule();
while (m != null) {
  if (m.get_productName() == "YoctoHub-Wireless") {
    m.set_allSettings(default_config);
  }
  m = m.next();
}
```

Chercher le bon firmware

La première étape pour mettre à jour un module Yoctopuce est de trouver quel firmware il faut utiliser, c'est le travail de la méthode `checkFirmware(path, onlynew)` de l'objet YModule. Cette méthode vérifie que le firmware passé en argument (`path`) est compatible avec le module. Si le paramètre `onlynew` est vrai, cette méthode vérifie si le firmware est plus récent que la version qui est actuellement utilisée par le module. Quand le fichier n'est pas compatible (ou si le fichier est plus vieux que la version installée), cette méthode retourne une chaîne vide. Si au contraire le fichier est valide, la méthode retourne le chemin d'accès d'un fichier.

Le code suivant vérifie si le fichier `c:\tmp\METEOMK1.17328.byn` est compatible avec le module stocké dans la variable *m*.

```
YModule m = YModule.FirstModule();
...
...
string path = "c:\\tmp\METEOMK1.17328.byn";
string newfirm = m.checkFirmware(path, false);
if (newfirm != "") {
  Console.WriteLine("firmware " + newfirm + " is compatible");
}
...
```

[1] Les librairies JavaScript, Node.js et PHP ne permettent pas encore de mettre à jour les modules, mais ces fonctions seront disponibles dans un prochain build.

Il est possible de passer un répertoire en argument (au lieu d'un fichier). Dans ce cas la méthode va parcourir récursivement tous les fichiers du répertoire et retourner le firmware compatible le plus récent. Le code suivant vérifie s'il existe un firmware plus récent dans le répertoire `c:\tmp\`.

```
YModule m = YModule.FirstModule();
...
...
string path = "c:\\tmp";
string newfirm = m.checkFirmware(path, true);
if (newfirm != "") {
  Console.WriteLine("firmware " + newfirm + " is compatible and newer");
}
...
```

Il est aussi possible de passer la chaîne "www.yoctopuce.com" en argument pour vérifier s'il existe un firmware plus récent publié sur le site web de Yoctopuce. Dans ce cas, la méthode retournera l'URL du firmware. Vous pourrez soit utiliser cette URL pour télécharger le firmware sur votre disque, soit utiliser cette URL lors de la mise à jour du firmware (voir ci-dessous). Bien évidemment, cette possibilité ne fonctionne que si votre machine est reliée à Internet.

```
YModule m = YModule.FirstModule();
...
...
string url = m.checkFirmware("www.yoctopuce.com", true);
if (url != "") {
  Console.WriteLine("new firmware is available at " + url );
}
...
```

Mettre à jour le firmware

La mise à jour du firmware peut prendre plusieurs minutes, c'est pourquoi le processus de mise à jour est exécuté par la librairie en arrière plan et est contrôlé par le code utilisateur à l'aide de la classe `YFirmwareUdpate`.

Pour mettre à jour un module Yoctopuce, il faut obtenir une instance de la classe `YFirmwareUpdate` à l'aide de la méthode `updateFirmware` d'un objet YModule. Le seul paramètre de cette méthode est le *path* du firmware à installer. Cette méthode ne démarre pas immédiatement la mise à jour, mais retourne un objet `YFirmwareUpdate` configuré pour mettre à jour le module.

```
string newfirm = m.checkFirmware("www.yoctopuce.com", true);
.....
YFirmwareUpdate fw_update = m.updateFirmware(newfirm);
```

La méthode `startUpdate()` démarre la mise à jour en arrière plan. Ce processus en arrière plan se charge automatiquement de:

1. sauvegarder des paramètres du module,
2. redémarrer le module en mode "mise à jour"
3. mettre à jour le firmware
4. démarrer le module avec la nouvelle version du firmware
5. restaurer les paramètres

Les méthodes `get_progress()` et `get_progressMessage()` permettent de suivre la progression de la mise à jour. `get_progress()` retourne la progression sous forme de pourcentage (100 = mise à jour terminée). `get_progressMessage()` retourne une chaîne de caractères décrivant l'opération en cours (effacement, écriture, reboot,...). Si la méthode `get_progress()` retourne une valeur négative, c'est que le processus de mise à jour à échoué. Dans ce cas la méthode `get_progressMessage()` retourne le message d'erreur.

Le code suivant démarre la mise à jour et affiche la progression sur la sortie standard.

```
YFirmwareUpdate fw_update = m.updateFirmware(newfirm);
....
int status = fw_update.startUpdate();
while (status < 100 && status >= 0) {
  int newstatus = fw_update.get_progress();
  if (newstatus != status) {
    Console.WriteLine(status + "% "
      + fw_update.get_progressMessage());
  }
  YAPI.Sleep(500, ref errmsg);
  status = newstatus;
}

if (status < 0) {
  Console.WriteLine("Firmware Update failed: "
    + fw_update.get_progressMessage());
} else {
  Console.WriteLine("Firmware Updated Successfully!");
}
```

Particularité d'Android

Il est possible de mettre à jour un firmware d'un module en utilisant la librairie Android. Mais pour les modules branchés en USB, Android va demander à l'utilisateur d'autoriser l'application à accéder au port USB.

Pendant la mise à jour du firmware, le module redémarre plusieurs fois. Android interprète le reboot d'un périphérique USB comme une déconnexion et reconnexion du port USB, et interdit tout accès USB tant que l'utilisateur n'a pas fermé le pop-up. L'utilisateur est obligé de cliquer sur *OK* pour que la procédure de mise à jour puisse continuer correctement. **Il n'est pas possible de mettre à jour un module branché en USB à un appareil Android sans que l'utilisateur ne soit obligé d'interagir avec l'appareil**.

23.5. Le mode "mise à jour"

Si vous désirez effacer tous les paramètres du module ou que votre module ne démarre plus correctement, il est possible d'installer un firmware depuis le mode "mise à jour".

Pour forcer le module à fonctionner dans le mode "mis à jour", débranchez-le, attendez quelques secondes, et rebranchez-le en maintenant le *Yocto-Bouton* appuyé. Cela a pour effet de faire démarrer le module en mode "mise à jour". Ce mode de fonctionnement est protégé contre les corruptions et est toujours accessible.

Dans ce mode, le module n'est plus détecté par les objets YModules. Pour obtenir la liste des modules connectés en mode "mise à jour", il faut utiliser la fonction `YAPI.GetAllBootLoaders()`. Cette fonction retourne un tableau de chaînes de caractères avec le numéro de série des modules en le mode "mise à jour".

```
List<string> allBootLoader = YAPI.GetAllBootLoaders();
```

La procédure de mise à jour est identique au cas standard (voir section précédente), mais il faut instancier manuellement l'objet `YFirmwareUpdate` au lieu d'appeler `module.updateFirmware()`. Le constructeur prend en argument trois paramètres: le numéro de série du module, le path du firmware à installer, et un tableau de bytes avec les paramètres à restaurer à la fin de la mise à jour (ou `null` pour restaurer les paramètres d'origine).

```
YFirmwareUpdateupdate fw_update;
fw_update = new YFirmwareUpdate(allBootLoader[0], newfirm, null);
int status = fw_update.startUpdate();
.....
```

24. Référence de l'API de haut niveau

Ce chapitre résume les fonctions de l'API de haut niveau pour commander votre Yocto-Motor-DC. La syntaxe et les types précis peuvent varier d'un langage à l'autre mais, sauf avis contraire toutes sont disponibles dans chaque language. Pour une information plus précise sur les types des arguments et des valeurs de retour dans un langage donné, veuillez vous référer au fichier de définition pour ce langage (`yocto_api.*` ainsi que les autres fichiers `yocto_*` définissant les interfaces des fonctions).

Dans les langages qui supportent les exceptions, toutes ces fonctions vont par défaut générer des exceptions en cas d'erreur plutôt que de retourner la valeur d'erreur documentée pour chaque fonction, afin de faciliter le déboguage. Il est toutefois possible de désactiver l'utilisation d'exceptions à l'aide de la fonction `yDisableExceptions()`, si l'on préfère travailler avec des valeurs de retour d'erreur.

Ce chapitre ne reprend pas en détail les concepts de programmation décrits plus tôt, afin d'offrir une référence plus concise. En cas de doute, n'hésitez pas à retourner au chapitre décrivant en détail de chaque attribut configurable.

24.1. La classe YAPI

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:

java	import com.yoctopuce.YoctoAPI.YAPI;
dnp	import YoctoProxyAPI.YAPIProxy
cp	#include "yocto_api_proxy.h"
ml	import YoctoProxyAPI.YAPIProxy"
js	<script type='text/javascript' src='yocto_api.js'></script>
cpp	#include "yocto_api.h"
m	#import "yocto_api.h"
pas	uses yocto_api;
vb	yocto_api.vb
cs	yocto_api.cs
uwp	import com.yoctopuce.YoctoAPI.YModule;
py	from yocto_api import *
php	require_once('yocto_api.php');
ts	in HTML: import { YAPI, YErrorMsg, YModule, YSensor } from '../../dist/esm/yocto_api_browser.js'; in Node.js: import { YAPI, YErrorMsg, YModule, YSensor } from 'yoctolib-cjs/yocto_api_nodejs.js';
es	in HTML: <script src="../../lib/yocto_api.js"></script> in node.js: require('yoctolib-es2017/yocto_api.js');
vi	YModule.vi

Fonction globales
YAPI.CheckLogicalName(**name**) Vérifie si un nom donné est valide comme nom logique pour un module ou une fonction.
YAPI.ClearHTTPCallbackCacheDir(**bool_removeFiles**) Désactive le cache de callback HTTP.
YAPI.DisableExceptions() Désactive l'utilisation d'exceptions pour la gestion des erreurs.
YAPI.EnableExceptions() Réactive l'utilisation d'exceptions pour la gestion des erreurs.
YAPI.EnableUSBHost(**osContext**) Cette fonction est utilisée uniquement sous Android.
YAPI.FreeAPI() Attends que toutes les communications en cours avec les modules Yoctopuce soient terminées puis libère les ressources utilisées par la librairie Yoctopuce.
YAPI.GetAPIVersion() Retourne la version de la librairie Yoctopuce utilisée.
YAPI.GetCacheValidity() Retourne la durée de validité des données chargée par la libraire.
YAPI.GetDeviceListValidity()

Retourne le délai entre chaque énumération forcée des YoctoHubs utilisés.

YAPI.GetDllArchitecture()

Retourne l'architecture du binaire de la librairie de communication Yoctopuce utilisée.

YAPI.GetDllPath()

Retourne l'emplacement sur le disque des librairies Yoctopuce utilisées.

YAPI.GetLog(lastLogLine)

Récupère les messages de logs de la librairie de communication Yoctopuce.

YAPI.GetNetworkTimeout()

Retourne le délai de connexion réseau pour `yRegisterHub()` et `yUpdateDeviceList()`.

YAPI.GetTickCount()

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

YAPI.HandleEvents(errmsg)

Maintient la communication de la librairie avec les modules Yoctopuce.

YAPI.InitAPI(mode, errmsg)

Initialise la librairie de programmation de Yoctopuce explicitement.

YAPI.PreregisterHub(url, errmsg)

Alternative plus tolerante à `yRegisterHub()`.

YAPI.RegisterDeviceArrivalCallback(arrivalCallback)

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

YAPI.RegisterDeviceRemovalCallback(removalCallback)

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

YAPI.RegisterHub(url, errmsg)

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

YAPI.RegisterHubDiscoveryCallback(hubDiscoveryCallback)

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

YAPI.RegisterHubWebsocketCallback(ws, errmsg, authpwd)

Variante de la fonction `yRegisterHub()` destinée à initialiser l'API Yoctopuce sur une session Websocket existante, dans le cadre d'un callback WebSocket entrant.

YAPI.RegisterLogFunction(logfun)

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

YAPI.SelectArchitecture(arch)

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

YAPI.SetCacheValidity(cacheValidityMs)

Change la durée de validité des données chargées par la librairie.

YAPI.SetDelegate(object)

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

YAPI.SetDeviceListValidity(deviceListValidity)

Change le délai entre chaque énumération forcée des YoctoHub utilisés.

YAPI.SetHTTPCallbackCacheDir(str_directory)

Active le cache du callback HTTP.

YAPI.SetNetworkTimeout(networkMsTimeout)

Change le délai de connexion réseau pour `yRegisterHub()` et `yUpdateDeviceList()`.

YAPI.SetTimeout(callback, ms_timeout, args)

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

YAPI.SetUSBPacketAckMs(pktAckDelay)

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

YAPI.Sleep(ms_duration, errmsg)

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

YAPI.TestHub(url, mstimeout, errmsg)

Test si un hub est joignable.

YAPI.TriggerHubDiscovery(errmsg)

Relance une détection des hubs réseau.

YAPI.UnregisterHub(url)

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

YAPI.UpdateDeviceList(errmsg)

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

YAPI.UpdateDeviceList_async(callback, context)

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

YAPI.CheckLogicalName() YAPI

YAPI.CheckLogicalName()

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

```
js    function yCheckLogicalName( name)
cpp   bool CheckLogicalName( string name)
m     +(BOOL) CheckLogicalName :(NSString *) name
pas   boolean yCheckLogicalName( name: string): boolean
vb    function CheckLogicalName( ByVal name As String) As Boolean
cs    static bool CheckLogicalName( string name)
java  boolean CheckLogicalName( String name)
uwp   bool CheckLogicalName( string name)
py    CheckLogicalName( name)
php   function CheckLogicalName( $name)
ts    async CheckLogicalName( name: string): Promise<boolean>
es    async CheckLogicalName( name)
```

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

Paramètres :

name une chaîne de caractères contenant le nom vérifier.

Retourne :

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

YAPI.ClearHTTPCallbackCacheDir() YAPI

YAPI.ClearHTTPCallbackCacheDir()

Désactive le cache de callback HTTP.

```
php function ClearHTTPCallbackCacheDir( $bool_removeFiles)
```

Cette méthode désctive la cache de callback HTTP, et permet également d'en effacer le contenu.

Paramètres :

bool_removeFiles Vrai pour effacer le contenu du répertoire de cache.

Retourne :

nothing.

YAPI.DisableExceptions() YAPI

YAPI.DisableExceptions()

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

js	function **yDisableExceptions**( )
cpp	void **DisableExceptions**( )
m	+(void) **DisableExceptions**
pas	**yDisableExceptions**( )
vb	procedure **DisableExceptions**( )
cs	static void **DisableExceptions**( )
uwp	void **DisableExceptions**( )
py	**DisableExceptions**( )
php	function **DisableExceptions**( )
ts	async **DisableExceptions**( ): Promise<void>
es	async **DisableExceptions**( )

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

YAPI.EnableExceptions() YAPI

YAPI.EnableExceptions()

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

js	function **yEnableExceptions**( )
cpp	void **EnableExceptions**( )
m	+(void) **EnableExceptions**
pas	**yEnableExceptions**( )
vb	procedure **EnableExceptions**( )
cs	static void **EnableExceptions**( )
uwp	void **EnableExceptions**( )
py	**EnableExceptions**( )
php	function **EnableExceptions**( )
ts	async **EnableExceptions**( ): Promise<void>
es	async **EnableExceptions**( )

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

YAPI.EnableUSBHost() YAPI

YAPI.EnableUSBHost()

Cette fonction est utilisée uniquement sous Android.

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

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

Paramètres :

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

YAPI.FreeAPI() YAPI

YAPI.FreeAPI()

Attends que toutes les communications en cours avec les modules Yoctopuce soient terminées puis libère les ressources utilisées par la librairie Yoctopuce.

js	function **yFreeAPI**( )
cpp	void **FreeAPI**( )
m	+(void) **FreeAPI**
pas	**yFreeAPI**( )
vb	procedure **FreeAPI**( )
cs	static void **FreeAPI**( )
java	void **FreeAPI**( )
uwp	void **FreeAPI**( )
py	**FreeAPI**( )
php	function **FreeAPI**( )
ts	async **FreeAPI**( ): Promise<void>
es	async **FreeAPI**( )
dnp	static void **FreeAPI**( )
cp	static void **FreeAPI**( )

Du point de vue du système d'exploitation, il n'est en général pas nécessaire d'appeler cette fonction puisque que l'OS libèrera automatiquement les ressources allouées dès que le programme sera terminé. Cependant il existe deux situations où vous auriez un intérêt à l'appeler: - Vous désirez libérer tous les blocs de mémoire alloués dynamiquement dans le but d'identifier une source de blocs perdus par exemple. - Votre code envoie des commandes aux modules juste avant de se terminer. Les commandes étant envoyées de manière asynchrone, sans cet appel le programme risquerait de se terminer avant que toutes les commandes n'aient eu le temps de partir. Vous ne devez plus appeler aucune fonction de la librairie après avoir appelé `yFreeAPI()`, sous peine de crash.

YAPI.GetAPIVersion() YAPI

YAPI.GetAPIVersion()

Retourne la version de la librairie Yoctopuce utilisée.

js	function **yGetAPIVersion**( )
cpp	string **GetAPIVersion**( )
m	+(NSString*) **GetAPIVersion**
pas	string **yGetAPIVersion**( ): string
vb	function **GetAPIVersion**( ) As String
cs	static String **GetAPIVersion**( )
java	static String **GetAPIVersion**( )
uwp	static string **GetAPIVersion**( )
py	**GetAPIVersion**( )
php	function **GetAPIVersion**( )
ts	async **GetAPIVersion**( )
es	async **GetAPIVersion**( )
dnp	static string **GetAPIVersion**( )
cp	static string **GetAPIVersion**( )

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

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

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

YAPI.GetCacheValidity() YAPI

YAPI.GetCacheValidity()

Retourne la durée de validité des données chargée par la libraire.

cpp	static u64 **GetCacheValidity**( )
m	+(u64) **GetCacheValidity**
pas	u64 **yGetCacheValidity**( ): u64
vb	function **GetCacheValidity**( ) As Long
cs	ulong **GetCacheValidity**( )
java	long **GetCacheValidity**( )
uwp	async Task<ulong> **GetCacheValidity**( )
py	**GetCacheValidity**( )
php	function **GetCacheValidity**( )
ts	async **GetCacheValidity**( ): Promise<number>
es	async **GetCacheValidity**( )

Cette méthode retourne la durée de mise en cache de tous les attributs des fonctions du module. Note: Cette fonction doit être appelée après `yInitAPI`.

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

YAPI.GetDeviceListValidity() YAPI

YAPI.GetDeviceListValidity()

Retourne le délai entre chaque énumération forcée des YoctoHubs utilisés.

cpp	static int **GetDeviceListValidity**( )
m	+(int) **GetDeviceListValidity**
pas	LongInt **yGetDeviceListValidity**( ): LongInt
vb	function **GetDeviceListValidity**( ) As Integer
cs	int **GetDeviceListValidity**( )
java	int **GetDeviceListValidity**( )
uwp	async Task<int> **GetDeviceListValidity**( )
py	**GetDeviceListValidity**( )
php	function **GetDeviceListValidity**( )
ts	async **GetDeviceListValidity**( ): Promise<number>
es	async **GetDeviceListValidity**( )

Note: Cette fonction doit être appelée après `yInitAPI`.

Retourne :
le nombre de secondes entre chaque énumération.

YAPI.GetDllArchitecture() YAPI

YAPI.GetDllArchitecture()

Retourne l'architecture du binaire de la librairie de communication Yoctopuce utilisée.

dnp static string **GetDllArchitecture**( )

Pour Windows, l'architecture peut être `"Win32"` ou `"Win64"`. Pour les machines ARM, l'architecture est `"Armhf32"` ou `"Aarch64"`. Pour les autres machines Linux, l'architecture est `"Linux32"` ou `"Linux64"`. Pour MacOS, l'architecture est `"MacOs32"` ou `"MacOs64"`.

Retourne :
une chaîne de caractères décrivant l'architecture du binaire de la librairie de communication Yoctopuce utilisée.

YAPI.GetDllPath() YAPI

YAPI.GetDllPath()

Retourne l'emplacement sur le disque des librairies Yoctopuce utilisées.

dnp static string **GetDllPath**( )

Pour les architectures qui demandent l'utilisation de plusieurs DLLs, et en particulier dans le cadre de l'utilisation de la librairie sous forme de .NET Assembly, la chaîne retournée est au format suivant: `"DotNetProxy=/...; yapi=/...;"`, où le premier path correspond à l'emplacement de la DLL .NET Assembly, et le deuxième path correspond à la librairie de communication de bas niveau.

Retourne :
une chaîne de caractères décrivant l'emplacement de la DLL.

YAPI.GetLog() YAPI
YAPI.GetLog()

Récupère les messages de logs de la librairie de communication Yoctopuce.

dnp static string **GetLog**(string **lastLogLine**)

cp static string **GetLog**(string **lastLogLine**)

Cette méthode permet de récupérer progressivement les messages de logs, ligne par ligne. La méthode doit être appelée en boucle, jusqu'à ce qu'elle retourne une chaîne vide. Prenez garde à sortir de votre boucle dès qu'une chaîne vide est retournée, car si vous repassez une chaîne vide dans le paramètre `lastLogLine` lors de l'appel suivant, la fonction recommencera à énumérer les messages depuis le plus vieux message disponible.

Paramètres :

lastLogLine Lors du premier appel, passez une chaîne vide. Pour les appels suivants, passez le dernier message retourné par `GetLog()`.

Retourne :

une chaîne de caractères contenant le message de log qui suit immédiatement celui passé en argument, si une telle ligne existe. Si ce n'est pas le cas, lorsque tous les messages ont été retournés, retourne une chaîne vide.

YAPI.GetNetworkTimeout() YAPI

YAPI.GetNetworkTimeout()

Retourne le délai de connexion réseau pour `yRegisterHub()` et `yUpdateDeviceList()`.

cpp	static int **GetNetworkTimeout**( )
m	+(int) **GetNetworkTimeout**
pas	LongInt **yGetNetworkTimeout**( ): LongInt
vb	function **GetNetworkTimeout**( ) As Integer
cs	int **GetNetworkTimeout**( )
java	int **GetNetworkTimeout**( )
uwp	async Task<int> **GetNetworkTimeout**( )
py	**GetNetworkTimeout**( )
php	function **GetNetworkTimeout**( )
ts	async **GetNetworkTimeout**( ): Promise<number>
es	async **GetNetworkTimeout**( )
dnp	static int **GetNetworkTimeout**( )
cp	static int **GetNetworkTimeout**( )

Ce délai n'affecte que les YoctoHubs et VirtualHub qui sont accessibles par réseau. Par défaut, ce délai est de 20000 millisecondes, mais en fonction de votre réseau il peut être souhaitable de changer ce délai, par exemple, si votre infrastructure réseau utilise une connexion GSM.

Retourne :
le délai de connexion réseau en millisecondes.

YAPI.GetTickCount() YAPI

YAPI.GetTickCount()

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

```
js    function yGetTickCount( )
cpp   u64 GetTickCount( )
m     +(u64) GetTickCount
pas   u64 yGetTickCount( ): u64
vb    function GetTickCount( ) As Long
cs    static ulong GetTickCount( )
java  static long GetTickCount( )
uwp   static ulong GetTickCount( )
py    GetTickCount( )
php   function GetTickCount( )
ts    GetTickCount( ): number
es    GetTickCount( )
```

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

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

YAPI.HandleEvents() YAPI

YAPI.HandleEvents()

Maintient la communication de la librairie avec les modules Yoctopuce.

```
js    function yHandleEvents( errmsg)
cpp   YRETCODE HandleEvents( string errmsg)
m     +(YRETCODE) HandleEvents :(NSError**) errmsg
pas   integer yHandleEvents( var errmsg: string): integer
vb    function HandleEvents( ByRef errmsg As String) As YRETCODE
cs    static YRETCODE HandleEvents( ref string errmsg)
java  int HandleEvents( )
uwp   async Task<int> HandleEvents( )
py    HandleEvents( errmsg=None)
php   function HandleEvents( &$errmsg)
ts    async HandleEvents( errmsg: YErrorMsg | null): Promise<number>
es    async HandleEvents( errmsg)
```

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

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

Paramètres :

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

YAPI.InitAPI() YAPI

YAPI.InitAPI()

Initialise la librairie de programmation de Yoctopuce explicitement.

js	function **yInitAPI**(**mode**, **errmsg**)
cpp	YRETCODE **InitAPI**(int **mode**, string **errmsg**)
m	+(YRETCODE) **InitAPI** :(int) **mode** :(NSError**) **errmsg**
pas	integer **yInitAPI**(**mode**: integer, var **errmsg**: string): integer
vb	function **InitAPI**(ByVal **mode** As Integer, ByRef **errmsg** As String) As Integer
cs	static int **InitAPI**(int **mode**, ref string **errmsg**)
java	int **InitAPI**(int **mode**)
uwp	async Task<int> **InitAPI**(int **mode**)
py	**InitAPI**(**mode**, **errmsg**=None)
php	function **InitAPI**(**$mode**, **&$errmsg**)
ts	async **InitAPI**(**mode**: number, **errmsg**: YErrorMsg): Promise<number>
es	async **InitAPI**(**mode**, **errmsg**)

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

Lorsque cette fonctin est utilisée avec comme `mode` la valeur `YAPI.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 `YAPI.DETECT_NONE`, `YAPI.DETECT_USB`, `YAPI.DETECT_NET` et `YAPI.DETECT_ALL`.

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

YAPI.PreregisterHub() YAPI

YAPI.PreregisterHub()

Alternative plus tolerante à `yRegisterHub()`.

```
js    function yPreregisterHub( url, errmsg)
cpp   YRETCODE PreregisterHub( string url, string errmsg)
m     +(YRETCODE) PreregisterHub :(NSString *) url :(NSError**) errmsg
pas   integer yPreregisterHub( url: string, var errmsg: string): integer
vb    function PreregisterHub( ByVal url As String,
                               ByRef errmsg As String) As Integer
cs    static int PreregisterHub( string url, ref string errmsg)
java  int PreregisterHub( String url)
uwp   async Task<int> PreregisterHub( string url)
py    PreregisterHub( url, errmsg=None)
php   function PreregisterHub( $url, &$errmsg)
ts    async PreregisterHub( url: string, errmsg: YErrorMsg): Promise<number>
es    async PreregisterHub( url, errmsg)
dnp   static string PreregisterHub( string url)
cp    static string PreregisterHub( string url)
```

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

Paramètres :

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

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

YAPI.RegisterDeviceArrivalCallback() YAPI

YAPI.RegisterDeviceArrivalCallback()

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

```
js    function yRegisterDeviceArrivalCallback( arrivalCallback)
cpp   void RegisterDeviceArrivalCallback( yDeviceUpdateCallback arrivalCallback)
m     +(void) RegisterDeviceArrivalCallback :(yDeviceUpdateCallback) arrivalCallback
pas   yRegisterDeviceArrivalCallback( arrivalCallback: yDeviceUpdateFunc)
vb    procedure RegisterDeviceArrivalCallback( ByVal arrivalCallback As yDeviceUpdateFunc)
cs    static void RegisterDeviceArrivalCallback( yDeviceUpdateFunc arrivalCallback)
java  void RegisterDeviceArrivalCallback( DeviceArrivalCallback arrivalCallback)
uwp   void RegisterDeviceArrivalCallback( DeviceUpdateHandler arrivalCallback)
py    RegisterDeviceArrivalCallback( arrivalCallback)
php   function RegisterDeviceArrivalCallback( $arrivalCallback)
ts    async RegisterDeviceArrivalCallback( arrivalCallback: YDeviceUpdateCallback| null): Promise<void>
es    async RegisterDeviceArrivalCallback( arrivalCallback)
```

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

Paramètres :

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

YAPI.RegisterDeviceRemovalCallback() YAPI

YAPI.RegisterDeviceRemovalCallback()

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

js	function **yRegisterDeviceRemovalCallback**(**removalCallback**)
cpp	void **RegisterDeviceRemovalCallback**(yDeviceUpdateCallback **removalCallback**)
m	+(void) **RegisterDeviceRemovalCallback** :(yDeviceUpdateCallback) **removalCallback**
pas	**yRegisterDeviceRemovalCallback**(**removalCallback**: yDeviceUpdateFunc)
vb	procedure **RegisterDeviceRemovalCallback**(ByVal **removalCallback** As yDeviceUpdateFunc)
cs	static void **RegisterDeviceRemovalCallback**(yDeviceUpdateFunc **removalCallback**)
java	void **RegisterDeviceRemovalCallback**(DeviceRemovalCallback **removalCallback**)
uwp	void **RegisterDeviceRemovalCallback**(DeviceUpdateHandler **removalCallback**)
py	**RegisterDeviceRemovalCallback**(**removalCallback**)
php	function **RegisterDeviceRemovalCallback**(**$removalCallback**)
ts	async **RegisterDeviceRemovalCallback**(**removalCallback**: YDeviceUpdateCallback| null): Promise<void>
es	async **RegisterDeviceRemovalCallback**(**removalCallback**)

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

Paramètres :

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

YAPI.RegisterHub() YAPI
YAPI.RegisterHub()

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

```
js    function yRegisterHub( url, errmsg)
cpp   YRETCODE RegisterHub( string url, string errmsg)
m     +(YRETCODE) RegisterHub :(NSString *) url :(NSError**) errmsg
pas   integer yRegisterHub( url: string, var errmsg: string): integer
vb    function RegisterHub( ByVal url As String,
                            ByRef errmsg As String) As Integer
cs    static int RegisterHub( string url, ref string errmsg)
java  int RegisterHub( String url)
uwp   async Task<int> RegisterHub( string url)
py    RegisterHub( url, errmsg=None)
php   function RegisterHub( $url, &$errmsg)
ts    async RegisterHub( url: string, errmsg: YErrorMsg): Promise<number>
es    async RegisterHub( url, errmsg)
dnp   static string RegisterHub( string url)
cp    static string RegisterHub( string url)
```

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

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

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

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

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

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

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

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

Paramètres :

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

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

YAPI.RegisterHubDiscoveryCallback() YAPI
YAPI.RegisterHubDiscoveryCallback()

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

cpp	void **RegisterHubDiscoveryCallback**(YHubDiscoveryCallback **hubDiscoveryCallback**)
m	+(void) **RegisterHubDiscoveryCallback** : (YHubDiscoveryCallback) **hubDiscoveryCallback**
pas	**yRegisterHubDiscoveryCallback**(**hubDiscoveryCallback**: YHubDiscoveryCallback)
vb	procedure **RegisterHubDiscoveryCallback**(ByVal **hubDiscoveryCallback** As YHubDiscoveryCallback)
cs	static void **RegisterHubDiscoveryCallback**(YHubDiscoveryCallback **hubDiscoveryCallback**)
java	void **RegisterHubDiscoveryCallback**(HubDiscoveryCallback **hubDiscoveryCallback**)
uwp	async Task **RegisterHubDiscoveryCallback**(HubDiscoveryHandler **hubDiscoveryCallback**)
py	**RegisterHubDiscoveryCallback**(**hubDiscoveryCallback**)
ts	async **RegisterHubDiscoveryCallback**(**hubDiscoveryCallback**: YHubDiscoveryCallback): Promise<number>
es	async **RegisterHubDiscoveryCallback**(**hubDiscoveryCallback**)

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

Paramètres :

hubDiscoveryCallback une procédure qui prend deux chaîne de caractères en paramètre, le numéro de série et l'URL du hub découvert. Pour supprimer un callback déjà enregistré,

YAPI.RegisterHubWebsocketCallback() YAPI

YAPI.RegisterHubWebsocketCallback()

Variante de la fonction `yRegisterHub()` destinée à initialiser l'API Yoctopuce sur une session Websocket existante, dans le cadre d'un callback WebSocket entrant.

Paramètres :

ws l'objet WebSocket lié à au callback WebSocket entrant.

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

authpwd le mot de passe optionnel, nécessaire seulement si une authentification WebSocket est configurée sur le hub appelant.

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

YAPI.RegisterLogFunction() YAPI

YAPI.RegisterLogFunction()

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

cpp	void **RegisterLogFunction**(yLogFunction **logfun**)
m	+(void) **RegisterLogFunction** :(yLogCallback) **logfun**
pas	**yRegisterLogFunction**(**logfun**: yLogFunc)
vb	procedure **RegisterLogFunction**(ByVal **logfun** As yLogFunc)
cs	static void **RegisterLogFunction**(yLogFunc **logfun**)
java	void **RegisterLogFunction**(LogCallback **logfun**)
uwp	void **RegisterLogFunction**(LogHandler **logfun**)
py	**RegisterLogFunction**(**logfun**)
ts	async **RegisterLogFunction**(**logfun**: YLogCallback): Promise<number>
es	async **RegisterLogFunction**(**logfun**)

Utile pour débugger le fonctionnement de l'API.

Paramètres :

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

YAPI.SelectArchitecture() YAPI

YAPI.SelectArchitecture()

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

py **SelectArchitecture**(**arch**)

Par défaut, la libraire Python détecte automatiquement la version de la libraire dynamique à utiliser pour accéder au port USB. Sous Linux ARM il n'est pas possible de détecter de manière fiable si il s'agit d'une installation Soft float (armel) ou Hard float (armhf). Dans ce cas, il est donc recommendé d'appeler `SelectArchitecture()` avant tout autre appel à la librairie pour forcer l'utilisation d'une architecture spécifiée.

Paramètres :

arch une chaîne de caractère spécifiant l'architecture à utiliser. Les valeurs possibles sont `"armhf"`,`"armel"`, `"aarch64"`,`"i386"`,`"x86_64"`, `"32bit"`, `"64bit"`

Retourne :

rien.

En cas d'erreur, déclenche une exception.

YAPI.SetCacheValidity() YAPI

YAPI.SetCacheValidity()

Change la durée de validité des données chargées par la librairie.

cpp	static void **SetCacheValidity**(u64 **cacheValidityMs**)
m	+(void) **SetCacheValidity** : (u64) **cacheValidityMs**
pas	**ySetCacheValidity**(**cacheValidityMs**: u64)
vb	procedure **SetCacheValidity**(ByVal **cacheValidityMs** As Long)
cs	void **SetCacheValidity**(ulong **cacheValidityMs**)
java	void **SetCacheValidity**(long **cacheValidityMs**)
uwp	async Task **SetCacheValidity**(ulong **cacheValidityMs**)
py	**SetCacheValidity**(**cacheValidityMs**)
php	function **SetCacheValidity**(**$cacheValidityMs**)
ts	async **SetCacheValidity**(**cacheValidityMs**: number): Promise<void>
es	async **SetCacheValidity**(**cacheValidityMs**)

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 changer cette durée, par exemple dans le but de réduire le trafic réseau ou USB. Ce paramètre n'affecte pas les callbacks de changement de valeur Note: Cette fonction doit être appelée après `yInitAPI`.

Paramètres :

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

YAPI.SetDelegate() YAPI
YAPI.SetDelegate()

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

m +(void) **SetDelegate** :(id) **object**

Les méthodes `yDeviceArrival` et `yDeviceRemoval` seront appelées pendant l'exécution de la fonction `yUpdateDeviceList`, que vous devrez appeler régulièrement.

Paramètres :

object un objet qui soit se conformer au protocole `YAPIDelegate`, ou `nil`

YAPI.SetDeviceListValidity() YAPI

YAPI.SetDeviceListValidity()

Change le délai entre chaque énumération forcée des YoctoHub utilisés.

cpp	static void **SetDeviceListValidity**(int **deviceListValidity**)
m	+(void) **SetDeviceListValidity** : (int) **deviceListValidity**
pas	**ySetDeviceListValidity**(**deviceListValidity**: LongInt)
vb	procedure **SetDeviceListValidity**(ByVal **deviceListValidity** As Integer)
cs	void **SetDeviceListValidity**(int **deviceListValidity**)
java	void **SetDeviceListValidity**(int **deviceListValidity**)
uwp	async Task **SetDeviceListValidity**(int **deviceListValidity**)
py	**SetDeviceListValidity**(**deviceListValidity**)
php	function **SetDeviceListValidity**(**$deviceListValidity**)
ts	async **SetDeviceListValidity**(**deviceListValidity**: number): Promise<void>
es	async **SetDeviceListValidity**(**deviceListValidity**)

Par défaut la librairie effectue une énumération complète toute les 10 secondes. Pour réduire le trafic réseau, il est possible d'augmenter ce délai. C'est particulièrement utile lors un YoctoHub est connecté à un réseau GSM où le trafic est facturé. Ce paramètre n'affecte pas les modules connectés par USB, ni le fonctionnement des callback de connexion/déconnexion de module. Note: Cette fonction doit être appelée après `yInitAPI`.

Paramètres :

deviceListValidity nombre de secondes entre chaque énumération.

YAPI.SetHTTPCallbackCacheDir() YAPI

YAPI.SetHTTPCallbackCacheDir()

Active le cache du callback HTTP.

```
php function SetHTTPCallbackCacheDir( $str_directory)
```

Le cache du callback HTTP permet de réduire de 50 à 70 % la quantité de données transmise au script PHP. Pour activer le cache, la méthode `ySetHTTPCallbackCacheDir()` doit être appelée avant premier appel à `yRegisterHub()`. Cette méthode prend en paramètre le path du répertoire dans lequel seront stockés les données entre chaque callback. Ce répertoire doit exister et le script PHP doit y avoir les droits d'écriture. Il est recommandé d'utiliser un répertoire qui n'est pas accessible depuis le serveur Web, car la librairie va y stocker des données provenant des modules Yoctopuce.

Note : Cette fonctionnalité est supportée par les YoctoHub et VirtualHub depuis la version 27750

Paramètres :

str_directory le path du répertoire utilisé comme cache.

Retourne :

nothing.

On failure, throws an exception.

YAPI.SetNetworkTimeout() YAPI

YAPI.SetNetworkTimeout()

Change le délai de connexion réseau pour `yRegisterHub()` et `yUpdateDeviceList()`.

cpp	static void **SetNetworkTimeout**(int **networkMsTimeout**)
m	+(void) **SetNetworkTimeout** : (int) **networkMsTimeout**
pas	**ySetNetworkTimeout**(**networkMsTimeout**: LongInt)
vb	procedure **SetNetworkTimeout**(ByVal **networkMsTimeout** As Integer)
cs	void **SetNetworkTimeout**(int **networkMsTimeout**)
java	void **SetNetworkTimeout**(int **networkMsTimeout**)
uwp	async Task **SetNetworkTimeout**(int **networkMsTimeout**)
py	**SetNetworkTimeout**(**networkMsTimeout**)
php	function **SetNetworkTimeout**(**$networkMsTimeout**)
ts	async **SetNetworkTimeout**(**networkMsTimeout**: number): Promise<void>
es	async **SetNetworkTimeout**(**networkMsTimeout**)
dnp	static void **SetNetworkTimeout**(int **networkMsTimeout**)
cp	static void **SetNetworkTimeout**(int **networkMsTimeout**)

Ce délai n'affecte que les YoctoHubs et VirtualHub qui sont accessibles par réseau. Par défaut ce délai est de 20000 millisecondes, mais en fonction de votre réseau il peut être souhaitable de changer ce délai, par exemple si votre infrastructure réseau utilise une connexion GSM.

Paramètres :

networkMsTimeout le délais de connexion réseau en millisecondes.

YAPI.SetTimeout() YAPI

YAPI.SetTimeout()

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

```
js  function ySetTimeout( callback, ms_timeout, args)
ts  SetTimeout( callback: Function, ms_timeout: number): number
es  SetTimeout( callback, ms_timeout, args)
```

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

Paramètres :

callback la fonction à appeler lorsque le temps d'attente est écoulé. Sous Microsoft Internet Explorer, le callback doit être spécifié sous forme d'une string à évaluer.

ms_timeout un entier correspondant à la durée de l'attente, en millisecondes

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

YAPI.SetUSBPacketAckMs() YAPI

YAPI.SetUSBPacketAckMs()

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

```
java void SetUSBPacketAckMs( int pktAckDelay)
```

Cette fonction permet à la librairie de fonctionner même sur les téléphones Android qui perdent des paquets USB. Par défaut, la quittance est désactivée, car elle double le nombre de paquets échangés et donc ralentit sensiblement le fonctionnement de L'API. La quittance des paquets USB ne doit donc être activée que sur des tablette ou des téléphones qui posent problème. Un délais de 50 millisecondes est en général suffisant. En cas de doute contacter le support Yoctopuce. Pour désactiver la quittance des paquets USB, appeler cette fonction avec la valeur 0. Note : Cette fonctionnalité est disponible uniquement sous Android.

Paramètres :

pktAckDelay nombre de ms avant que le module ne renvoie

YAPI.Sleep() YAPI

YAPI.Sleep()

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

```
js    function ySleep( ms_duration, errmsg)
cpp   YRETCODE Sleep( unsigned ms_duration, string errmsg)
m     +(YRETCODE) Sleep :(unsigned) ms_duration :(NSError **) errmsg
pas   integer ySleep( ms_duration: integer, var errmsg: string): integer
vb    function Sleep( ByVal ms_duration As Integer,
                      ByRef errmsg As String) As Integer
cs    static int Sleep( int ms_duration, ref string errmsg)
java  int Sleep( long ms_duration)
uwp   async Task<int> Sleep( ulong ms_duration)
py    Sleep( ms_duration, errmsg=None)
php   function Sleep( $ms_duration, &$errmsg)
ts    async Sleep( ms_duration: number, errmsg: YErrorMsg | null): Promise<number>
es    async Sleep( ms_duration, errmsg)
```

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

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

Paramètres :

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

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

YAPI.TestHub() YAPI
YAPI.TestHub()

Test si un hub est joignable.

```
cpp   YRETCODE TestHub( string url, int mstimeout, string errmsg)
m     +(YRETCODE) TestHub : (NSString*) url
                          : (int) mstimeout
                          : (NSError**) errmsg
pas   integer yTestHub( url: string,
                        mstimeout: integer,
                        var errmsg: string): integer
vb    function TestHub( ByVal url As String,
                        ByVal mstimeout As Integer,
                        ByRef errmsg As String) As Integer
cs    static int TestHub( string url, int mstimeout, ref string errmsg)
java  int TestHub( String url, int mstimeout)
uwp   async Task<int> TestHub( string url, uint mstimeout)
py    TestHub( url, mstimeout, errmsg=None)
php   function TestHub( $url, $mstimeout, &$errmsg)
ts    async TestHub( url: string, mstimeout: number, errmsg: YErrorMsg): Promise<number>
es    async TestHub( url, mstimeout, errmsg)
dnp   static string TestHub( string url, int mstimeout)
cp    static string TestHub( string url, int mstimeout)
```

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

Paramètres :

url	une chaîne de caractères contenant **"usb"**,**"callback",** ou l'URL racine du VirtualHub à utiliser.
mstimeout	le nombre de millisecondes disponible pour tester la connexion.
errmsg	une chaîne de caractères passée par référence, dans laquelle sera stocké un éventuel message d'erreur.

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur retourne un code d'erreur négatif.

YAPI.TriggerHubDiscovery() YAPI

YAPI.TriggerHubDiscovery()

Relance une détection des hubs réseau.

cpp	YRETCODE **TriggerHubDiscovery**(string **errmsg**)
m	+(YRETCODE) **TriggerHubDiscovery** : (NSError**) **errmsg**
pas	integer **yTriggerHubDiscovery**(var **errmsg**: string): integer
vb	function **TriggerHubDiscovery**(ByRef **errmsg** As String) As Integer
cs	static int **TriggerHubDiscovery**(ref string **errmsg**)
java	int **TriggerHubDiscovery**( )
uwp	Task<int> **TriggerHubDiscovery**( )
py	**TriggerHubDiscovery**(**errmsg**=None)
ts	async **TriggerHubDiscovery**(**errmsg**: YErrorMsg | null): Promise<number>
es	async **TriggerHubDiscovery**(**errmsg**)

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

Paramètres :

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

Retourne :

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

YAPI.UnregisterHub() YAPI

YAPI.UnregisterHub()

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

```
js    function yUnregisterHub( url)
cpp   void UnregisterHub( string url)
m     +(void) UnregisterHub :(NSString *) url
pas   yUnregisterHub( url: string)
vb    procedure UnregisterHub( ByVal url As String)
cs    static void UnregisterHub( string url)
java  void UnregisterHub( String url)
uwp   async Task UnregisterHub( string url)
py    UnregisterHub( url)
php   function UnregisterHub( $url)
ts    async UnregisterHub( url: string): Promise<void>
es    async UnregisterHub( url)
```

Paramètres :

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

YAPI.UpdateDeviceList() YAPI

YAPI.UpdateDeviceList()

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

js	function **yUpdateDeviceList**(**errmsg**)
cpp	YRETCODE **UpdateDeviceList**(string **errmsg**)
m	+(YRETCODE) **UpdateDeviceList** :(NSError**) **errmsg**
pas	integer **yUpdateDeviceList**(var **errmsg**: string): integer
vb	function **UpdateDeviceList**(ByRef **errmsg** As String) As YRETCODE
cs	static YRETCODE **UpdateDeviceList**(ref string **errmsg**)
java	int **UpdateDeviceList**( )
uwp	async Task<int> **UpdateDeviceList**( )
py	**UpdateDeviceList**(**errmsg**=None)
php	function **UpdateDeviceList**(**&$errmsg**)
ts	async **UpdateDeviceList**(**errmsg**: YErrorMsg | null): Promise<number>
es	async **UpdateDeviceList**(**errmsg**)

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

Cette fonction peut être appelée aussi souvent que désiré, afin de rendre l'application réactive aux évènements de hot-plug. Néanmoins la détection des modules étant un processus assez lourd, il est recommandé de ne pas appeler UpdateDeviceList plus d'une fois toutes les deux secondes.

Paramètres :

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

YAPI.UpdateDeviceList_async() YAPI

YAPI.UpdateDeviceList_async()

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

```
js function yUpdateDeviceList_async( callback, context)
```

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.

24.2. La classe YModule

Interface de contrôle des paramètres généraux des modules Yoctopuce

La classe `YModule` est utilisable avec tous les modules USB de Yoctopuce. Elle permet de contrôler les paramètres généraux du module, et d'énumérer les fonctions fournies par chaque module.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_api.js'></script>
cpp	#include "yocto_api.h"
m	#import "yocto_api.h"
pas	uses yocto_api;
vb	yocto_api.vb
cs	yocto_api.cs
java	import com.yoctopuce.YoctoAPI.YModule;
uwp	import com.yoctopuce.YoctoAPI.YModule;
py	from yocto_api import *
php	require_once('yocto_api.php');
ts	in HTML: import { YAPI, YErrorMsg, YModule, YSensor } from '../../dist/esm/yocto_api_browser.js'; in Node.js: import { YAPI, YErrorMsg, YModule, YSensor } from 'yoctolib-cjs/yocto_api_nodejs.js';
es	in HTML: <script src="../../lib/yocto_api.js"></script> in node.js: require('yoctolib-es2017/yocto_api.js');
dnp	import YoctoProxyAPI.YModuleProxy
cp	#include "yocto_module_proxy.h"
vi	YModule.vi
ml	import YoctoProxyAPI.YModuleProxy"

Fonction globales
YModule.FindModule(**func**) Permet de retrouver un module d'après son numéro de série ou son nom logique.
YModule.FindModuleInContext(**yctx**, **func**) Permet de retrouver un module d'après un identifiant donné dans un Context YAPI.
YModule.FirstModule() Commence l'énumération des modules accessibles par la librairie.
Propriétés des objets `YModuleProxy`
module→Beacon *[modifiable]* état de la balise de localisation.
module→FirmwareRelease *[lecture seule]* Version du logiciel embarqué du module.
module→FunctionId *[lecture seule]* Identifiant matériel de la *n*ième fonction du module.
module→HardwareId *[lecture seule]* Identifiant unique du module.
module→IsOnline *[lecture seule]* Vérifie si le module est joignable.
module→LogicalName *[modifiable]* Nom logique du module.
module→Luminosity *[modifiable]*

Luminosité des leds informatives du module (valeur entre 0 et 100).

module→ProductId *[lecture seule]*

Identifiant USB du module, préprogrammé en usine.

module→ProductName *[lecture seule]*

Nom commercial du module, préprogrammé en usine.

module→ProductRelease *[lecture seule]*

Numéro uméro de révision du module hardware, préprogrammé en usine.

module→SerialNumber *[lecture seule]*

Numéro de série du module, préprogrammé en usine.

Méthodes des objets `YModule`

module→checkFirmware(path, onlynew)

Teste si le fichier byn est valide pour le module.

module→clearCache()

Invalide le cache.

module→describe()

Retourne un court texte décrivant le module.

module→download(pathname)

Télécharge le fichier choisi du module et retourne son contenu.

module→functionBaseType(functionIndex)

Retourne le type de base de la *n*ième fonction du module.

module→functionCount()

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

module→functionId(functionIndex)

Retourne l'identifiant matériel de la *n*ième fonction du module.

module→functionName(functionIndex)

Retourne le nom logique de la *n*ième fonction du module.

module→functionType(functionIndex)

Retourne le type de la *n*ième fonction du module.

module→functionValue(functionIndex)

Retourne la valeur publiée par la *n*ième fonction du module.

module→get_allSettings()

Retourne tous les paramètres de configuration du module.

module→get_beacon()

Retourne l'état de la balise de localisation.

module→get_errorMessage()

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

module→get_errorType()

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

module→get_firmwareRelease()

Retourne la version du logiciel embarqué du module.

module→get_functionIds(funType)

Retourne les identifiants matériels des fonctions correspondant au type passé en argument.

module→get_hardwareId()

Retourne l'identifiant unique du module.

module→get_icon2d()

Retourne l'icône du module.

module→get_lastLogs()

Retourne une chaine de charactère contenant les derniers logs du module.

module→get_logicalName()

Retourne le nom logique du module.

module→get_luminosity()

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

module→get_parentHub()

Retourne le numéro de série du YoctoHub sur lequel est connecté le module.

module→get_persistentSettings()

Retourne l'état courant des réglages persistents du module.

module→get_productId()

Retourne l'identifiant USB du module, préprogrammé en usine.

module→get_productName()

Retourne le nom commercial du module, préprogrammé en usine.

module→get_productRelease()

Retourne le numéro uméro de révision du module hardware, préprogrammé en usine.

module→get_rebootCountdown()

Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

module→get_serialNumber()

Retourne le numéro de série du module, préprogrammé en usine.

module→get_subDevices()

Retourne la liste des modules branchés au module courant.

module→get_upTime()

Retourne le numbre de millisecondes écoulées depuis la mise sous tension du module

module→get_url()

Retourne l'URL utilisée pour accéder au module.

module→get_usbCurrent()

Retourne le courant consommé par le module sur le bus USB, en milliampères.

module→get_userData()

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

module→get_userVar()

Retourne la valeur entière précédemment stockée dans cet attribut.

module→hasFunction(funcId)

Teste la présence d'une fonction pour le module courant.

module→isOnline()

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

module→isOnline_async(callback, context)

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

module→load(msValidity)

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

module→load_async(msValidity, callback, context)

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

module→log(text)

Ajoute un message arbitraire dans les logs du module.

module→nextModule()

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

module→reboot(secBeforeReboot)

Agende un simple redémarrage du module dans un nombre donné de secondes.

module→registerBeaconCallback(callback)

Enregistre une fonction de callback qui sera appelée à chaque changement d'état de la balise de localisation du module.

module→registerConfigChangeCallback(callback)

Enregistre une fonction de callback qui sera appelée à chaque fois qu'un réglage persistant d'un module est modifié (par exemple changement d'unité de mesure, etc.)

module→registerLogCallback(callback)

Enregistre une fonction de callback qui sera appelée à chaque fois le module émet un message de log.

module→revertFromFlash()

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

module→saveToFlash()

Sauve les réglages courants dans la mémoire non volatile du module.

module→set_allSettings(settings)

Rétablit tous les paramètres du module.

module→set_allSettingsAndFiles(settings)

Rétablit tous les paramètres de configuration et fichiers sur un module.

module→set_beacon(newval)

Allume ou éteint la balise de localisation du module.

module→set_logicalName(newval)

Change le nom logique du module.

module→set_luminosity(newval)

Modifie la luminosité des leds informatives du module.

module→set_userData(data)

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

module→set_userVar(newval)

Stocke une valeur 32 bits dans la mémoire volatile du module.

module→triggerConfigChangeCallback()

Force le déclanchement d'un callback de changement de configuration, afin de vérifier si ils sont disponibles ou pas.

module→triggerFirmwareUpdate(secBeforeReboot)

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

module→updateFirmware(path)

Prepare une mise à jour de firmware du module.

module→updateFirmwareEx(path, force)

Prepare une mise à jour de firmware du module.

module→wait_async(callback, context)

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

YModule.FindModule() YModule
YModule.FindModule()

Permet de retrouver un module d'après son numéro de série ou son nom logique.

js	function **yFindModule**(**func**)
cpp	YModule* **FindModule**(string **func**)
m	+(YModule*) **FindModule** : (NSString*) **func**
pas	TYModule **yFindModule**(**func**: string): TYModule
vb	function **FindModule**(ByVal **func** As String) As YModule
cs	static YModule **FindModule**(string **func**)
java	static YModule **FindModule**(String **func**)
uwp	static YModule **FindModule**(string **func**)
py	**FindModule**(**func**)
php	function **FindModule**(**$func**)
ts	static **FindModule**(**func**: string): YModule
es	static **FindModule**(**func**)
dnp	static YModuleProxy **FindModule**(string **func**)
cp	static YModuleProxy * **FindModule**(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.

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

Paramètres :

func une chaîne de caractères contenant soit le numéro de série, soit le nom logique du module désiré

Retourne :

un objet de classe `YModule` qui permet ensuite de contrôler le module ou d'obtenir de plus amples informations sur le module.

YModule.FindModuleInContext() YModule
YModule.FindModuleInContext()

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

```
java  static YModule FindModuleInContext( YAPIContext yctx, String func)
uwp   static YModule FindModuleInContext( YAPIContext yctx, string func)
ts    static FindModuleInContext( yctx: YAPIContext, func: string): YModule
es    static FindModuleInContext( yctx, func)
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence le module sans ambiguïté, par exemple `MyDevice.module`.

Retourne :

un objet de classe `YModule` qui permet ensuite de contrôler le module.

YModule.FirstModule() YModule

YModule.FirstModule()

Commence l'énumération des modules accessibles par la librairie.

js	function **yFirstModule**( )
cpp	YModule * **FirstModule**( )
m	+(YModule*) **FirstModule**
pas	TYModule **yFirstModule**( ): TYModule
vb	function **FirstModule**( ) As YModule
cs	static YModule **FirstModule**( )
java	static YModule **FirstModule**( )
uwp	static YModule **FirstModule**( )
py	**FirstModule**( )
php	function **FirstModule**( )
ts	static **FirstModule**( ): YModule | null
es	static **FirstModule**( )

Utiliser la fonction `YModule.nextModule()` pour itérer sur les autres modules.

Retourne :

un pointeur sur un objet `YModule`, correspondant au premier module accessible en ligne, ou `null` si aucun module n'a été trouvé.

module→Beacon YModule

état de la balise de localisation.

dnp int **Beacon**

Modifiable. Allume ou éteint la balise de localisation du module.

module→FirmwareRelease YModule

Version du logiciel embarqué du module.

dnp string **FirmwareRelease**

module→FunctionId YModule

Identifiant matériel de la *n*ième fonction du module.

dnp string **FunctionId**

@param functionIndex : l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

module→HardwareId YModule

Identifiant unique du module.

dnp string **HardwareId**

L'identifiant unique est composé du numéro de série du module suivi de la chaîne ".module".

module→IsOnline YModule

Vérifie si le module est joignable.

dnp 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.

module→LogicalName YModule

Nom logique du module.

dnp string **LogicalName**

Modifiable. Change le nom logique du module. 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.

module→Luminosity YModule

Luminosité des leds informatives du module (valeur entre 0 et 100).

dnp int **Luminosity**

Modifiable. Modifie la luminosité des leds informatives du module. Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

module→ProductId YModule

Identifiant USB du module, préprogrammé en usine.

dnp int **ProductId**

module→ProductName YModule

Nom commercial du module, préprogrammé en usine.

dnp string **ProductName**

module→ProductRelease YModule

Numéro uméro de révision du module hardware, préprogrammé en usine.

dnp int **ProductRelease**

La révision originale du retourne la valeur 1, la révision B retourne la valeur 2, etc.

module→SerialNumber YModule

Numéro de série du module, préprogrammé en usine.

dnp string **SerialNumber**

module→checkFirmware() YModule

Teste si le fichier byn est valide pour le module.

```
js    function checkFirmware( path, onlynew)
cpp   string checkFirmware( string path, bool onlynew)
m     -(NSString*) checkFirmware : (NSString*) path
                                 : (bool) onlynew
pas   string checkFirmware( path: string, onlynew: boolean): string
vb    function checkFirmware( ByVal path As String, ByVal onlynew As Boolean) As String
cs    string checkFirmware( string path, bool onlynew)
java  String checkFirmware( String path, boolean onlynew)
uwp   async Task<string> checkFirmware( string path, bool onlynew)
py    checkFirmware( path, onlynew)
php   function checkFirmware( $path, $onlynew)
ts    async checkFirmware( path: string, onlynew: boolean): Promise<string>
es    async checkFirmware( path, onlynew)
dnp   string checkFirmware( string path, bool onlynew)
cp    string checkFirmware( string path, bool onlynew)
cmd   YModule target checkFirmware path onlynew
```

Cette méthode est utile pour vérifier si il est nécessaire de mettre à jour le module avec un nouveau firmware. Il est possible de passer un répertoire qui contiens plusieurs fichier `.byn`. Dans ce cas cette methode retourne le path du fichier `.byn` compatible le plus récent. Si le parametre `onlynew` est vrais, les firmwares équivalents ou plus anciens que le firmware actuellement installé sont ignorés.

Paramètres :

path le path d'un fichier `.byn` ou d'un répertoire contenant plusieurs fichier `.byn`

onlynew retourne uniquement les fichiers strictement plus récents

Retourne :

le path du fichier `.byn` à utiliser, ou une chaîne vide si aucun firmware plus récent n'est disponible En cas d'erreur, déclenche une exception ou retourne une chaine de caractère qui comment par "error:".

module→clearCache() YModule

Invalide le cache.

js	function **clearCache**( )
cpp	void **clearCache**( )
m	-(void) **clearCache**
pas	**clearCache**( )
vb	procedure **clearCache**( )
cs	void **clearCache**( )
java	void **clearCache**( )
py	**clearCache**( )
php	function **clearCache**( )
ts	async **clearCache**( ): Promise<void>
es	async **clearCache**( )

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

module→describe() YModule

Retourne un court texte décrivant le module.

```
js    function describe( )
cpp   string describe( )
m     -(NSString*) describe
pas   string describe( ): string
vb    function describe( ) As String
cs    string describe( )
java  String describe( )
py    describe( )
php   function describe( )
ts    async describe( ): Promise<string>
es    async describe( )
```

Ce texte peut contenir soit le nom logique du module, soit son numéro de série.

Retourne :
une chaîne de caractères décrivant le module

module→download() YModule

Télécharge le fichier choisi du module et retourne son contenu.

```
js      function download( pathname)
cpp     string download( string pathname)
m       -(NSMutableData*) download : (NSString*) pathname
pas     TByteArray download( pathname: string): TByteArray
vb      function download( ByVal pathname As String) As Byte
cs      byte[] download( string pathname)
java    byte[] download( String pathname)
uwp     async Task<byte[]> download( string pathname)
py      download( pathname)
php     function download( $pathname)
ts      async download( pathname: string): Promise<Uint8Array>
es      async download( pathname)
dnp     byte[] download( string pathname)
cp      string download( string pathname)
cmd     YModule target download pathname
```

Paramètres :

pathname nom complet du fichier

Retourne :

le contenu du fichier chargé

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

module→functionBaseType() YModule

Retourne le type de base de la *n*ième fonction du module.

```
js    function functionBaseType( functionIndex)
cpp   string functionBaseType( int functionIndex)
pas   string functionBaseType( functionIndex: integer): string
vb    function functionBaseType( ByVal functionIndex As Integer) As String
cs    string functionBaseType( int functionIndex)
java  String functionBaseType( int functionIndex)
py    functionBaseType( functionIndex)
php   function functionBaseType( $functionIndex)
ts    async functionBaseType( functionIndex: number): Promise<string>
es    async functionBaseType( functionIndex)
```

Par exemple, le type de base de toutes les fonctions de mesure est "Sensor".

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant au type de base de la fonction

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

module→functionCount() YModule

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

js	function **functionCount**( )
cpp	int **functionCount**( )
m	-(int) **functionCount**
pas	integer **functionCount**( ): integer
vb	function **functionCount**( ) As Integer
cs	int **functionCount**( )
java	int **functionCount**( )
py	**functionCount**( )
php	function **functionCount**( )
ts	async **functionCount**( ): Promise<number>
es	async **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() YModule

Retourne l'identifiant matériel de la *n*ième fonction du module.

```
js    function functionId( functionIndex)
cpp   string functionId( int functionIndex)
m     -(NSString*) functionId : (int) functionIndex
pas   string functionId( functionIndex: integer): string
vb    function functionId( ByVal functionIndex As Integer) As String
cs    string functionId( int functionIndex)
java  String functionId( int functionIndex)
py    functionId( functionIndex)
php   function functionId( $functionIndex)
ts    async functionId( functionIndex: number): Promise<string>
es    async functionId( functionIndex)
```

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant à l'identifiant matériel unique de la fonction désirée

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

module→functionName() YModule

Retourne le nom logique de la *n*ième fonction du module.

```
js    function functionName( functionIndex)
cpp   string functionName( int functionIndex)
m     -(NSString*) functionName : (int) functionIndex
pas   string functionName( functionIndex: integer): string
vb    function functionName( ByVal functionIndex As Integer) As String
cs    string functionName( int functionIndex)
java  String functionName( int functionIndex)
py    functionName( functionIndex)
php   function functionName( $functionIndex)
ts    async functionName( functionIndex: number): Promise<string>
es    async functionName( functionIndex)
```

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant au nom logique de la fonction désirée

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

module→functionType() YModule

Retourne le type de la *n*ième fonction du module.

```
js    function functionType( functionIndex)
cpp   string functionType( int functionIndex)
pas   string functionType( functionIndex: integer): string
vb    function functionType( ByVal functionIndex As Integer) As String
cs    string functionType( int functionIndex)
java  String functionType( int functionIndex)
py    functionType( functionIndex)
php   function functionType( $functionIndex)
ts    async functionType( functionIndex: number): Promise<string>
es    async functionType( functionIndex)
```

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant au type de la fonction

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

module→functionValue() YModule

Retourne la valeur publiée par la *n*ième fonction du module.

```
js    function functionValue( functionIndex)
cpp   string functionValue( int functionIndex)
m     -(NSString*) functionValue : (int) functionIndex
pas   string functionValue( functionIndex: integer): string
vb    function functionValue( ByVal functionIndex As Integer) As String
cs    string functionValue( int functionIndex)
java  String functionValue( int functionIndex)
py    functionValue( functionIndex)
php   function functionValue( $functionIndex)
ts    async functionValue( functionIndex: number): Promise<string>
es    async functionValue( functionIndex)
```

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant à la valeur publiée par la fonction désirée

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

module→get_allSettings() YModule
module→allSettings()

Retourne tous les paramètres de configuration du module.

```
js    function get_allSettings( )
cpp   string get_allSettings( )
m     -(NSMutableData*) allSettings
pas   TByteArray get_allSettings( ): TByteArray
vb    function get_allSettings( ) As Byte
cs    byte[] get_allSettings( )
java  byte[] get_allSettings( )
uwp   async Task<byte[]> get_allSettings( )
py    get_allSettings( )
php   function get_allSettings( )
ts    async get_allSettings( ): Promise<Uint8Array>
es    async get_allSettings( )
dnp   byte[] get_allSettings( )
cp    string get_allSettings( )
cmd   YModule target get_allSettings
```

Utile pour sauvgarder les noms logiques, les calibrations et fichies uploadés d'un module.

Retourne :
un objet binaire avec tous les paramètres

En cas d'erreur, déclenche une exception ou retourne un objet binaire de taille 0.

module→get_beacon() YModule
module→beacon()

Retourne l'état de la balise de localisation.

```
js    function get_beacon( )
cpp   Y_BEACON_enum get_beacon( )
m     -(Y_BEACON_enum) beacon
pas   Integer get_beacon( ): Integer
vb    function get_beacon( ) As Integer
cs    int get_beacon( )
java  int get_beacon( )
uwp   async Task<int> get_beacon( )
py    get_beacon( )
php   function get_beacon( )
ts    async get_beacon( ): Promise<YModule_Beacon>
es    async get_beacon( )
dnp   int get_beacon( )
cp    int get_beacon( )
cmd   YModule target get_beacon
```

Retourne :
soit `YModule.BEACON_OFF`, soit `YModule.BEACON_ON`, selon l'état de la balise de localisation

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

module→get_errorMessage() YModule
module→errorMessage()

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

js	function **get_errorMessage**( )
cpp	string **get_errorMessage**( )
m	-(NSString*) errorMessage
pas	string **get_errorMessage**( ): string
vb	function **get_errorMessage**( ) As String
cs	string **get_errorMessage**( )
java	String **get_errorMessage**( )
py	**get_errorMessage**( )
php	function **get_errorMessage**( )
ts	**get_errorMessage**( ): string
es	**get_errorMessage**( )

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

Retourne :
une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du module

module→get_errorType() YModule
module→errorType()

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

```
js      function get_errorType( )
cpp     YRETCODE get_errorType( )
m       -(YRETCODE) errorType
pas     YRETCODE get_errorType( ): YRETCODE
vb      function get_errorType( ) As YRETCODE
cs      YRETCODE get_errorType( )
java    int get_errorType( )
py      get_errorType( )
php     function get_errorType( )
ts      get_errorType( ): number
es      get_errorType( )
```

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

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

module→get_firmwareRelease() YModule
module→firmwareRelease()

Retourne la version du logiciel embarqué du module.

```
js    function get_firmwareRelease( )
cpp   string get_firmwareRelease( )
m     -(NSString*) firmwareRelease
pas   string get_firmwareRelease( ): string
vb    function get_firmwareRelease( ) As String
cs    string get_firmwareRelease( )
java  String get_firmwareRelease( )
uwp   async Task<string> get_firmwareRelease( )
py    get_firmwareRelease( )
php   function get_firmwareRelease( )
ts    async get_firmwareRelease( ): Promise<string>
es    async get_firmwareRelease( )
dnp   string get_firmwareRelease( )
cp    string get_firmwareRelease( )
cmd   YModule target get_firmwareRelease
```

Retourne :
une chaîne de caractères représentant la version du logiciel embarqué du module

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

module→get_functionIds() YModule
module→functionIds()

Retourne les identifiants matériels des fonctions correspondant au type passé en argument.

```
js    function get_functionIds( funType)
cpp   vector<string> get_functionIds( string funType)
m     -(NSMutableArray*) functionIds : (NSString*) funType
pas   TStringArray get_functionIds( funType: string): TStringArray
vb    function get_functionIds( ByVal funType As String) As List
cs    List<string> get_functionIds( string funType)
java  ArrayList<String> get_functionIds( String funType)
uwp   async Task<List<string>> get_functionIds( string funType)
py    get_functionIds( funType)
php   function get_functionIds( $funType)
ts    async get_functionIds( funType: string): Promise<string[]
es    async get_functionIds( funType)
dnp   string[] get_functionIds( string funType)
cp    vector<string> get_functionIds( string funType)
cmd   YModule target get_functionIds funType
```

Paramètres :

funType Le type de fonction (Relay, LightSensor, Voltage,...)

Retourne :

un tableau de chaînes de caractère.

module→get_hardwareId() YModule
module→hardwareId()

Retourne l'identifiant unique du module.

js	function **get_hardwareId**( )
cpp	string **get_hardwareId**( )
m	-(NSString*) hardwareId
vb	function **get_hardwareId**( ) As String
cs	string **get_hardwareId**( )
java	String **get_hardwareId**( )
py	**get_hardwareId**( )
php	function **get_hardwareId**( )
ts	async **get_hardwareId**( ): Promise<string>
es	async **get_hardwareId**( )
dnp	string **get_hardwareId**( )
cp	string **get_hardwareId**( )
pas	string **get_hardwareId**( ): string
uwp	async Task<string> **get_hardwareId**( )
cmd	YModule **target get_hardwareId**

L'identifiant unique est composé du numéro de série du module suivi de la chaîne ".module".

Retourne :
une chaîne de caractères identifiant la fonction

module→get_icon2d() YModule
module→icon2d()

Retourne l'icône du module.

```
js    function get_icon2d( )
cpp   string get_icon2d( )
m     -(NSMutableData*) icon2d
pas   TByteArray get_icon2d( ): TByteArray
vb    function get_icon2d( ) As Byte
cs    byte[] get_icon2d( )
java  byte[] get_icon2d( )
uwp   async Task<byte[]> get_icon2d( )
py    get_icon2d( )
php   function get_icon2d( )
ts    async get_icon2d( ): Promise<Uint8Array>
es    async get_icon2d( )
dnp   byte[] get_icon2d( )
cp    string get_icon2d( )
cmd   YModule target get_icon2d
```

L'icone est au format PNG et a une taille maximale de 1536 octets.

Retourne :
un buffer binaire contenant l'icone, au format png. En cas d'erreur, déclenche une exception ou retourne `YAPI_INVALID_STRING`.

module→get_lastLogs() YModule
module→lastLogs()

Retourne une chaine de charactère contenant les derniers logs du module.

js	function **get_lastLogs**( )
cpp	string **get_lastLogs**( )
m	-(NSString*) lastLogs
pas	string **get_lastLogs**( ): string
vb	function **get_lastLogs**( ) As String
cs	string **get_lastLogs**( )
java	String **get_lastLogs**( )
uwp	async Task<string> **get_lastLogs**( )
py	**get_lastLogs**( )
php	function **get_lastLogs**( )
ts	async **get_lastLogs**( ): Promise<string>
es	async **get_lastLogs**( )
dnp	string **get_lastLogs**( )
cp	string **get_lastLogs**( )
cmd	YModule **target get_lastLogs**

Cette méthode retourne les derniers logs qui sont encore stocké dans le module.

Retourne :
une chaîne de caractère contenant les derniers logs du module. En cas d'erreur, déclenche une exception ou retourne `YAPI_INVALID_STRING`.

module→get_logicalName() YModule
module→logicalName()

Retourne le nom logique du module.

```
js    function get_logicalName( )
cpp   string get_logicalName( )
m     -(NSString*) logicalName
pas   string get_logicalName( ): string
vb    function get_logicalName( ) As String
cs    string get_logicalName( )
java  String get_logicalName( )
uwp   async Task<string> get_logicalName( )
py    get_logicalName( )
php   function get_logicalName( )
ts    async get_logicalName( ): Promise<string>
es    async get_logicalName( )
dnp   string get_logicalName( )
cp    string get_logicalName( )
cmd   YModule target get_logicalName
```

Retourne :
une chaîne de caractères représentant le nom logique du module

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

module→get_luminosity() YModule
module→luminosity()

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

```
js    function get_luminosity( )
cpp   int get_luminosity( )
m     -(int) luminosity
pas   LongInt get_luminosity( ): LongInt
vb    function get_luminosity( ) As Integer
cs    int get_luminosity( )
java  int get_luminosity( )
uwp   async Task<int> get_luminosity( )
py    get_luminosity( )
php   function get_luminosity( )
ts    async get_luminosity( ): Promise<number>
es    async get_luminosity( )
dnp   int get_luminosity( )
cp    int get_luminosity( )
cmd   YModule target get_luminosity
```

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

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

module→get_parentHub() YModule
module→parentHub()

Retourne le numéro de série du YoctoHub sur lequel est connecté le module.

```
js    function get_parentHub( )
cpp   string get_parentHub( )
m     -(NSString*) parentHub
pas   string get_parentHub( ): string
vb    function get_parentHub( ) As String
cs    string get_parentHub( )
java  String get_parentHub( )
uwp   async Task<string> get_parentHub( )
py    get_parentHub( )
php   function get_parentHub( )
ts    async get_parentHub( ): Promise<string>
es    async get_parentHub( )
dnp   string get_parentHub( )
cp    string get_parentHub( )
cmd   YModule target get_parentHub
```

Si le module est connecté par USB, ou si le module est le YoctoHub racine, une chaîne vide est retournée.

Retourne :
une chaîne de caractères contenant le numéro de série du YoctoHub, ou une chaîne vide.

module→get_persistentSettings() YModule
module→persistentSettings()

Retourne l'état courant des réglages persistents du module.

```
js    function get_persistentSettings( )
cpp   Y_PERSISTENTSETTINGS_enum get_persistentSettings( )
m     -(Y_PERSISTENTSETTINGS_enum) persistentSettings
pas   Integer get_persistentSettings( ): Integer
vb    function get_persistentSettings( ) As Integer
cs    int get_persistentSettings( )
java  int get_persistentSettings( )
uwp   async Task<int> get_persistentSettings( )
py    get_persistentSettings( )
php   function get_persistentSettings( )
ts    async get_persistentSettings( ): Promise<YModule_PersistentSettings>
es    async get_persistentSettings( )
dnp   int get_persistentSettings( )
cp    int get_persistentSettings( )
cmd   YModule target get_persistentSettings
```

Retourne :

une valeur parmi `YModule.PERSISTENTSETTINGS_LOADED`, `YModule.PERSISTENTSETTINGS_SAVED` et `YModule.PERSISTENTSETTINGS_MODIFIED` représentant l'état courant des réglages persistents du module

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

module→get_productId() YModule
module→productId()

Retourne l'identifiant USB du module, préprogrammé en usine.

```
js    function get_productId( )
cpp   int get_productId( )
m     -(int) productId
pas   LongInt get_productId( ): LongInt
vb    function get_productId( ) As Integer
cs    int get_productId( )
java  int get_productId( )
uwp   async Task<int> get_productId( )
py    get_productId( )
php   function get_productId( )
ts    async get_productId( ): Promise<number>
es    async get_productId( )
dnp   int get_productId( )
cp    int get_productId( )
cmd   YModule target get_productId
```

Retourne :
un entier représentant l'identifiant USB du module, préprogrammé en usine

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

module→get_productName() YModule
module→productName()

Retourne le nom commercial du module, préprogrammé en usine.

```
js    function get_productName( )
cpp   string get_productName( )
m     -(NSString*) productName
pas   string get_productName( ): string
vb    function get_productName( ) As String
cs    string get_productName( )
java  String get_productName( )
uwp   async Task<string> get_productName( )
py    get_productName( )
php   function get_productName( )
ts    async get_productName( ): Promise<string>
es    async get_productName( )
dnp   string get_productName( )
cp    string get_productName( )
cmd   YModule target get_productName
```

Retourne :
une chaîne de caractères représentant le nom commercial du module, préprogrammé en usine

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

module→get_productRelease() YModule
module→productRelease()

Retourne le numéro uméro de révision du module hardware, préprogrammé en usine.

```
js    function get_productRelease( )
cpp   int get_productRelease( )
m     -(int) productRelease
pas   LongInt get_productRelease( ): LongInt
vb    function get_productRelease( ) As Integer
cs    int get_productRelease( )
java  int get_productRelease( )
uwp   async Task<int> get_productRelease( )
py    get_productRelease( )
php   function get_productRelease( )
ts    async get_productRelease( ): Promise<number>
es    async get_productRelease( )
dnp   int get_productRelease( )
cp    int get_productRelease( )
cmd   YModule target get_productRelease
```

La révision originale du retourne la valeur 1, la révision B retourne la valeur 2, etc.

Retourne :
un entier représentant le numéro uméro de révision du module hardware, préprogrammé en usine

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

module→get_rebootCountdown() YModule
module→rebootCountdown()

Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

```
js      function get_rebootCountdown( )
cpp     int get_rebootCountdown( )
m       -(int) rebootCountdown
pas     LongInt get_rebootCountdown( ): LongInt
vb      function get_rebootCountdown( ) As Integer
cs      int get_rebootCountdown( )
java    int get_rebootCountdown( )
uwp     async Task<int> get_rebootCountdown( )
py      get_rebootCountdown( )
php     function get_rebootCountdown( )
ts      async get_rebootCountdown( ): Promise<number>
es      async get_rebootCountdown( )
dnp     int get_rebootCountdown( )
cp      int get_rebootCountdown( )
cmd     YModule target get_rebootCountdown
```

Retourne :

un entier représentant le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé

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

module→get_serialNumber() YModule
module→serialNumber()

Retourne le numéro de série du module, préprogrammé en usine.

```
js    function get_serialNumber( )
cpp   string get_serialNumber( )
m     -(NSString*) serialNumber
pas   string get_serialNumber( ): string
vb    function get_serialNumber( ) As String
cs    string get_serialNumber( )
java  String get_serialNumber( )
uwp   async Task<string> get_serialNumber( )
py    get_serialNumber( )
php   function get_serialNumber( )
ts    async get_serialNumber( ): Promise<string>
es    async get_serialNumber( )
dnp   string get_serialNumber( )
cp    string get_serialNumber( )
cmd   YModule target get_serialNumber
```

Retourne :
une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine

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

module→get_subDevices() YModule
module→subDevices()

Retourne la liste des modules branchés au module courant.

```
js      function get_subDevices( )
cpp     vector<string> get_subDevices( )
m       -(NSMutableArray*) subDevices
pas     TStringArray get_subDevices( ): TStringArray
vb      function get_subDevices( ) As List
cs      List<string> get_subDevices( )
java    ArrayList<String> get_subDevices( )
uwp     async Task<List<string>> get_subDevices( )
py      get_subDevices( )
php     function get_subDevices( )
ts      async get_subDevices( ): Promise<string[]
es      async get_subDevices( )
dnp     string[] get_subDevices( )
cp      vector<string> get_subDevices( )
cmd     YModule target get_subDevices
```

Cette fonction n'est pertinente que lorsqu'elle appelée pour un YoctoHub ou pour le VirtualHub. Dans le cas contraire, un tableau vide est retourné.

Retourne :
un tableau de chaînes de caractères contenant les numéros de série des sous-modules connectés au module

module→get_upTime() YModule
module→upTime()

Retourne le numbre de millisecondes écoulées depuis la mise sous tension du module

```
js    function get_upTime( )
cpp   s64 get_upTime( )
m     -(s64) upTime
pas   int64 get_upTime( ): int64
vb    function get_upTime( ) As Long
cs    long get_upTime( )
java  long get_upTime( )
uwp   async Task<long> get_upTime( )
py    get_upTime( )
php   function get_upTime( )
ts    async get_upTime( ): Promise<number>
es    async get_upTime( )
dnp   long get_upTime( )
cp    s64 get_upTime( )
cmd   YModule target get_upTime
```

Retourne :
un entier représentant le numbre de millisecondes écoulées depuis la mise sous tension du module

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

module→get_url() YModule
module→url()

Retourne l'URL utilisée pour accéder au module.

```
js      function get_url( )
cpp     string get_url( )
m       -(NSString*) url
pas     string get_url( ): string
vb      function get_url( ) As String
cs      string get_url( )
java    String get_url( )
uwp     async Task<string> get_url( )
py      get_url( )
php     function get_url( )
ts      async get_url( ): Promise<string>
es      async get_url( )
dnp     string get_url( )
cp      string get_url( )
cmd     YModule target get_url
```

Si le module est connecté par USB la chaîne de caractère 'usb' est retournée.

Retourne :
une chaîne de caractère contenant l'URL du module.

module→get_usbCurrent() YModule
module→usbCurrent()

Retourne le courant consommé par le module sur le bus USB, en milliampères.

```
js    function get_usbCurrent( )
cpp   int get_usbCurrent( )
m     -(int) usbCurrent
pas   LongInt get_usbCurrent( ): LongInt
vb    function get_usbCurrent( ) As Integer
cs    int get_usbCurrent( )
java  int get_usbCurrent( )
uwp   async Task<int> get_usbCurrent( )
py    get_usbCurrent( )
php   function get_usbCurrent( )
ts    async get_usbCurrent( ): Promise<number>
es    async get_usbCurrent( )
dnp   int get_usbCurrent( )
cp    int get_usbCurrent( )
cmd   YModule target get_usbCurrent
```

Retourne :
un entier représentant le courant consommé par le module sur le bus USB, en milliampères

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

module→get_userData() YModule
module→userData()

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

```
js      function get_userData( )
cpp     void * get_userData( )
m       -(id) userData
pas     Tobject get_userData( ): Tobject
vb      function get_userData( ) As Object
cs      object get_userData( )
java    Object get_userData( )
py      get_userData( )
php     function get_userData( )
ts      async get_userData( ): Promise<object|null>
es      async get_userData( )
```

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

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

module→get_userVar() YModule
module→userVar()

Retourne la valeur entière précédemment stockée dans cet attribut.

```
js    function get_userVar( )
cpp   int get_userVar( )
m     -(int) userVar
pas   LongInt get_userVar( ): LongInt
vb    function get_userVar( ) As Integer
cs    int get_userVar( )
java  int get_userVar( )
uwp   async Task<int> get_userVar( )
py    get_userVar( )
php   function get_userVar( )
ts    async get_userVar( ): Promise<number>
es    async get_userVar( )
dnp   int get_userVar( )
cp    int get_userVar( )
cmd   YModule target get_userVar
```

Au démarrage du module (ou après un redémarrage), la valeur est toujours zéro.

Retourne :
un entier représentant la valeur entière précédemment stockée dans cet attribut

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

module→hasFunction() YModule

Teste la présence d'une fonction pour le module courant.

```
js    function hasFunction( funcId)
cpp   bool hasFunction( string funcId)
m     -(bool) hasFunction : (NSString*) funcId
pas   boolean hasFunction( funcId: string): boolean
vb    function hasFunction( ByVal funcId As String) As Boolean
cs    bool hasFunction( string funcId)
java  boolean hasFunction( String funcId)
uwp   async Task<bool> hasFunction( string funcId)
py    hasFunction( funcId)
php   function hasFunction( $funcId)
ts    async hasFunction( funcId: string): Promise<boolean>
es    async hasFunction( funcId)
dnp   bool hasFunction( string funcId)
cp    bool hasFunction( string funcId)
cmd   YModule target hasFunction funcId
```

La méthode prend en paramètre l'identifiant de la fonction (relay1, voltage2,...) et retourne un booléen.

Paramètres :

funcId identifiant matériel de la fonction

Retourne :

vrai si le module inclut la fonction demandée

module→isOnline() YModule

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

js	function **isOnline**( )
cpp	bool **isOnline**( )
m	-(BOOL) **isOnline**
pas	boolean **isOnline**( ): boolean
vb	function **isOnline**( ) As Boolean
cs	bool **isOnline**( )
java	boolean **isOnline**( )
py	**isOnline**( )
php	function **isOnline**( )
ts	async **isOnline**( ): Promise<boolean>
es	async **isOnline**( )
dnp	bool **isOnline**( )
cp	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

module→isOnline_async() YModule

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

```
js function isOnline_async( callback, context)
```

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() YModule

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

js	function **load**(**msValidity**)
cpp	YRETCODE **load**(int **msValidity**)
m	-(YRETCODE) **load** : (u64) **msValidity**
pas	YRETCODE **load**(**msValidity**: u64): YRETCODE
vb	function **load**(ByVal **msValidity** As Long) As YRETCODE
cs	YRETCODE **load**(ulong **msValidity**)
java	int **load**(long **msValidity**)
py	**load**(**msValidity**)
php	function **load**(**$msValidity**)
ts	async **load**(**msValidity**: number): Promise<number>
es	async **load**(**msValidity**)

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

Paramètres :

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

module→load_async() YModule

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

```
js function load_async( msValidity, callback, context)
```

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

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→log() YModule

Ajoute un message arbitraire dans les logs du module.

```
js    function log( text)
cpp   int log( string text)
m     -(int) log : (NSString*) text
pas   LongInt log( text: string): LongInt
vb    function log( ByVal text As String) As Integer
cs    int log( string text)
java  int log( String text)
uwp   async Task<int> log( string text)
py    log( text)
php   function log( $text)
ts    async log( text: string): Promise<number>
es    async log( text)
dnp   int log( string text)
cp    int log( string text)
cmd   YModule target log text
```

Cette fonction est utile en particulier pour tracer l'exécution de callbacks HTTP. Si un saut de ligne est désiré après le message, il doit être inclus dans la chaîne de caractère.

Paramètres :

text le message à ajouter aux logs du module.

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

module→nextModule() YModule

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

js	function **nextModule**( )
cpp	YModule * **nextModule**( )
m	-(nullable YModule*) **nextModule**
pas	TYModule **nextModule**( ): TYModule
vb	function **nextModule**( ) As YModule
cs	YModule **nextModule**( )
java	YModule **nextModule**( )
uwp	YModule **nextModule**( )
py	**nextModule**( )
php	function **nextModule**( )
ts	**nextModule**( ): YModule | null
es	**nextModule**( )

Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les modules sont retournés. Si vous souhaitez retrouver un module spécifique, utilisez `Module.findModule()` avec un hardwareID ou un nom logique.

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

module→reboot() YModule

Agende un simple redémarrage du module dans un nombre donné de secondes.

```
js    function reboot( secBeforeReboot)
cpp   int reboot( int secBeforeReboot)
m     -(int) reboot : (int) secBeforeReboot
pas   LongInt reboot( secBeforeReboot: LongInt): LongInt
vb    function reboot( ByVal secBeforeReboot As Integer) As Integer
cs    int reboot( int secBeforeReboot)
java  int reboot( int secBeforeReboot)
uwp   async Task<int> reboot( int secBeforeReboot)
py    reboot( secBeforeReboot)
php   function reboot( $secBeforeReboot)
ts    async reboot( secBeforeReboot: number): Promise<number>
es    async reboot( secBeforeReboot)
dnp   int reboot( int secBeforeReboot)
cp    int reboot( int secBeforeReboot)
cmd   YModule target reboot secBeforeReboot
```

Paramètres :

secBeforeReboot nombre de secondes avant de redémarrer

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

module→registerBeaconCallback() YModule

Enregistre une fonction de callback qui sera appelée à chaque changement d'état de la balise de localisation du module.

```
js    function registerBeaconCallback( callback)
cpp   int registerBeaconCallback( YModuleBeaconCallback callback)
m     -(int) registerBeaconCallback : (YModuleBeaconCallback _Nullable) callback
pas   LongInt registerBeaconCallback( callback: TYModuleBeaconCallback): LongInt
vb    function registerBeaconCallback( ByVal callback As YModuleBeaconCallback) As Integer
cs    int registerBeaconCallback( BeaconCallback callback)
java  int registerBeaconCallback( BeaconCallback callback)
uwp   async Task<int> registerBeaconCallback( BeaconCallback callback)
py    registerBeaconCallback( callback)
php   function registerBeaconCallback( $callback)
ts    async registerBeaconCallback( callback: YModuleBeaconCallback | null): Promise<number>
es    async registerBeaconCallback( callback)
```

La fonction de callback doit accepter deux arguments: l'objet YModule dont la balise a changé, et un entier représentant l'état de la balise de localisation.

Paramètres :

callback la fonction de callback à rappeler, ou `null`

module→registerConfigChangeCallback() YModule

Enregistre une fonction de callback qui sera appelée à chaque fois qu'un réglage persistant d'un module est modifié (par exemple changement d'unité de mesure, etc.)

```
js    function registerConfigChangeCallback( callback)
cpp   int registerConfigChangeCallback( YModuleConfigChangeCallback callback)
m     -(int) registerConfigChangeCallback : (YModuleConfigChangeCallback _Nullable) callback
pas   LongInt registerConfigChangeCallback( callback: TYModuleConfigChangeCallback): LongInt
vb    function registerConfigChangeCallback( ByVal callback As YModuleConfigChangeCallback) As
      Integer
cs    int registerConfigChangeCallback( ConfigChangeCallback callback)
java  int registerConfigChangeCallback( ConfigChangeCallback callback)
uwp   async Task<int> registerConfigChangeCallback( ConfigChangeCallback callback)
py    registerConfigChangeCallback( callback)
php   function registerConfigChangeCallback( $callback)
ts    async registerConfigChangeCallback( callback: YModuleConfigChangeCallback | null):
      Promise<number>
es    async registerConfigChangeCallback( callback)
```

Paramètres :

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

module→registerLogCallback() YModule

Enregistre une fonction de callback qui sera appelée à chaque fois le module émet un message de log.

```
js    function registerLogCallback( callback)
cpp   int registerLogCallback( YModuleLogCallback callback)
m     -(int) registerLogCallback : (YModuleLogCallback _Nullable) callback
pas   LongInt registerLogCallback( callback: TYModuleLogCallback): LongInt
vb    function registerLogCallback( ByVal callback As YModuleLogCallback) As Integer
cs    int registerLogCallback( LogCallback callback)
java  int registerLogCallback( LogCallback callback)
uwp   async Task<int> registerLogCallback( LogCallback callback)
py    registerLogCallback( callback)
php   function registerLogCallback( $callback)
ts    async registerLogCallback( callback: YModuleLogCallback | null): Promise<number>
es    async registerLogCallback( callback)
```

Utile pour débugger le fonctionnement d'un module Yoctopuce.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet module qui a produit un log, un chaîne de caractère qui contiens le log

module→revertFromFlash() YModule

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

```
js    function revertFromFlash( )
cpp   int revertFromFlash( )
m     -(int) revertFromFlash
pas   LongInt revertFromFlash( ): LongInt
vb    function revertFromFlash( ) As Integer
cs    int revertFromFlash( )
java  int revertFromFlash( )
uwp   async Task<int> revertFromFlash( )
py    revertFromFlash( )
php   function revertFromFlash( )
ts    async revertFromFlash( ): Promise<number>
es    async revertFromFlash( )
dnp   int revertFromFlash( )
cp    int revertFromFlash( )
cmd   YModule target revertFromFlash
```

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

module→saveToFlash() YModule

Sauve les réglages courants dans la mémoire non volatile du module.

js	function **saveToFlash**( )
cpp	int **saveToFlash**( )
m	-(int) **saveToFlash**
pas	LongInt **saveToFlash**( ): LongInt
vb	function **saveToFlash**( ) As Integer
cs	int **saveToFlash**( )
java	int **saveToFlash**( )
uwp	async Task<int> **saveToFlash**( )
py	**saveToFlash**( )
php	function **saveToFlash**( )
ts	async **saveToFlash**( ): Promise<number>
es	async **saveToFlash**( )
dnp	int **saveToFlash**( )
cp	int **saveToFlash**( )
cmd	YModule **target saveToFlash**

Attention le nombre total de sauvegardes possibles durant la vie du module est limité (environ 100000 cycles). N'appelez pas cette fonction dans une boucle.

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

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

module→set_allSettings() YModule
module→setAllSettings()

Rétablit tous les paramètres du module.

```
js    function set_allSettings( settings)
cpp   int set_allSettings( string settings)
m     -(int) setAllSettings : (NSData*) settings
pas   LongInt set_allSettings( settings: TByteArray): LongInt
vb    procedure set_allSettings( ByVal settings As Byte()
cs    int set_allSettings( byte[] settings)
java  int set_allSettings( byte[] settings)
uwp   async Task<int> set_allSettings( byte[] settings)
py    set_allSettings( settings)
php   function set_allSettings( $settings)
ts    async set_allSettings( settings: Uint8Array): Promise<number>
es    async set_allSettings( settings)
dnp   int set_allSettings( byte[] settings)
cp    int set_allSettings( string settings)
cmd   YModule target set_allSettings settings
```

Utile pour restorer les noms logiques et les calibrations du module depuis une sauvgarde. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si les réglages doivent être préservés.

Paramètres :

settings un objet binaire avec touts les paramètres

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

module→set_allSettingsAndFiles() YModule
module→setAllSettingsAndFiles()

Rétablit tous les paramètres de configuration et fichiers sur un module.

```
js    function set_allSettingsAndFiles( settings)
cpp   int set_allSettingsAndFiles( string settings)
m     -(int) setAllSettingsAndFiles : (NSData*) settings
pas   LongInt set_allSettingsAndFiles( settings: TByteArray): LongInt
vb    procedure set_allSettingsAndFiles( ByVal settings As Byte()
cs    int set_allSettingsAndFiles( byte[] settings)
java  int set_allSettingsAndFiles( byte[] settings)
uwp   async Task<int> set_allSettingsAndFiles( byte[] settings)
py    set_allSettingsAndFiles( settings)
php   function set_allSettingsAndFiles( $settings)
ts    async set_allSettingsAndFiles( settings: Uint8Array): Promise<number>
es    async set_allSettingsAndFiles( settings)
dnp   int set_allSettingsAndFiles( byte[] settings)
cp    int set_allSettingsAndFiles( string settings)
cmd   YModule target set_allSettingsAndFiles settings
```

Cette méthode est utile pour récupérer les noms logiques, les calibrations, les fichiers uploadés, etc. du module depuis une sauvgarde. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si les réglages doivent être préservés.

Paramètres :

settings un buffer binaire avec touts les paramètres

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

module→set_beacon() YModule
module→setBeacon()

Allume ou éteint la balise de localisation du module.

```
js      function set_beacon( newval)
cpp     int set_beacon( Y_BEACON_enum newval)
m       -(int) setBeacon : (Y_BEACON_enum) newval
pas     integer set_beacon( newval: Integer): integer
vb      function set_beacon( ByVal newval As Integer) As Integer
cs      int set_beacon( int newval)
java    int set_beacon( int newval)
uwp     async Task<int> set_beacon( int newval)
py      set_beacon( newval)
php     function set_beacon( $newval)
ts      async set_beacon( newval: YModule_Beacon): Promise<number>
es      async set_beacon( newval)
dnp     int set_beacon( int newval)
cp      int set_beacon( int newval)
cmd     YModule target set_beacon newval
```

Paramètres :

newval soit `YModule.BEACON_OFF`, soit `YModule.BEACON_ON`

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

module→set_logicalName() YModule
module→setLogicalName()

Change le nom logique du module.

```
js    function set_logicalName( newval)
cpp   int set_logicalName( string newval)
m     -(int) setLogicalName : (NSString*) newval
pas   integer set_logicalName( newval: string): integer
vb    function set_logicalName( ByVal newval As String) As Integer
cs    int set_logicalName( string newval)
java  int set_logicalName( String newval)
uwp   async Task<int> set_logicalName( string newval)
py    set_logicalName( newval)
php   function set_logicalName( $newval)
ts    async set_logicalName( newval: string): Promise<number>
es    async set_logicalName( newval)
dnp   int set_logicalName( string newval)
cp    int set_logicalName( string newval)
cmd   YModule target set_logicalName newval
```

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

Paramètres :

newval une chaîne de caractères

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

module→set_luminosity() YModule
module→setLuminosity()

Modifie la luminosité des leds informatives du module.

```
js    function set_luminosity( newval)
cpp   int set_luminosity( int newval)
m     -(int) setLuminosity : (int) newval
pas   integer set_luminosity( newval: LongInt): integer
vb    function set_luminosity( ByVal newval As Integer) As Integer
cs    int set_luminosity( int newval)
java  int set_luminosity( int newval)
uwp   async Task<int> set_luminosity( int newval)
py    set_luminosity( newval)
php   function set_luminosity( $newval)
ts    async set_luminosity( newval: number): Promise<number>
es    async set_luminosity( newval)
dnp   int set_luminosity( int newval)
cp    int set_luminosity( int newval)
cmd   YModule target set_luminosity newval
```

Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la luminosité des leds informatives du module

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

module→set_userData() YModule
module→setUserData()

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

```
js    function set_userData( data)
cpp   void set_userData( void * data)
m     -(void) setUserData : (id) data
pas   set_userData( data: Tobject)
vb    procedure set_userData( ByVal data As Object)
cs    void set_userData( object data)
java  void set_userData( Object data)
py    set_userData( data)
php   function set_userData( $data)
ts    async set_userData( data: object|null): Promise<void>
es    async set_userData( data)
```

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

Paramètres :

data objet quelconque à mémoriser

module→set_userVar() YModule
module→setUserVar()

Stocke une valeur 32 bits dans la mémoire volatile du module.

```
js    function set_userVar( newval)
cpp   int set_userVar( int newval)
m     -(int) setUserVar : (int) newval
pas   integer set_userVar( newval: LongInt): integer
vb    function set_userVar( ByVal newval As Integer) As Integer
cs    int set_userVar( int newval)
java  int set_userVar( int newval)
uwp   async Task<int> set_userVar( int newval)
py    set_userVar( newval)
php   function set_userVar( $newval)
ts    async set_userVar( newval: number): Promise<number>
es    async set_userVar( newval)
dnp   int set_userVar( int newval)
cp    int set_userVar( int newval)
cmd   YModule target set_userVar newval
```

Cet attribut est à la disposition du programmeur pour y stocker par exemple une variable d'état. Au démarrage du module (ou après un redémarrage), la valeur est toujours zéro.

Paramètres :

newval un entier

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

module→triggerConfigChangeCallback() YModule

Force le déclanchement d'un callback de changement de configuration, afin de vérifier si ils sont disponibles ou pas.

js	function **triggerConfigChangeCallback**( )
cpp	int **triggerConfigChangeCallback**( )
m	-(int) **triggerConfigChangeCallback**
pas	LongInt **triggerConfigChangeCallback**( ): LongInt
vb	function **triggerConfigChangeCallback**( ) As Integer
cs	int **triggerConfigChangeCallback**( )
java	int **triggerConfigChangeCallback**( )
uwp	async Task<int> **triggerConfigChangeCallback**( )
py	**triggerConfigChangeCallback**( )
php	function **triggerConfigChangeCallback**( )
ts	async **triggerConfigChangeCallback**( ): Promise<number>
es	async **triggerConfigChangeCallback**( )
dnp	int **triggerConfigChangeCallback**( )
cp	int **triggerConfigChangeCallback**( )
cmd	YModule **target triggerConfigChangeCallback**

module→triggerFirmwareUpdate() YModule

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

```
js    function triggerFirmwareUpdate( secBeforeReboot)
cpp   int triggerFirmwareUpdate( int secBeforeReboot)
m     -(int) triggerFirmwareUpdate : (int) secBeforeReboot
pas   LongInt triggerFirmwareUpdate( secBeforeReboot: LongInt): LongInt
vb    function triggerFirmwareUpdate( ByVal secBeforeReboot As Integer) As Integer
cs    int triggerFirmwareUpdate( int secBeforeReboot)
java  int triggerFirmwareUpdate( int secBeforeReboot)
uwp   async Task<int> triggerFirmwareUpdate( int secBeforeReboot)
py    triggerFirmwareUpdate( secBeforeReboot)
php   function triggerFirmwareUpdate( $secBeforeReboot)
ts    async triggerFirmwareUpdate( secBeforeReboot: number): Promise<number>
es    async triggerFirmwareUpdate( secBeforeReboot)
dnp   int triggerFirmwareUpdate( int secBeforeReboot)
cp    int triggerFirmwareUpdate( int secBeforeReboot)
cmd   YModule target triggerFirmwareUpdate secBeforeReboot
```

Paramètres :

secBeforeReboot nombre de secondes avant de redémarrer

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

module→updateFirmware() YModule

Prepare une mise à jour de firmware du module.

```
js      function updateFirmware( path)
cpp     YFirmwareUpdate updateFirmware( string path)
m       -(YFirmwareUpdate*) updateFirmware : (NSString*) path
pas     TYFirmwareUpdate updateFirmware( path: string): TYFirmwareUpdate
vb      function updateFirmware( ByVal path As String) As YFirmwareUpdate
cs      YFirmwareUpdate updateFirmware( string path)
java    YFirmwareUpdate updateFirmware( String path)
uwp     async Task<YFirmwareUpdate> updateFirmware( string path)
py      updateFirmware( path)
php     function updateFirmware( $path)
ts      async updateFirmware( path: string): Promise<YFirmwareUpdate>
es      async updateFirmware( path)
dnp     YFirmwareUpdateProxy updateFirmware( string path)
cp      YFirmwareUpdateProxy* updateFirmware( string path)
cmd     YModule target updateFirmware path
```

Cette méthode retourne un object `YFirmwareUpdate` qui est utilisé pour mettre à jour le firmware du module.

Paramètres :

path le path du fichier `.byn` à utiliser

Retourne :

un object `YFirmwareUpdate` ou NULL en cas d'erreur

module→updateFirmwareEx() YModule

Prepare une mise à jour de firmware du module.

```
js    function updateFirmwareEx( path, force)
cpp   YFirmwareUpdate updateFirmwareEx( string path, bool force)
m     -(YFirmwareUpdate*) updateFirmwareEx : (NSString*) path
                                           : (bool) force
pas   TYFirmwareUpdate updateFirmwareEx( path: string, force: boolean): TYFirmwareUpdate
vb    function updateFirmwareEx( ByVal path As String,
                                 ByVal force As Boolean) As YFirmwareUpdate
cs    YFirmwareUpdate updateFirmwareEx( string path, bool force)
java  YFirmwareUpdate updateFirmwareEx( String path, boolean force)
uwp   async Task<YFirmwareUpdate> updateFirmwareEx( string path, bool force)
py    updateFirmwareEx( path, force)
php   function updateFirmwareEx( $path, $force)
ts    async updateFirmwareEx( path: string, force: boolean): Promise<YFirmwareUpdate>
es    async updateFirmwareEx( path, force)
dnp   YFirmwareUpdateProxy updateFirmwareEx( string path, bool force)
cp    YFirmwareUpdateProxy* updateFirmwareEx( string path,
                                              bool force)
cmd   YModule target updateFirmwareEx path force
```

Cette méthode retourne un object `YFirmwareUpdate` qui est utilisé pour mettre à jour le firmware du module.

Paramètres :

path le path du fichier `.byn` à utiliser

force vrai pour forceer la mise à jour même si un prérequis ne semble pas satisfait

Retourne :

un object `YFirmwareUpdate` ou NULL en cas d'erreur

module→wait_async() YModule

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

```
js  function wait_async( callback, context)
ts  wait_async( callback: Function, context: object)
es  wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

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

Retourne :

rien du tout.

24.3. La classe YMotor

Interface pour intéragir avec les moteurs, disponibles par exemple dans le Yocto-Motor-DC

La classe `YMotor` permet de piloter un moteur DC. Elle permet de configurer la puissance envoyée au moteur pour le faire tourner aussi bien dans un sens que dans l'autre, mais aussi de piloter des accélérations linéaires: le moteur accélére alors tout seul sans que vous vous ayez à vous en occuper. La librairie permet aussi de freiner le moteur: cela est réalisé en court-circuitant les pôles du moteur, ce qui le transforme en frein électro-magnétique.

Pour utiliser les fonctions décrites ici, vous devez inclure:

es	in HTML: <script src="../../lib/yocto_motor.js"></script> in node.js: require('yoctolib-es2017/yocto_motor.js');
js	<script type='text/javascript' src='yocto_motor.js'></script>
cpp	#include "yocto_motor.h"
m	#import "yocto_motor.h"
pas	uses yocto_motor;
vb	yocto_motor.vb
cs	yocto_motor.cs
java	import com.yoctopuce.YoctoAPI.YMotor;
uwp	import com.yoctopuce.YoctoAPI.YMotor;
py	from yocto_motor import *
php	require_once('yocto_motor.php');
ts	in HTML: import { YMotor } from '../../dist/esm/yocto_motor.js'; in Node.js: import { YMotor } from 'yoctolib-cjs/yocto_motor.js';
dnp	import YoctoProxyAPI.YMotorProxy
cp	#include "yocto_motor_proxy.h"
vi	YMotor.vi
ml	import YoctoProxyAPI.YMotorProxy

Fonction globales

YMotor.FindMotor(**func**)

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

YMotor.FindMotorInContext(**yctx**, **func**)

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

YMotor.FirstMotor()

Commence l'énumération des moteurs accessibles par la librairie.

YMotor.FirstMotorInContext(**yctx**)

Commence l'énumération des moteurs accessibles par la librairie.

YMotor.GetSimilarFunctions()

Enumère toutes les fonctions de type Motor disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

Propriétés des objets `YMotorProxy`

motor→AdvertisedValue *[lecture seule]*

Courte chaîne de caractères représentant l'état courant de la fonction.

motor→CutOffVoltage *[modifiable]*

Limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

motor→FailSafeTimeout *[modifiable]*

Temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

motor→Frequency *[modifiable]*

Fréquence du signal PWM utilisé pour contrôler le moteur.

motor→FriendlyName *[lecture seule]*

Identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

motor→FunctionId *[lecture seule]*

Identifiant matériel du moteur, sans référence au module.

motor→HardwareId *[lecture seule]*

Identifiant matériel unique de la fonction au format `SERIAL.FUNCTIONID`.

motor→IsOnline *[lecture seule]*

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

motor→LogicalName *[modifiable]*

Nom logique de la fonction.

motor→MotorStatus *[modifiable]*

état du contrôleur de moteur.

motor→OverCurrentLimit *[modifiable]*

Valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

motor→SerialNumber *[lecture seule]*

Numéro de série du module, préprogrammé en usine.

motor→StarterTime *[modifiable]*

Durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

Méthodes des objets `YMotor`

motor→brakingForceMove(targetPower, delay)

Modifie progressivement la force de freinage appliquée au moteur sur une durée donnée.

motor→clearCache()

Invalide le cache.

motor→describe()

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

motor→drivingForceMove(targetPower, delay)

Modifie progressivement la puissance envoyée au moteur sur une durée donnée.

motor→get_advertisedValue()

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

motor→get_brakingForce()

Retourne la force de freinage appliquée au moteur, sous forme de pourcentage.

motor→get_cutOffVoltage()

Retourne la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

motor→get_drivingForce()

Retourne la puissance actuelle envoyée au moteur, sous forme de nombre réel entre -100% et +100%.

motor→get_errorMessage()

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

motor→get_errorType()

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

motor→get_failSafeTimeout()

Retourne le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

motor→get_frequency()

Retourne la fréquence du signal PWM utilisé pour contrôler le moteur.

motor→get_friendlyName()

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

motor→get_functionDescriptor()

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

motor→get_functionId()

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

motor→get_hardwareId()

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

motor→get_logicalName()

Retourne le nom logique du moteur.

motor→get_module()

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

motor→get_module_async(callback, context)

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

motor→get_motorStatus()

Retourne l'état du contrôleur de moteur.

motor→get_overCurrentLimit()

Retourne la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

motor→get_serialNumber()

Retourne le numéro de série du module, préprogrammé en usine.

motor→get_starterTime()

Retourne la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

motor→get_userData()

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

motor→isOnline()

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

motor→isOnline_async(callback, context)

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

motor→isReadOnly()

Test si la fonction est en lecture seule.

motor→keepALive()

Réarme la sécurité failsafe du contrôleur.

motor→load(msValidity)

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

motor→loadAttribute(attrName)

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

motor→load_async(msValidity, callback, context)

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

motor→muteValueCallbacks()

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

motor→nextMotor()

Continue l'énumération des moteurs commencée à l'aide de `yFirstMotor()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les moteurs sont retournés.

motor→registerValueCallback(callback)

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

motor→resetStatus()

Réinitialise l'état du contrôleur à IDLE.

motor→set_brakingForce(newval)

Modifie immédiatement la force de freinage appliquée au moteur (en pourcents).

motor→set_cutOffVoltage(newval)

Modifie la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

motor→set_drivingForce(newval)

Modifie immédiatement la puissance envoyée au moteur.

motor→set_failSafeTimeout(newval)

Modifie le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

motor→set_frequency(newval)

Modifie la fréquence du signal PWM utilisée pour contrôler le moteur.

motor→set_logicalName(newval)

Modifie le nom logique du moteur.

motor→set_overCurrentLimit(newval)

Modifie la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

motor→set_starterTime(newval)

Modifie la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

motor→set_userData(data)

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

motor→unmuteValueCallbacks()

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

motor→wait_async(callback, context)

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

YMotor.FindMotor() YMotor
YMotor.FindMotor()

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

```
js    function yFindMotor( func)
cpp   YMotor* FindMotor( string func)
m     +(YMotor*) FindMotor : (NSString*) func
pas   TYMotor yFindMotor( func: string): TYMotor
vb    function FindMotor( ByVal func As String) As YMotor
cs    static YMotor FindMotor( string func)
java  static YMotor FindMotor( String func)
uwp   static YMotor FindMotor( string func)
py    FindMotor( func)
php   function FindMotor( $func)
ts    static FindMotor( func: string): YMotor
es    static FindMotor( func)
dnp   static YMotorProxy FindMotor( string func)
cp    static YMotorProxy * FindMotor( string func)
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

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

Paramètres :

func une chaîne de caractères qui référence le moteur sans ambiguïté, par exemple `MOTORCTL.motor`.

Retourne :

un objet de classe `YMotor` qui permet ensuite de contrôler le moteur.

YMotor.FindMotorInContext() YMotor
YMotor.FindMotorInContext()

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

```
java  static YMotor FindMotorInContext( YAPIContext yctx, String func)
uwp   static YMotor FindMotorInContext( YAPIContext yctx, string func)
ts    static FindMotorInContext( yctx: YAPIContext, func: string): YMotor
es    static FindMotorInContext( yctx, func)
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence le moteur sans ambiguïté, par exemple `MOTORCTL.motor`.

Retourne :

un objet de classe `YMotor` qui permet ensuite de contrôler le moteur.

YMotor.FirstMotor() YMotor

YMotor.FirstMotor()

Commence l'énumération des moteurs accessibles par la librairie.

js	function **yFirstMotor**( )
cpp	YMotor * **FirstMotor**( )
m	+(YMotor*) **FirstMotor**
pas	TYMotor **yFirstMotor**( ): TYMotor
vb	function **FirstMotor**( ) As YMotor
cs	static YMotor **FirstMotor**( )
java	static YMotor **FirstMotor**( )
uwp	static YMotor **FirstMotor**( )
py	**FirstMotor**( )
php	function **FirstMotor**( )
ts	static **FirstMotor**( ): YMotor | null
es	static **FirstMotor**( )

Utiliser la fonction `YMotor.nextMotor()` pour itérer sur les autres moteurs.

Retourne :

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

YMotor.FirstMotorInContext() YMotor

YMotor.FirstMotorInContext()

Commence l'énumération des moteurs accessibles par la librairie.

java	static YMotor **FirstMotorInContext**(YAPIContext **yctx**)
uwp	static YMotor **FirstMotorInContext**(YAPIContext **yctx**)
ts	static **FirstMotorInContext**(**yctx**: YAPIContext): YMotor | null
es	static **FirstMotorInContext**(**yctx**)

Utiliser la fonction `YMotor.nextMotor()` pour itérer sur les autres moteurs.

Paramètres :

yctx un contexte YAPI.

Retourne :

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

YMotor.GetSimilarFunctions() YMotor

YMotor.GetSimilarFunctions()

Enumère toutes les fonctions de type Motor disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

dnp static new string[] **GetSimilarFunctions**( )

cp static vector<string> **GetSimilarFunctions**( )

Chaque chaîne retournée peut être passée en argument à la méthode `YMotor.FindMotor` pour obtenir une objet permettant d'intéragir avec le module correspondant.

Retourne :

un tableau de chaînes de caractères, contenant les identifiants matériels de chaque fonction disponible trouvée.

motor→AdvertisedValue YMotor

Courte chaîne de caractères représentant l'état courant de la fonction.

dnp string **AdvertisedValue**

motor→CutOffVoltage YMotor

Limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

dnp double **CutOffVoltage**

Ce réglage permet d'éviter d'endommager un accumulateur un continuant à l'utiliser une fois "vide".

Modifiable. Ce réglage permet d'éviter d'endommager un accumulateur un continuant à l'utiliser une fois "vide". Attention, quel que soit le réglage du cutoff, le variateur passera en erreur si l'alimentation passe (même brièvement) en dessous de 3V. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

motor→FailSafeTimeout YMotor

Temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

dnp int **FailSafeTimeout**

Passé ce delai, le contrôleur arrêtera le moteur et passera en mode erreur FAILSAFE. La sécurité failsafe est désactivée quand la valeur est à zéro.

Modifiable. Passé ce delai, le contrôleur arrêtera le moteur et passera en mode erreur FAILSAFE. La sécurité failsafe est désactivée quand la valeur est à zéro. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

motor→Frequency YMotor

Fréquence du signal PWM utilisé pour contrôler le moteur.

dnp double **Frequency**

Modifiable. Modifie la fréquence du signal PWM utilisée pour contrôler le moteur. Une fréquence basse est généralement plus efficace (les composant chauffent moins et le moteur démarre plus facilement), mais un bruit audible peut être généré. Une fréquence élevée peut réduire le bruit, mais il y a plus d'énergie perdue en chaleur. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

motor→FriendlyName YMotor

Identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

dnp string **FriendlyName**

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

motor→FunctionId YMotor

Identifiant matériel du moteur, sans référence au module.

dnp string **FunctionId**

Par example `relay1`.

motor→HardwareId YMotor

Identifiant matériel unique de la fonction au format `SERIAL.FUNCTIONID`.

dnp string **HardwareId**

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

motor→IsOnline YMotor

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

dnp 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.

motor→LogicalName YMotor

Nom logique de la fonction.

dnp string **LogicalName**

Modifiable. 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.

motor→MotorStatus YMotor

état du contrôleur de moteur.

dnp int **MotorStatus**

Les états possibles sont: IDLE si le moteur est à l'arrêt/en roue libre, prêt à démarrer; FORWD si le contrôleur fait tourner le moteur en marche avant; BACKWD si le contrôleur fait tourner le moteur en marche arrière; BRAKE si le contrôleur est en train de freiner; LOVOLT si le contrôleur a détecté une tension trop basse; HICURR si le contrôleur a détecté une surconsommation; HIHEAT si le contrôleur a détecté une surchauffe; FAILSF si le contrôleur est passé en protection failsafe.

Si le contrôleur est en erreur (LOVOLT, HICURR, HIHEAT,FAILSF), il doit être explicitement réinitialisé avec la fonction `resetStatus`.

motor→OverCurrentLimit YMotor

Valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

dnp int **OverCurrentLimit**

Une valeur nulle signifie qu'aucune limite n'est définie.

Modifiable. Une valeur nulle signifie qu'aucune limite n'est définie. Attention, quel que soit le réglage choisi, le variateur passera en erreur si le courant passe, même brièvement, en dessus de 32A. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

motor→SerialNumber YMotor

Numéro de série du module, préprogrammé en usine.

dnp string **SerialNumber**

motor→StarterTime YMotor

Durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

dnp int **StarterTime**

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

motor→brakingForceMove() YMotor

Modifie progressivement la force de freinage appliquée au moteur sur une durée donnée.

```
js    function brakingForceMove( targetPower, delay)
cpp   int brakingForceMove( double targetPower, int delay)
m     -(int) brakingForceMove : (double) targetPower
                              : (int) delay
pas   LongInt brakingForceMove( targetPower: double, delay: LongInt): LongInt
vb    function brakingForceMove( ByVal targetPower As Double,
                                 ByVal delay As Integer) As Integer
cs    int brakingForceMove( double targetPower, int delay)
java  int brakingForceMove( double targetPower, int delay)
uwp   async Task<int> brakingForceMove( double targetPower, int delay)
py    brakingForceMove( targetPower, delay)
php   function brakingForceMove( $targetPower, $delay)
ts    async brakingForceMove( targetPower: number, delay: number): Promise<number>
es    async brakingForceMove( targetPower, delay)
dnp   int brakingForceMove( double targetPower, int delay)
cp    int brakingForceMove( double targetPower, int delay)
cmd   YMotor target brakingForceMove targetPower delay
```

Paramètres :

targetPower force de freinage finale, en pourcentage

delay durée (en ms) sur laquelle le changement de puissance sera effectué

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

motor→clearCache() YMotor

Invalide le cache.

js	function **clearCache**( )
cpp	void **clearCache**( )
m	-(void) **clearCache**
pas	**clearCache**( )
vb	procedure **clearCache**( )
cs	void **clearCache**( )
java	void **clearCache**( )
py	**clearCache**( )
php	function **clearCache**( )
ts	async **clearCache**( ): Promise<void>
es	async **clearCache**( )

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

motor→describe() YMotor

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

```
js    function describe( )
cpp   string describe( )
m     -(NSString*) describe
pas   string describe( ): string
vb    function describe( ) As String
cs    string describe( )
java  String describe( )
py    describe( )
php   function describe( )
ts    async describe( ): Promise<string>
es    async describe( )
```

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

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

motor→drivingForceMove() YMotor

Modifie progressivement la puissance envoyée au moteur sur une durée donnée.

```
js    function drivingForceMove( targetPower, delay)
cpp   int drivingForceMove( double targetPower, int delay)
m     -(int) drivingForceMove : (double) targetPower
                              : (int) delay
pas   LongInt drivingForceMove( targetPower: double, delay: LongInt): LongInt
vb    function drivingForceMove( ByVal targetPower As Double,
                                 ByVal delay As Integer) As Integer
cs    int drivingForceMove( double targetPower, int delay)
java  int drivingForceMove( double targetPower, int delay)
uwp   async Task<int> drivingForceMove( double targetPower, int delay)
py    drivingForceMove( targetPower, delay)
php   function drivingForceMove( $targetPower, $delay)
ts    async drivingForceMove( targetPower: number, delay: number): Promise<number>
es    async drivingForceMove( targetPower, delay)
dnp   int drivingForceMove( double targetPower, int delay)
cp    int drivingForceMove( double targetPower, int delay)
cmd   YMotor target drivingForceMove targetPower delay
```

Paramètres :

targetPower puissance finale désirée, en pourcentage de -100% à +100%

delay durée (en ms) sur laquelle le changement de puissance sera effectué

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

motor→get_advertisedValue() YMotor
motor→advertisedValue()

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

```
js    function get_advertisedValue( )
cpp   string get_advertisedValue( )
m     -(NSString*) advertisedValue
pas   string get_advertisedValue( ): string
vb    function get_advertisedValue( ) As String
cs    string get_advertisedValue( )
java  String get_advertisedValue( )
uwp   async Task<string> get_advertisedValue( )
py    get_advertisedValue( )
php   function get_advertisedValue( )
ts    async get_advertisedValue( ): Promise<string>
es    async get_advertisedValue( )
dnp   string get_advertisedValue( )
cp    string get_advertisedValue( )
cmd   YMotor target get_advertisedValue
```

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

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

motor→get_brakingForce() YMotor
motor→brakingForce()

Retourne la force de freinage appliquée au moteur, sous forme de pourcentage.

```
js    function get_brakingForce( )
cpp   double get_brakingForce( )
m     -(double) brakingForce
pas   double get_brakingForce( ): double
vb    function get_brakingForce( ) As Double
cs    double get_brakingForce( )
java  double get_brakingForce( )
uwp   async Task<double> get_brakingForce( )
py    get_brakingForce( )
php   function get_brakingForce( )
ts    async get_brakingForce( ): Promise<number>
es    async get_brakingForce( )
dnp   double get_brakingForce( )
cp    double get_brakingForce( )
cmd   YMotor target get_brakingForce
```

La valeur 0 correspond ne pas freiner (moteur en roue libre).

Retourne :
une valeur numérique représentant la force de freinage appliquée au moteur, sous forme de pourcentage

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

motor→get_cutOffVoltage() YMotor
motor→cutOffVoltage()

Retourne la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

js	function **get_cutOffVoltage**( )
cpp	double **get_cutOffVoltage**( )
m	-(double) cutOffVoltage
pas	double **get_cutOffVoltage**( ): double
vb	function **get_cutOffVoltage**( ) As Double
cs	double **get_cutOffVoltage**( )
java	double **get_cutOffVoltage**( )
uwp	async Task<double> **get_cutOffVoltage**( )
py	**get_cutOffVoltage**( )
php	function **get_cutOffVoltage**( )
ts	async **get_cutOffVoltage**( ): Promise<number>
es	async **get_cutOffVoltage**( )
dnp	double **get_cutOffVoltage**( )
cp	double **get_cutOffVoltage**( )
cmd	YMotor **target** **get_cutOffVoltage**

Ce réglage permet d'éviter d'endommager un accumulateur un continuant à l'utiliser une fois "vide".

Retourne :
une valeur numérique représentant la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation

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

motor→get_drivingForce() YMotor
motor→drivingForce()

Retourne la puissance actuelle envoyée au moteur, sous forme de nombre réel entre -100% et +100%.

js	function **get_drivingForce**( )
cpp	double **get_drivingForce**( )
m	-(double) drivingForce
pas	double **get_drivingForce**( ): double
vb	function **get_drivingForce**( ) As Double
cs	double **get_drivingForce**( )
java	double **get_drivingForce**( )
uwp	async Task<double> **get_drivingForce**( )
py	**get_drivingForce**( )
php	function **get_drivingForce**( )
ts	async **get_drivingForce**( ): Promise<number>
es	async **get_drivingForce**( )
dnp	double **get_drivingForce**( )
cp	double **get_drivingForce**( )
cmd	YMotor **target get_drivingForce**

Retourne :
une valeur numérique représentant la puissance actuelle envoyée au moteur, sous forme de nombre réel entre -100% et +100%

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

motor→get_errorMessage() YMotor
motor→errorMessage()

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

```
js    function get_errorMessage( )
cpp   string get_errorMessage( )
m     -(NSString*) errorMessage
pas   string get_errorMessage( ): string
vb    function get_errorMessage( ) As String
cs    string get_errorMessage( )
java  String get_errorMessage( )
py    get_errorMessage( )
php   function get_errorMessage( )
ts    get_errorMessage( ): string
es    get_errorMessage( )
```

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

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

motor→get_errorType() YMotor
motor→errorType()

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

js	function **get_errorType**( )
cpp	YRETCODE **get_errorType**( )
m	-(YRETCODE) errorType
pas	YRETCODE **get_errorType**( ): YRETCODE
vb	function **get_errorType**( ) As YRETCODE
cs	YRETCODE **get_errorType**( )
java	int **get_errorType**( )
py	**get_errorType**( )
php	function **get_errorType**( )
ts	**get_errorType**( ): number
es	**get_errorType**( )

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

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

motor→get_failSafeTimeout() YMotor
motor→failSafeTimeout()

Retourne le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

js	function **get_failSafeTimeout**( )
cpp	int **get_failSafeTimeout**( )
m	-(int) failSafeTimeout
pas	LongInt **get_failSafeTimeout**( ): LongInt
vb	function **get_failSafeTimeout**( ) As Integer
cs	int **get_failSafeTimeout**( )
java	int **get_failSafeTimeout**( )
uwp	async Task<int> **get_failSafeTimeout**( )
py	**get_failSafeTimeout**( )
php	function **get_failSafeTimeout**( )
ts	async **get_failSafeTimeout**( ): Promise<number>
es	async **get_failSafeTimeout**( )
dnp	int **get_failSafeTimeout**( )
cp	int **get_failSafeTimeout**( )
cmd	YMotor **target** **get_failSafeTimeout**

Passé ce delai, le contrôleur arrêtera le moteur et passera en mode erreur FAILSAFE. La sécurité failsafe est désactivée quand la valeur est à zéro.

Retourne :
un entier représentant le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle

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

motor→get_frequency() YMotor
motor→frequency()

Retourne la fréquence du signal PWM utilisé pour contrôler le moteur.

```
js    function get_frequency( )
cpp   double get_frequency( )
m     -(double) frequency
pas   double get_frequency( ): double
vb    function get_frequency( ) As Double
cs    double get_frequency( )
java  double get_frequency( )
uwp   async Task<double> get_frequency( )
py    get_frequency( )
php   function get_frequency( )
ts    async get_frequency( ): Promise<number>
es    async get_frequency( )
dnp   double get_frequency( )
cp    double get_frequency( )
cmd   YMotor target get_frequency
```

Retourne :
une valeur numérique représentant la fréquence du signal PWM utilisé pour contrôler le moteur

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

motor→get_friendlyName() YMotor
motor→friendlyName()

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

```
js    function get_friendlyName( )
cpp   string get_friendlyName( )
m     -(NSString*) friendlyName
cs    string get_friendlyName( )
java  String get_friendlyName( )
py    get_friendlyName( )
php   function get_friendlyName( )
ts    async get_friendlyName( ): Promise<string>
es    async get_friendlyName( )
dnp   string get_friendlyName( )
cp    string get_friendlyName( )
```

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

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

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

motor→get_functionDescriptor() YMotor
motor→functionDescriptor()

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

js	function **get_functionDescriptor**( )
cpp	YFUN_DESCR **get_functionDescriptor**( )
m	-(YFUN_DESCR) functionDescriptor
pas	YFUN_DESCR **get_functionDescriptor**( ): YFUN_DESCR
vb	function **get_functionDescriptor**( ) As YFUN_DESCR
cs	YFUN_DESCR **get_functionDescriptor**( )
java	String **get_functionDescriptor**( )
py	**get_functionDescriptor**( )
php	function **get_functionDescriptor**( )
ts	async **get_functionDescriptor**( ): Promise<string>
es	async **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$CLASSNAME$.FUNCTIONDESCRIPTOR_INVALID

motor→get_functionId() YMotor
motor→functionId()

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

```
js    function get_functionId( )
cpp   string get_functionId( )
m     -(NSString*) functionId
vb    function get_functionId( ) As String
cs    string get_functionId( )
java  String get_functionId( )
py    get_functionId( )
php   function get_functionId( )
ts    async get_functionId( ): Promise<string>
es    async get_functionId( )
dnp   string get_functionId( )
cp    string get_functionId( )
```

Par example `relay1`.

Retourne :
une chaîne de caractères identifiant le moteur (ex: `relay1`)

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

motor→get_hardwareId() YMotor
motor→hardwareId()

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

js	function **get_hardwareId**( )
cpp	string **get_hardwareId**( )
m	-(NSString*) hardwareId
vb	function **get_hardwareId**( ) As String
cs	string **get_hardwareId**( )
java	String **get_hardwareId**( )
py	**get_hardwareId**( )
php	function **get_hardwareId**( )
ts	async **get_hardwareId**( ): Promise<string>
es	async **get_hardwareId**( )
dnp	string **get_hardwareId**( )
cp	string **get_hardwareId**( )

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

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

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

motor→get_logicalName() YMotor
motor→logicalName()

Retourne le nom logique du moteur.

```
js    function get_logicalName( )
cpp   string get_logicalName( )
m     -(NSString*) logicalName
pas   string get_logicalName( ): string
vb    function get_logicalName( ) As String
cs    string get_logicalName( )
java  String get_logicalName( )
uwp   async Task<string> get_logicalName( )
py    get_logicalName( )
php   function get_logicalName( )
ts    async get_logicalName( ): Promise<string>
es    async get_logicalName( )
dnp   string get_logicalName( )
cp    string get_logicalName( )
cmd   YMotor target get_logicalName
```

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

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

motor→get_module() YMotor
motor→module()

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

js	function **get_module**( )
cpp	YModule * **get_module**( )
m	-(YModule*) module
pas	TYModule **get_module**( ): TYModule
vb	function **get_module**( ) As YModule
cs	YModule **get_module**( )
java	YModule **get_module**( )
py	**get_module**( )
php	function **get_module**( )
ts	async **get_module**( ): Promise<YModule>
es	async **get_module**( )
dnp	YModuleProxy **get_module**( )
cp	YModuleProxy * **get_module**( )

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

Retourne :
une instance de `YModule`

motor→get_module_async() YMotor
motor→module_async()

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

```
js  function get_module_async( callback, context)
```

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.

motor→get_motorStatus() YMotor
motor→motorStatus()

Retourne l'état du contrôleur de moteur.

```
js    function get_motorStatus( )
cpp   Y_MOTORSTATUS_enum get_motorStatus( )
m     -(Y_MOTORSTATUS_enum) motorStatus
pas   Integer get_motorStatus( ): Integer
vb    function get_motorStatus( ) As Integer
cs    int get_motorStatus( )
java  int get_motorStatus( )
uwp   async Task<int> get_motorStatus( )
py    get_motorStatus( )
php   function get_motorStatus( )
ts    async get_motorStatus( ): Promise<YMotor_MotorStatus>
es    async get_motorStatus( )
dnp   int get_motorStatus( )
cp    int get_motorStatus( )
cmd   YMotor target get_motorStatus
```

Les états possibles sont: IDLE si le moteur est à l'arrêt/en roue libre, prêt à démarrer; FORWD si le contrôleur fait tourner le moteur en marche avant; BACKWD si le contrôleur fait tourner le moteur en marche arrière; BRAKE si le contrôleur est en train de freiner; LOVOLT si le contrôleur a détecté une tension trop basse; HICURR si le contrôleur a détecté une surconsommation; HIHEAT si le contrôleur a détecté une surchauffe; FAILSF si le contrôleur est passé en protection failsafe.

Si le contrôleur est en erreur (LOVOLT, HICURR, HIHEAT,FAILSF), il doit être explicitement réinitialisé avec la fonction `resetStatus`.

Retourne :

une valeur parmi `YMotor.MOTORSTATUS_IDLE`, `YMotor.MOTORSTATUS_BRAKE`, `YMotor.MOTORSTATUS_FORWD`, `YMotor.MOTORSTATUS_BACKWD`, `YMotor.MOTORSTATUS_LOVOLT`, `YMotor.MOTORSTATUS_HICURR`, `YMotor.MOTORSTATUS_HIHEAT` et `YMotor.MOTORSTATUS_FAILSF` représentant l'état du contrôleur de moteur

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

motor→get_overCurrentLimit() YMotor
motor→overCurrentLimit()

Retourne la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

js	function **get_overCurrentLimit**( )
cpp	int **get_overCurrentLimit**( )
m	-(int) overCurrentLimit
pas	LongInt **get_overCurrentLimit**( ): LongInt
vb	function **get_overCurrentLimit**( ) As Integer
cs	int **get_overCurrentLimit**( )
java	int **get_overCurrentLimit**( )
uwp	async Task<int> **get_overCurrentLimit**( )
py	**get_overCurrentLimit**( )
php	function **get_overCurrentLimit**( )
ts	async **get_overCurrentLimit**( ): Promise<number>
es	async **get_overCurrentLimit**( )
dnp	int **get_overCurrentLimit**( )
cp	int **get_overCurrentLimit**( )
cmd	YMotor **target** **get_overCurrentLimit**

Une valeur nulle signifie qu'aucune limite n'est définie.

Retourne :
un entier représentant la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur

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

motor→get_serialNumber() YMotor
motor→serialNumber()

Retourne le numéro de série du module, préprogrammé en usine.

```
js    function get_serialNumber( )
cpp   string get_serialNumber( )
m     -(NSString*) serialNumber
pas   string get_serialNumber( ): string
vb    function get_serialNumber( ) As String
cs    string get_serialNumber( )
java  String get_serialNumber( )
uwp   async Task<string> get_serialNumber( )
py    get_serialNumber( )
php   function get_serialNumber( )
ts    async get_serialNumber( ): Promise<string>
es    async get_serialNumber( )
dnp   string get_serialNumber( )
cp    string get_serialNumber( )
cmd   YMotor target get_serialNumber
```

Retourne :
: une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine.

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

motor→get_starterTime() YMotor
motor→starterTime()

Retourne la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

js	function **get_starterTime**( )
cpp	int **get_starterTime**( )
m	-(int) starterTime
pas	LongInt **get_starterTime**( ): LongInt
vb	function **get_starterTime**( ) As Integer
cs	int **get_starterTime**( )
java	int **get_starterTime**( )
uwp	async Task<int> **get_starterTime**( )
py	**get_starterTime**( )
php	function **get_starterTime**( )
ts	async **get_starterTime**( ): Promise<number>
es	async **get_starterTime**( )
dnp	int **get_starterTime**( )
cp	int **get_starterTime**( )
cmd	YMotor **target** **get_starterTime**

Retourne :

un entier représentant la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage

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

motor→get_userData() YMotor
motor→userData()

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

js	function **get_userData**( )
cpp	void * **get_userData**( )
m	-(id) userData
pas	Tobject **get_userData**( ): Tobject
vb	function **get_userData**( ) As Object
cs	object **get_userData**( )
java	Object **get_userData**( )
py	**get_userData**( )
php	function **get_userData**( )
ts	async **get_userData**( ): Promise<object|null>
es	async **get_userData**( )

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

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

motor→isOnline() YMotor

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

js	function **isOnline**( )
cpp	bool **isOnline**( )
m	-(BOOL) **isOnline**
pas	boolean **isOnline**( ): boolean
vb	function **isOnline**( ) As Boolean
cs	bool **isOnline**( )
java	boolean **isOnline**( )
py	**isOnline**( )
php	function **isOnline**( )
ts	async **isOnline**( ): Promise<boolean>
es	async **isOnline**( )
dnp	bool **isOnline**( )
cp	bool **isOnline**( )

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

Retourne :
`true` si le moteur est joignable, `false` sinon

motor→isOnline_async() YMotor

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

```
js function isOnline_async( callback, context)
```

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

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 machine virtuelle Javascript avec une attente active.

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.

motor→isReadOnly() YMotor

Test si la fonction est en lecture seule.

cpp	bool **isReadOnly**( )
m	-(bool) **isReadOnly**
pas	boolean **isReadOnly**( ): boolean
vb	function **isReadOnly**( ) As Boolean
cs	bool **isReadOnly**( )
java	boolean **isReadOnly**( )
uwp	async Task<bool> **isReadOnly**( )
py	**isReadOnly**( )
php	function **isReadOnly**( )
ts	async **isReadOnly**( ): Promise<boolean>
es	async **isReadOnly**( )
dnp	bool **isReadOnly**( )
cp	bool **isReadOnly**( )
cmd	YMotor **target isReadOnly**

Retourne vrais si la fonction est protégé en ecriture ou que la fontion n'est pas disponible.

Retourne :
`true` si la fonction est protégé en ecriture ou que la fontion n'est pas disponible

motor→keepALive() YMotor

Réarme la sécurité failsafe du contrôleur.

js	function **keepALive**( )
cpp	int **keepALive**( )
m	-(int) **keepALive**
pas	LongInt **keepALive**( ): LongInt
vb	function **keepALive**( ) As Integer
cs	int **keepALive**( )
java	int **keepALive**( )
uwp	async Task<int> **keepALive**( )
py	**keepALive**( )
php	function **keepALive**( )
ts	async **keepALive**( ): Promise<number>
es	async **keepALive**( )
dnp	int **keepALive**( )
cp	int **keepALive**( )
cmd	YMotor **target keepALive**

Lorsque le moteur est en marche et que la sécurité failsafe est activée, cette fonction doit être appelée périodiquement pour confirmer le bon fonctionnement du processus de contrôle. A défaut, le moteur s'arrêtera automatiquement au bout du temps prévu. Notez que l'appel à une fonction de type *set* du moteur réarme aussi la sécurité failsafe.

motor→load() YMotor

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

js	function **load**(**msValidity**)
cpp	YRETCODE **load**(int **msValidity**)
m	-(YRETCODE) **load** : (u64) **msValidity**
pas	YRETCODE **load**(**msValidity**: u64): YRETCODE
vb	function **load**(ByVal **msValidity** As Long) As YRETCODE
cs	YRETCODE **load**(ulong **msValidity**)
java	int **load**(long **msValidity**)
py	**load**(**msValidity**)
php	function **load**(**$msValidity**)
ts	async **load**(**msValidity**: number): Promise<number>
es	async **load**(**msValidity**)

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

Paramètres :

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

motor→loadAttribute() YMotor

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

```
js    function loadAttribute( attrName)
cpp   string loadAttribute( string attrName)
m     -(NSString*) loadAttribute : (NSString*) attrName
pas   string loadAttribute( attrName: string): string
vb    function loadAttribute( ByVal attrName As String) As String
cs    string loadAttribute( string attrName)
java  String loadAttribute( String attrName)
uwp   async Task<string> loadAttribute( string attrName)
py    loadAttribute( attrName)
php   function loadAttribute( $attrName)
ts    async loadAttribute( attrName: string): Promise<string>
es    async loadAttribute( attrName)
dnp   string loadAttribute( string attrName)
cp    string loadAttribute( string attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

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

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

motor→load_async() YMotor

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

```
js function load_async( msValidity, callback, context)
```

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 machine virtuelle Javascript avec une attente active.

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.

motor→muteValueCallbacks() YMotor

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

js	function **muteValueCallbacks**( )
cpp	int **muteValueCallbacks**( )
m	-(int) **muteValueCallbacks**
pas	LongInt **muteValueCallbacks**( ): LongInt
vb	function **muteValueCallbacks**( ) As Integer
cs	int **muteValueCallbacks**( )
java	int **muteValueCallbacks**( )
uwp	async Task<int> **muteValueCallbacks**( )
py	**muteValueCallbacks**( )
php	function **muteValueCallbacks**( )
ts	async **muteValueCallbacks**( ): Promise<number>
es	async **muteValueCallbacks**( )
dnp	int **muteValueCallbacks**( )
cp	int **muteValueCallbacks**( )
cmd	YMotor **target muteValueCallbacks**

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

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

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

motor→nextMotor() YMotor

Continue l'énumération des moteurs commencée à l'aide de `yFirstMotor()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les moteurs sont retournés.

js	function **nextMotor**( )
cpp	YMotor * **nextMotor**( )
m	-(nullable YMotor*) **nextMotor**
pas	TYMotor **nextMotor**( ): TYMotor
vb	function **nextMotor**( ) As YMotor
cs	YMotor **nextMotor**( )
java	YMotor **nextMotor**( )
uwp	YMotor **nextMotor**( )
py	**nextMotor**( )
php	function **nextMotor**( )
ts	**nextMotor**( ): YMotor | null
es	**nextMotor**( )

Si vous souhaitez retrouver un moteur spécifique, utilisez `Motor.findMotor()` avec un hardwareID ou un nom logique.

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

motor→registerValueCallback() YMotor

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

js	function **registerValueCallback**(**callback**)
cpp	int **registerValueCallback**(YMotorValueCallback **callback**)
m	-(int) **registerValueCallback** : (YMotorValueCallback _Nullable) **callback**
pas	LongInt **registerValueCallback**(**callback**: TYMotorValueCallback): LongInt
vb	function **registerValueCallback**(ByVal **callback** As YMotorValueCallback) As Integer
cs	int **registerValueCallback**(ValueCallback **callback**)
java	int **registerValueCallback**(UpdateCallback **callback**)
uwp	async Task<int> **registerValueCallback**(ValueCallback **callback**)
py	**registerValueCallback**(**callback**)
php	function **registerValueCallback**(**$callback**)
ts	async **registerValueCallback**(**callback**: YMotorValueCallback | null): Promise<number>
es	async **registerValueCallback**(**callback**)

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

Paramètres :

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

motor→resetStatus() YMotor

Réinitialise l'état du contrôleur à IDLE.

js	function **resetStatus**( )
cpp	int **resetStatus**( )
m	-(int) **resetStatus**
pas	LongInt **resetStatus**( ): LongInt
vb	function **resetStatus**( ) As Integer
cs	int **resetStatus**( )
java	int **resetStatus**( )
uwp	async Task<int> **resetStatus**( )
py	**resetStatus**( )
php	function **resetStatus**( )
ts	async **resetStatus**( ): Promise<number>
es	async **resetStatus**( )
dnp	int **resetStatus**( )
cp	int **resetStatus**( )
cmd	YMotor **target resetStatus**

Cette fonction doit être explicitement appelée après toute condition d'erreur pour permettre au contrôleur de repartir.

motor→set_brakingForce() YMotor
motor→setBrakingForce()

Modifie immédiatement la force de freinage appliquée au moteur (en pourcents).

```
js    function set_brakingForce( newval)
cpp   int set_brakingForce( double newval)
m     -(int) setBrakingForce : (double) newval
pas   integer set_brakingForce( newval: double): integer
vb    function set_brakingForce( ByVal newval As Double) As Integer
cs    int set_brakingForce( double newval)
java  int set_brakingForce( double newval)
uwp   async Task<int> set_brakingForce( double newval)
py    set_brakingForce( newval)
php   function set_brakingForce( $newval)
ts    async set_brakingForce( newval: number): Promise<number>
es    async set_brakingForce( newval)
dnp   int set_brakingForce( double newval)
cp    int set_brakingForce( double newval)
cmd   YMotor target set_brakingForce newval
```

La valeur 0 correspond à ne pas freiner (moteur en roue libre). Lorsque la force de freinage est changée, la puissance de traction est remise à zéro.

Paramètres :

newval une valeur numérique représentant immédiatement la force de freinage appliquée au moteur (en pourcents)

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

motor→set_cutOffVoltage() YMotor
motor→setCutOffVoltage()

Modifie la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation.

js	function **set_cutOffVoltage**(**newval**)
cpp	int **set_cutOffVoltage**(double **newval**)
m	-(int) setCutOffVoltage : (double) **newval**
pas	integer **set_cutOffVoltage**(**newval**: double): integer
vb	function **set_cutOffVoltage**(ByVal **newval** As Double) As Integer
cs	int **set_cutOffVoltage**(double **newval**)
java	int **set_cutOffVoltage**(double **newval**)
uwp	async Task<int> **set_cutOffVoltage**(double **newval**)
py	**set_cutOffVoltage**(**newval**)
php	function **set_cutOffVoltage**(**$newval**)
ts	async **set_cutOffVoltage**(**newval**: number): Promise<number>
es	async **set_cutOffVoltage**(**newval**)
dnp	int **set_cutOffVoltage**(double **newval**)
cp	int **set_cutOffVoltage**(double **newval**)
cmd	YMotor **target** **set_cutOffVoltage** **newval**

Ce réglage permet d'éviter d'endommager un accumulateur un continuant à l'utiliser une fois "vide". Attention, quel que soit le réglage du cutoff, le variateur passera en erreur si l'alimentation passe (même brièvement) en dessous de 3V. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur numérique représentant la limite de l'alimentation en dessous de laquelle le contrôleur va automatiquement se mettre en erreur et couper la consommation

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

motor→set_drivingForce() YMotor
motor→setDrivingForce()

Modifie immédiatement la puissance envoyée au moteur.

```
js    function set_drivingForce( newval)
cpp   int set_drivingForce( double newval)
m     -(int) setDrivingForce : (double) newval
pas   integer set_drivingForce( newval: double): integer
vb    function set_drivingForce( ByVal newval As Double) As Integer
cs    int set_drivingForce( double newval)
java  int set_drivingForce( double newval)
uwp   async Task<int> set_drivingForce( double newval)
py    set_drivingForce( newval)
php   function set_drivingForce( $newval)
ts    async set_drivingForce( newval: number): Promise<number>
es    async set_drivingForce( newval)
dnp   int set_drivingForce( double newval)
cp    int set_drivingForce( double newval)
cmd   YMotor target set_drivingForce newval
```

La valeur est donnée en pourcentage de -100% à +100%. Si vous voulez ménager votre mécanique et éviter d'induire des consommations excessives qui pourraient dépasser les capacités du contrôleur, évitez les changements de régime trop brusques. Par exemple, passer brutalement de marche avant à marche arrière est une très mauvaise idée. A chaque fois que la puissance envoyée au moteur est changée, le freinage est remis à zéro.

Paramètres :

newval une valeur numérique représentant immédiatement la puissance envoyée au moteur

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

motor→set_failSafeTimeout() motor→setFailSafeTimeout() YMotor

Modifie le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle.

```
js    function set_failSafeTimeout( newval)
cpp   int set_failSafeTimeout( int newval)
m     -(int) setFailSafeTimeout : (int) newval
pas   integer set_failSafeTimeout( newval: LongInt): integer
vb    function set_failSafeTimeout( ByVal newval As Integer) As Integer
cs    int set_failSafeTimeout( int newval)
java  int set_failSafeTimeout( int newval)
uwp   async Task<int> set_failSafeTimeout( int newval)
py    set_failSafeTimeout( newval)
php   function set_failSafeTimeout( $newval)
ts    async set_failSafeTimeout( newval: number): Promise<number>
es    async set_failSafeTimeout( newval)
dnp   int set_failSafeTimeout( int newval)
cp    int set_failSafeTimeout( int newval)
cmd   YMotor target set_failSafeTimeout newval
```

Passé ce delai, le contrôleur arrêtera le moteur et passera en mode erreur FAILSAFE. La sécurité failsafe est désactivée quand la valeur est à zéro. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant le temps en millisecondes pendant lequel le variateur pourra fonctionner sans instruction du processus de contrôle

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

motor→set_frequency() YMotor
motor→setFrequency()

Modifie la fréquence du signal PWM utilisée pour contrôler le moteur.

```
js    function set_frequency( newval)
cpp   int set_frequency( double newval)
m     -(int) setFrequency : (double) newval
pas   integer set_frequency( newval: double): integer
vb    function set_frequency( ByVal newval As Double) As Integer
cs    int set_frequency( double newval)
java  int set_frequency( double newval)
uwp   async Task<int> set_frequency( double newval)
py    set_frequency( newval)
php   function set_frequency( $newval)
ts    async set_frequency( newval: number): Promise<number>
es    async set_frequency( newval)
dnp   int set_frequency( double newval)
cp    int set_frequency( double newval)
cmd   YMotor target set_frequency newval
```

Une fréquence basse est généralement plus efficace (les composant chauffent moins et le moteur démarre plus facilement), mais un bruit audible peut être généré. Une fréquence élevée peut réduire le bruit, mais il y a plus d'énergie perdue en chaleur. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur numérique représentant la fréquence du signal PWM utilisée pour contrôler le moteur

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

motor→set_logicalName() motor→setLogicalName() YMotor

Modifie le nom logique du moteur.

```
js    function set_logicalName( newval)
cpp   int set_logicalName( string newval)
m     -(int) setLogicalName : (NSString*) newval
pas   integer set_logicalName( newval: string): integer
vb    function set_logicalName( ByVal newval As String) As Integer
cs    int set_logicalName( string newval)
java  int set_logicalName( String newval)
uwp   async Task<int> set_logicalName( string newval)
py    set_logicalName( newval)
php   function set_logicalName( $newval)
ts    async set_logicalName( newval: string): Promise<number>
es    async set_logicalName( newval)
dnp   int set_logicalName( string newval)
cp    int set_logicalName( string newval)
cmd   YMotor target set_logicalName newval
```

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

Paramètres :

newval une chaîne de caractères représentant le nom logique du moteur.

Retourne :

`YAPI.SUCCESS` si l'appel se déroule sans erreur.

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

motor→set_overCurrentLimit() YMotor
motor→setOverCurrentLimit()

Modifie la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur.

js	function **set_overCurrentLimit**(**newval**)
cpp	int **set_overCurrentLimit**(int **newval**)
m	-(int) setOverCurrentLimit : (int) **newval**
pas	integer **set_overCurrentLimit**(**newval**: LongInt): integer
vb	function **set_overCurrentLimit**(ByVal **newval** As Integer) As Integer
cs	int **set_overCurrentLimit**(int **newval**)
java	int **set_overCurrentLimit**(int **newval**)
uwp	async Task<int> **set_overCurrentLimit**(int **newval**)
py	**set_overCurrentLimit**(**newval**)
php	function **set_overCurrentLimit**(**$newval**)
ts	async **set_overCurrentLimit**(**newval**: number): Promise<number>
es	async **set_overCurrentLimit**(**newval**)
dnp	int **set_overCurrentLimit**(int **newval**)
cp	int **set_overCurrentLimit**(int **newval**)
cmd	YMotor **target set_overCurrentLimit newval**

Une valeur nulle signifie qu'aucune limite n'est définie. Attention, quel que soit le réglage choisi, le variateur passera en erreur si le courant passe, même brièvement, en dessus de 32A. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la valeur limite du courant (en mA) au dessus de laquelle le contrôleur va automatiquement se mettre en erreur

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

motor→set_starterTime() YMotor
motor→setStarterTime()

Modifie la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage.

js	function **set_starterTime**(**newval**)
cpp	int **set_starterTime**(int **newval**)
m	-(int) setStarterTime : (int) **newval**
pas	integer **set_starterTime**(**newval**: LongInt): integer
vb	function **set_starterTime**(ByVal **newval** As Integer) As Integer
cs	int **set_starterTime**(int **newval**)
java	int **set_starterTime**(int **newval**)
uwp	async Task<int> **set_starterTime**(int **newval**)
py	**set_starterTime**(**newval**)
php	function **set_starterTime**(**$newval**)
ts	async **set_starterTime**(**newval**: number): Promise<number>
es	async **set_starterTime**(**newval**)
dnp	int **set_starterTime**(int **newval**)
cp	int **set_starterTime**(int **newval**)
cmd	YMotor **target set_starterTime newval**

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

Paramètres :

newval un entier représentant la durée (en ms) pendant laquelle le moteur est piloté à basse fréquence pour faciliter son démarrage

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

motor→set_userData() YMotor
motor→setUserData()

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

```
js    function set_userData( data)
cpp   void set_userData( void * data)
m     -(void) setUserData : (id) data
pas   set_userData( data: Tobject)
vb    procedure set_userData( ByVal data As Object)
cs    void set_userData( object data)
java  void set_userData( Object data)
py    set_userData( data)
php   function set_userData( $data)
ts    async set_userData( data: object|null): Promise<void>
es    async set_userData( data)
```

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

Paramètres :

data objet quelconque à mémoriser

motor→unmuteValueCallbacks() YMotor

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

js	function **unmuteValueCallbacks**( )
cpp	int **unmuteValueCallbacks**( )
m	-(int) **unmuteValueCallbacks**
pas	LongInt **unmuteValueCallbacks**( ): LongInt
vb	function **unmuteValueCallbacks**( ) As Integer
cs	int **unmuteValueCallbacks**( )
java	int **unmuteValueCallbacks**( )
uwp	async Task<int> **unmuteValueCallbacks**( )
py	**unmuteValueCallbacks**( )
php	function **unmuteValueCallbacks**( )
ts	async **unmuteValueCallbacks**( ): Promise<number>
es	async **unmuteValueCallbacks**( )
dnp	int **unmuteValueCallbacks**( )
cp	int **unmuteValueCallbacks**( )
cmd	YMotor **target** **unmuteValueCallbacks**

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

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

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

motor→wait_async() YMotor

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

```
js  function wait_async( callback, context)
ts  wait_async( callback: Function, context: object)
es  wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

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

Retourne :

rien du tout.

24.4. La classe YVoltage

Interface pour intéragir avec les capteurs de tension, disponibles par exemple dans le Yocto-Motor-DC, le Yocto-Volt et le Yocto-Watt

La classe `YVoltage` permet de lire et de configurer les capteurs de tension Yoctopuce. Elle hérite de la classe `YSensor` toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

es	in HTML: <script src="../../lib/yocto_voltage.js"></script> in node.js: require('yoctolib-es2017/yocto_voltage.js');
js	<script type='text/javascript' src='yocto_voltage.js'></script>
cpp	#include "yocto_voltage.h"
m	#import "yocto_voltage.h"
pas	uses yocto_voltage;
vb	yocto_voltage.vb
cs	yocto_voltage.cs
java	import com.yoctopuce.YoctoAPI.YVoltage;
uwp	import com.yoctopuce.YoctoAPI.YVoltage;
py	from yocto_voltage import *
php	require_once('yocto_voltage.php');
ts	in HTML: import { YVoltage } from '../../dist/esm/yocto_voltage.js'; in Node.js: import { YVoltage } from 'yoctolib-cjs/yocto_voltage.js';
dnp	import YoctoProxyAPI.YVoltageProxy
cp	#include "yocto_voltage_proxy.h"
vi	YVoltage.vi
ml	import YoctoProxyAPI.YVoltageProxy

Fonction globales

YVoltage.FindVoltage(**func**)

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

YVoltage.FindVoltageInContext(**yctx**, **func**)

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

YVoltage.FirstVoltage()

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

YVoltage.FirstVoltageInContext(**yctx**)

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

YVoltage.GetSimilarFunctions()

Enumère toutes les fonctions de type Voltage disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

Propriétés des objets `YVoltageProxy`

voltage→AdvMode *[modifiable]*

Mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

voltage→AdvertisedValue *[lecture seule]*

Courte chaîne de caractères représentant l'état courant de la fonction.

voltage→Enabled *[modifiable]*

état d'activation de cette mesure.

voltage→FriendlyName *[lecture seule]*

Identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

voltage→FunctionId *[lecture seule]*

Identifiant matériel du senseur, sans référence au module.

voltage→HardwareId *[lecture seule]*

Identifiant matériel unique de la fonction au format `SERIAL.FUNCTIONID`.

voltage→IsOnline *[lecture seule]*

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

voltage→LogFrequency *[modifiable]*

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

voltage→LogicalName *[modifiable]*

Nom logique de la fonction.

voltage→ReportFrequency *[modifiable]*

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

voltage→Resolution *[modifiable]*

Résolution des valeurs mesurées.

voltage→SerialNumber *[lecture seule]*

Numéro de série du module, préprogrammé en usine.

Méthodes des objets `YVoltage`

voltage→calibrateFromPoints(rawValues, refValues)

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

voltage→clearCache()

Invalide le cache.

voltage→describe()

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

voltage→get_advMode()

Retourne le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

voltage→get_advertisedValue()

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

voltage→get_currentRawValue()

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

voltage→get_currentValue()

Retourne la valeur actuelle de la tension, en Volt, sous forme de nombre à virgule.

voltage→get_dataLogger()

Retourne l'objet `YDatalogger` du module qui héberge le senseur.

voltage→get_enabled()

Retourne l'état d'activation de cette mesure.

voltage→get_errorMessage()

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

voltage→get_errorType()

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

voltage→get_friendlyName()

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

voltage→get_functionDescriptor()

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

voltage→get_functionId()

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

voltage→get_hardwareId()

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

voltage→get_highestValue()

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

voltage→get_logFrequency()

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

voltage→get_logicalName()

Retourne le nom logique du capteur de tension.

voltage→get_lowestValue()

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

voltage→get_module()

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

voltage→get_module_async(callback, context)

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

voltage→get_recordedData(startTime, endTime)

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

voltage→get_reportFrequency()

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

voltage→get_resolution()

Retourne la résolution des valeurs mesurées.

voltage→get_sensorState()

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

voltage→get_serialNumber()

Retourne le numéro de série du module, préprogrammé en usine.

voltage→get_unit()

Retourne l'unité dans laquelle la tension est exprimée.

voltage→get_userData()

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

voltage→isOnline()

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

voltage→isOnline_async(callback, context)

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

voltage→isReadOnly()

Test si la fonction est en lecture seule.

voltage→isSensorReady()

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

voltage→load(msValidity)

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

voltage→loadAttribute(attrName)

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

voltage→loadCalibrationPoints(rawValues, refValues)

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

voltage→load_async(msValidity, callback, context)

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

voltage→muteValueCallbacks()

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

voltage→nextVoltage()

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstVoltage()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les capteurs de tension sont retournés.

voltage→registerTimedReportCallback(callback)

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

voltage→registerValueCallback(callback)

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

voltage→set_advMode(newval)

Modifie le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

voltage→set_enabled(newval)

Modifie l'état d'activation de le mesure de tension.

voltage→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

voltage→set_logFrequency(newval)

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

voltage→set_logicalName(newval)

Modifie le nom logique du capteur de tension.

voltage→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

voltage→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

voltage→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

voltage→set_userData(data)

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

voltage→startDataLogger()

Démarre l'enregistreur de données du module.

voltage→stopDataLogger()

Arrête l'enregistreur de données du module.

voltage→unmuteValueCallbacks()

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

voltage→wait_async(callback, context)

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

YVoltage.FindVoltage() YVoltage
YVoltage.FindVoltage()

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

```
js    function yFindVoltage( func)
cpp   YVoltage* FindVoltage( string func)
m     +(YVoltage*) FindVoltage : (NSString*) func
pas   TYVoltage yFindVoltage( func: string): TYVoltage
vb    function FindVoltage( ByVal func As String) As YVoltage
cs    static YVoltage FindVoltage( string func)
java  static YVoltage FindVoltage( String func)
uwp   static YVoltage FindVoltage( string func)
py    FindVoltage( func)
php   function FindVoltage( $func)
ts    static FindVoltage( func: string): YVoltage
es    static FindVoltage( func)
dnp   static YVoltageProxy FindVoltage( string func)
cp    static YVoltageProxy * FindVoltage( string func)
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

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

Paramètres :

func une chaîne de caractères qui référence le capteur de tension sans ambiguïté, par exemple `MOTORCTL.voltage`.

Retourne :

un objet de classe `YVoltage` qui permet ensuite de contrôler le capteur de tension.

YVoltage.FindVoltageInContext() YVoltage

YVoltage.FindVoltageInContext()

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

```
java  static YVoltage FindVoltageInContext( YAPIContext yctx, String func)
uwp   static YVoltage FindVoltageInContext( YAPIContext yctx, string func)
ts    static FindVoltageInContext( yctx: YAPIContext, func: string): YVoltage
es    static FindVoltageInContext( yctx, func)
```

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de tension sans ambiguïté, par exemple `MOTORCTL.voltage`.

Retourne :

un objet de classe `YVoltage` qui permet ensuite de contrôler le capteur de tension.

YVoltage.FirstVoltage() YVoltage

YVoltage.FirstVoltage()

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

js	function **yFirstVoltage**( )
cpp	YVoltage * **FirstVoltage**( )
m	+(YVoltage*) **FirstVoltage**
pas	TYVoltage **yFirstVoltage**( ): TYVoltage
vb	function **FirstVoltage**( ) As YVoltage
cs	static YVoltage **FirstVoltage**( )
java	static YVoltage **FirstVoltage**( )
uwp	static YVoltage **FirstVoltage**( )
py	**FirstVoltage**( )
php	function **FirstVoltage**( )
ts	static **FirstVoltage**( ): YVoltage | null
es	static **FirstVoltage**( )

Utiliser la fonction `YVoltage.nextVoltage()` pour itérer sur les autres capteurs de tension.

Retourne :

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

YVoltage.FirstVoltageInContext() YVoltage
YVoltage.FirstVoltageInContext()

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

java	static YVoltage **FirstVoltageInContext**(YAPIContext **yctx**)
uwp	static YVoltage **FirstVoltageInContext**(YAPIContext **yctx**)
ts	static **FirstVoltageInContext**(**yctx**: YAPIContext): YVoltage | null
es	static **FirstVoltageInContext**(**yctx**)

Utiliser la fonction `YVoltage.nextVoltage()` pour itérer sur les autres capteurs de tension.

Paramètres :

yctx un contexte YAPI.

Retourne :

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

YVoltage.GetSimilarFunctions() YVoltage

YVoltage.GetSimilarFunctions()

Enumère toutes les fonctions de type Voltage disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

dnp static new string[] **GetSimilarFunctions**( )

cp static vector<string> **GetSimilarFunctions**( )

Chaque chaîne retournée peut être passée en argument à la méthode `YVoltage.FindVoltage` pour obtenir une objet permettant d'intéragir avec le module correspondant.

Retourne :

un tableau de chaînes de caractères, contenant les identifiants matériels de chaque fonction disponible trouvée.

voltage→AdvMode YVoltage

Mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

dnp int **AdvMode**

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

voltage→AdvertisedValue YVoltage

Courte chaîne de caractères représentant l'état courant de la fonction.

dnp string **AdvertisedValue**

voltage→Enabled YVoltage

état d'activation de cette mesure.

dnp int **Enabled**

Modifiable. Modifie l'état d'activation de le mesure de tension. Lorsque la mesure de tension AC est désactivée, la mesure se fera uniquement en mode DC, et inversément. Lorsque les deux mesures AC et DC sont activées, le module bascule entre le mode AC et DC en fonction de l'amplitude relative des variations AC par rapport à la tension moyenne. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

voltage→FriendlyName YVoltage

Identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

dnp string **FriendlyName**

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

voltage→FunctionId YVoltage

Identifiant matériel du senseur, sans référence au module.

dnp string **FunctionId**

Par example `relay1`.

voltage→HardwareId YVoltage

Identifiant matériel unique de la fonction au format `SERIAL.FUNCTIONID`.

dnp string **HardwareId**

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

voltage→IsOnline YVoltage

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

dnp 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.

voltage→LogFrequency YVoltage

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

dnp string **LogFrequency**

Modifiable. Modifie la fréquence d'enregistrement des mesures dans le datalogger. La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF". Attention il est inutile, voir contre productif, de régler la fréquence d'enregistrement à une valeur supérieure à la fréquence d'échantillonnage native du capteur: ces deux fréquences sont complètement indépendantes. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

voltage→LogicalName YVoltage

Nom logique de la fonction.

dnp string **LogicalName**

Modifiable. 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.

voltage→ReportFrequency YVoltage

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

dnp string **ReportFrequency**

Modifiable. Modifie la fréquence de notification périodique des valeurs mesurées. La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF". Attention il est inutile, voir contre productif, de régler la fréquence de notification périodique à une valeur supérieure à la fréquence d'échantillonnage native du capteur: ces deux fréquences sont complètement indépendantes. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

voltage→Resolution YVoltage

Résolution des valeurs mesurées.

dnp double **Resolution**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Modifiable. Modifie la résolution des valeurs physique mesurées. La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

voltage→SerialNumber YVoltage

Numéro de série du module, préprogrammé en usine.

dnp string **SerialNumber**

voltage→calibrateFromPoints() YVoltage

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

```
js   function calibrateFromPoints( rawValues, refValues)
cpp  int calibrateFromPoints( vector<double> rawValues,
                              vector<double> refValues)
m    -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                                : (NSMutableArray*) refValues
pas  LongInt calibrateFromPoints( rawValues: TDoubleArray,
                                  refValues: TDoubleArray): LongInt
vb   procedure calibrateFromPoints( ByVal rawValues As List(Of)
cs   int calibrateFromPoints( List<double> rawValues,
                              List<double> refValues)
java int calibrateFromPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)
uwp  async Task<int> calibrateFromPoints( List<double> rawValues,
                                          List<double> refValues)
py   calibrateFromPoints( rawValues, refValues)
php  function calibrateFromPoints( $rawValues, $refValues)
ts   async calibrateFromPoints( rawValues: number[], refValues: number[]): Promise<number>
es   async calibrateFromPoints( rawValues, refValues)
dnp  int calibrateFromPoints( double[] rawValues,
                              double[] refValues)
cp   int calibrateFromPoints( vector<double> rawValues,
                              vector<double> refValues)
cmd  YVoltage target calibrateFromPoints rawValues refValues
```

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

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

Paramètres :

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

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

voltage→clearCache() YVoltage

Invalide le cache.

js	function **clearCache**( )
cpp	void **clearCache**( )
m	-(void) **clearCache**
pas	**clearCache**( )
vb	procedure **clearCache**( )
cs	void **clearCache**( )
java	void **clearCache**( )
py	**clearCache**( )
php	function **clearCache**( )
ts	async **clearCache**( ): Promise<void>
es	async **clearCache**( )

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

voltage→describe() YVoltage

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

js	function **describe**( )
cpp	string **describe**( )
m	-(NSString*) **describe**
pas	string **describe**( ): string
vb	function **describe**( ) As String
cs	string **describe**( )
java	String **describe**( )
py	**describe**( )
php	function **describe**( )
ts	async **describe**( ): Promise<string>
es	async **describe**( )

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

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

voltage→get_advMode() YVoltage
voltage→advMode()

Retourne le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

```
js    function get_advMode( )
cpp   Y_ADVMODE_enum get_advMode( )
m     -(Y_ADVMODE_enum) advMode
pas   Integer get_advMode( ): Integer
vb    function get_advMode( ) As Integer
cs    int get_advMode( )
java  int get_advMode( )
uwp   async Task<int> get_advMode( )
py    get_advMode( )
php   function get_advMode( )
ts    async get_advMode( ): Promise<YSensor_AdvMode>
es    async get_advMode( )
dnp   int get_advMode( )
cp    int get_advMode( )
cmd   YVoltage target get_advMode
```

Retourne :

une valeur parmi `YVoltage.ADVMODE_IMMEDIATE`, `YVoltage.ADVMODE_PERIOD_AVG`, `YVoltage.ADVMODE_PERIOD_MIN` et `YVoltage.ADVMODE_PERIOD_MAX` représentant le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue)

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

voltage→get_advertisedValue() YVoltage
voltage→advertisedValue()

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

```
js    function get_advertisedValue( )
cpp   string get_advertisedValue( )
m     -(NSString*) advertisedValue
pas   string get_advertisedValue( ): string
vb    function get_advertisedValue( ) As String
cs    string get_advertisedValue( )
java  String get_advertisedValue( )
uwp   async Task<string> get_advertisedValue( )
py    get_advertisedValue( )
php   function get_advertisedValue( )
ts    async get_advertisedValue( ): Promise<string>
es    async get_advertisedValue( )
dnp   string get_advertisedValue( )
cp    string get_advertisedValue( )
cmd   YVoltage target get_advertisedValue
```

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

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

voltage→get_currentRawValue() YVoltage
voltage→currentRawValue()

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

```
js    function get_currentRawValue( )
cpp   double get_currentRawValue( )
m     -(double) currentRawValue
pas   double get_currentRawValue( ): double
vb    function get_currentRawValue( ) As Double
cs    double get_currentRawValue( )
java  double get_currentRawValue( )
uwp   async Task<double> get_currentRawValue( )
py    get_currentRawValue( )
php   function get_currentRawValue( )
ts    async get_currentRawValue( ): Promise<number>
es    async get_currentRawValue( )
dnp   double get_currentRawValue( )
cp    double get_currentRawValue( )
cmd   YVoltage target get_currentRawValue
```

Retourne :

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

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

voltage→get_currentValue() YVoltage
voltage→currentValue()

Retourne la valeur actuelle de la tension, en Volt, sous forme de nombre à virgule.

```
js    function get_currentValue( )
cpp   double get_currentValue( )
m     -(double) currentValue
pas   double get_currentValue( ): double
vb    function get_currentValue( ) As Double
cs    double get_currentValue( )
java  double get_currentValue( )
uwp   async Task<double> get_currentValue( )
py    get_currentValue( )
php   function get_currentValue( )
ts    async get_currentValue( ): Promise<number>
es    async get_currentValue( )
dnp   double get_currentValue( )
cp    double get_currentValue( )
cmd   YVoltage target get_currentValue
```

Notez qu'un appel à get_currentValue() ne déclenche pas une mesure dans le module mais retourne simplement la valeur obtenue lors de la dernière mesure. En effet, en interne, chaque senseur Yoctopuce effectue des mesures en continu à une fréquence qui lui est propre.

Si vous rencontrez des problèmes de performances en utilisant la fonction get_currentValue() fréquemment, il vous faudra basculer sur un modèle de callbacks. Pour plus de détails jetez un coup d'oeil au chapitre "programmation avancée" du manuel de votre module.

Retourne :
une valeur numérique représentant la valeur actuelle de la tension, en Volt, sous forme de nombre à virgule

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

voltage→get_dataLogger() YVoltage
voltage→dataLogger()

Retourne l'objet `YDatalogger` du module qui héberge le senseur.

js	function **get_dataLogger**( )
cpp	YDataLogger* **get_dataLogger**( )
m	-(YDataLogger*) dataLogger
pas	TYDataLogger **get_dataLogger**( ): TYDataLogger
vb	function **get_dataLogger**( ) As YDataLogger
cs	YDataLogger **get_dataLogger**( )
java	YDataLogger **get_dataLogger**( )
uwp	async Task<YDataLogger> **get_dataLogger**( )
py	**get_dataLogger**( )
php	function **get_dataLogger**( )
ts	async **get_dataLogger**( ): Promise<YDataLogger | null>
es	async **get_dataLogger**( )
dnp	YDataLoggerProxy **get_dataLogger**( )
cp	YDataLoggerProxy* **get_dataLogger**( )

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

Retourne :
un objet `YDatalogger`, ou null en cas d'erreur.

voltage→get_enabled() YVoltage
voltage→enabled()

Retourne l'état d'activation de cette mesure.

```
js    function get_enabled( )
cpp   Y_ENABLED_enum get_enabled( )
m     -(Y_ENABLED_enum) enabled
pas   Integer get_enabled( ): Integer
vb    function get_enabled( ) As Integer
cs    int get_enabled( )
java  int get_enabled( )
uwp   async Task<int> get_enabled( )
py    get_enabled( )
php   function get_enabled( )
ts    async get_enabled( ): Promise<YVoltage_Enabled>
es    async get_enabled( )
dnp   int get_enabled( )
cp    int get_enabled( )
cmd   YVoltage target get_enabled
```

Retourne :

soit `YVoltage.ENABLED_FALSE`, soit `YVoltage.ENABLED_TRUE`, selon l'état d'activation de cette mesure

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

voltage→get_errorMessage() YVoltage
voltage→errorMessage()

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

js	function **get_errorMessage**( )
cpp	string **get_errorMessage**( )
m	-(NSString*) errorMessage
pas	string **get_errorMessage**( ): string
vb	function **get_errorMessage**( ) As String
cs	string **get_errorMessage**( )
java	String **get_errorMessage**( )
py	**get_errorMessage**( )
php	function **get_errorMessage**( )
ts	**get_errorMessage**( ): string
es	**get_errorMessage**( )

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

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

voltage→get_errorType() YVoltage
voltage→errorType()

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

js	function **get_errorType**( )
cpp	YRETCODE **get_errorType**( )
m	-(YRETCODE) errorType
pas	YRETCODE **get_errorType**( ): YRETCODE
vb	function **get_errorType**( ) As YRETCODE
cs	YRETCODE **get_errorType**( )
java	int **get_errorType**( )
py	**get_errorType**( )
php	function **get_errorType**( )
ts	**get_errorType**( ): number
es	**get_errorType**( )

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

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

voltage→get_friendlyName() YVoltage
voltage→friendlyName()

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

```
js    function get_friendlyName( )
cpp   string get_friendlyName( )
m     -(NSString*) friendlyName
cs    string get_friendlyName( )
java  String get_friendlyName( )
py    get_friendlyName( )
php   function get_friendlyName( )
ts    async get_friendlyName( ): Promise<string>
es    async get_friendlyName( )
dnp   string get_friendlyName( )
cp    string get_friendlyName( )
```

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

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

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

voltage→get_functionDescriptor() YVoltage
voltage→functionDescriptor()

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

js	function **get_functionDescriptor**( )
cpp	YFUN_DESCR **get_functionDescriptor**( )
m	-(YFUN_DESCR) functionDescriptor
pas	YFUN_DESCR **get_functionDescriptor**( ): YFUN_DESCR
vb	function **get_functionDescriptor**( ) As YFUN_DESCR
cs	YFUN_DESCR **get_functionDescriptor**( )
java	String **get_functionDescriptor**( )
py	**get_functionDescriptor**( )
php	function **get_functionDescriptor**( )
ts	async **get_functionDescriptor**( ): Promise<string>
es	async **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$CLASSNAME$.FUNCTIONDESCRIPTOR_INVALID

voltage→get_functionId() YVoltage
voltage→functionId()

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

js	function **get_functionId**( )
cpp	string **get_functionId**( )
m	-(NSString*) functionId
vb	function **get_functionId**( ) As String
cs	string **get_functionId**( )
java	String **get_functionId**( )
py	**get_functionId**( )
php	function **get_functionId**( )
ts	async **get_functionId**( ): Promise<string>
es	async **get_functionId**( )
dnp	string **get_functionId**( )
cp	string **get_functionId**( )

Par example `relay1`.

Retourne :
une chaîne de caractères identifiant le capteur de tension (ex: `relay1`)

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

voltage→get_hardwareId() YVoltage
voltage→hardwareId()

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

js	function **get_hardwareId**( )
cpp	string **get_hardwareId**( )
m	-(NSString*) hardwareId
vb	function **get_hardwareId**( ) As String
cs	string **get_hardwareId**( )
java	String **get_hardwareId**( )
py	**get_hardwareId**( )
php	function **get_hardwareId**( )
ts	async **get_hardwareId**( ): Promise<string>
es	async **get_hardwareId**( )
dnp	string **get_hardwareId**( )
cp	string **get_hardwareId**( )

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

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

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

voltage→get_highestValue() YVoltage
voltage→highestValue()

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

```
js    function get_highestValue( )
cpp   double get_highestValue( )
m     -(double) highestValue
pas   double get_highestValue( ): double
vb    function get_highestValue( ) As Double
cs    double get_highestValue( )
java  double get_highestValue( )
uwp   async Task<double> get_highestValue( )
py    get_highestValue( )
php   function get_highestValue( )
ts    async get_highestValue( ): Promise<number>
es    async get_highestValue( )
dnp   double get_highestValue( )
cp    double get_highestValue( )
cmd   YVoltage target get_highestValue
```

Peut être réinitialisé à une valeur arbitraire grâce à set_highestValue().

Retourne :
une valeur numérique représentant la valeur maximale observée pour la tension depuis le démarrage du module

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

voltage→get_logFrequency() YVoltage
voltage→logFrequency()

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

```
js    function get_logFrequency( )
cpp   string get_logFrequency( )
m     -(NSString*) logFrequency
pas   string get_logFrequency( ): string
vb    function get_logFrequency( ) As String
cs    string get_logFrequency( )
java  String get_logFrequency( )
uwp   async Task<string> get_logFrequency( )
py    get_logFrequency( )
php   function get_logFrequency( )
ts    async get_logFrequency( ): Promise<string>
es    async get_logFrequency( )
dnp   string get_logFrequency( )
cp    string get_logFrequency( )
cmd   YVoltage target get_logFrequency
```

Retourne :

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

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

voltage→get_logicalName() YVoltage
voltage→logicalName()

Retourne le nom logique du capteur de tension.

```
js    function get_logicalName( )
cpp   string get_logicalName( )
m     -(NSString*) logicalName
pas   string get_logicalName( ): string
vb    function get_logicalName( ) As String
cs    string get_logicalName( )
java  String get_logicalName( )
uwp   async Task<string> get_logicalName( )
py    get_logicalName( )
php   function get_logicalName( )
ts    async get_logicalName( ): Promise<string>
es    async get_logicalName( )
dnp   string get_logicalName( )
cp    string get_logicalName( )
cmd   YVoltage target get_logicalName
```

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

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

voltage→get_lowestValue() YVoltage
voltage→lowestValue()

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

js	function **get_lowestValue**( )
cpp	double **get_lowestValue**( )
m	-(double) lowestValue
pas	double **get_lowestValue**( ): double
vb	function **get_lowestValue**( ) As Double
cs	double **get_lowestValue**( )
java	double **get_lowestValue**( )
uwp	async Task<double> **get_lowestValue**( )
py	**get_lowestValue**( )
php	function **get_lowestValue**( )
ts	async **get_lowestValue**( ): Promise<number>
es	async **get_lowestValue**( )
dnp	double **get_lowestValue**( )
cp	double **get_lowestValue**( )
cmd	YVoltage **target get_lowestValue**

Peut être réinitialisé à une valeur arbitraire grâce à set_lowestValue().

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

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

voltage→get_module() YVoltage
voltage→module()

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

js	function **get_module**( )
cpp	YModule * **get_module**( )
m	-(YModule*) module
pas	TYModule **get_module**( ): TYModule
vb	function **get_module**( ) As YModule
cs	YModule **get_module**( )
java	YModule **get_module**( )
py	**get_module**( )
php	function **get_module**( )
ts	async **get_module**( ): Promise<YModule>
es	async **get_module**( )
dnp	YModuleProxy **get_module**( )
cp	YModuleProxy * **get_module**( )

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

Retourne :
une instance de `YModule`

voltage→get_module_async() YVoltage
voltage→module_async()

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

```
js  function get_module_async( callback, context)
```

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_recordedData() YVoltage
voltage→recordedData()

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

```
js    function get_recordedData( startTime, endTime)
cpp   YDataSet get_recordedData( double startTime, double endTime)
m     -(YDataSet*) recordedData : (double) startTime
                                : (double) endTime
pas   TYDataSet get_recordedData( startTime: double, endTime: double): TYDataSet
vb    function get_recordedData( ByVal startTime As Double,
                                 ByVal endTime As Double) As YDataSet
cs    YDataSet get_recordedData( double startTime, double endTime)
java  YDataSet get_recordedData( double startTime, double endTime)
uwp   async Task<YDataSet> get_recordedData( double startTime,
                                             double endTime)
py    get_recordedData( startTime, endTime)
php   function get_recordedData( $startTime, $endTime)
ts    async get_recordedData( startTime: number, endTime: number): Promise<YDataSet>
es    async get_recordedData( startTime, endTime)
dnp   YDataSetProxy get_recordedData( double startTime,
                                      double endTime)
cp    YDataSetProxy* get_recordedData( double startTime,
                                       double endTime)
cmd   YVoltage target get_recordedData startTime endTime
```

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

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

Paramètres :

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

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

Retourne :

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

voltage→get_reportFrequency() YVoltage
voltage→reportFrequency()

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

```
js    function get_reportFrequency( )
cpp   string get_reportFrequency( )
m     -(NSString*) reportFrequency
pas   string get_reportFrequency( ): string
vb    function get_reportFrequency( ) As String
cs    string get_reportFrequency( )
java  String get_reportFrequency( )
uwp   async Task<string> get_reportFrequency( )
py    get_reportFrequency( )
php   function get_reportFrequency( )
ts    async get_reportFrequency( ): Promise<string>
es    async get_reportFrequency( )
dnp   string get_reportFrequency( )
cp    string get_reportFrequency( )
cmd   YVoltage target get_reportFrequency
```

Retourne :

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

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

voltage→get_resolution() YVoltage
voltage→resolution()

Retourne la résolution des valeurs mesurées.

```
js    function get_resolution( )
cpp   double get_resolution( )
m     -(double) resolution
pas   double get_resolution( ): double
vb    function get_resolution( ) As Double
cs    double get_resolution( )
java  double get_resolution( )
uwp   async Task<double> get_resolution( )
py    get_resolution( )
php   function get_resolution( )
ts    async get_resolution( ): Promise<number>
es    async get_resolution( )
dnp   double get_resolution( )
cp    double get_resolution( )
cmd   YVoltage target get_resolution
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

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

voltage→get_sensorState() YVoltage
voltage→sensorState()

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

js	function **get_sensorState**( )
cpp	int **get_sensorState**( )
m	-(int) sensorState
pas	LongInt **get_sensorState**( ): LongInt
vb	function **get_sensorState**( ) As Integer
cs	int **get_sensorState**( )
java	int **get_sensorState**( )
uwp	async Task<int> **get_sensorState**( )
py	**get_sensorState**( )
php	function **get_sensorState**( )
ts	async **get_sensorState**( ): Promise<number>
es	async **get_sensorState**( )
dnp	int **get_sensorState**( )
cp	int **get_sensorState**( )
cmd	YVoltage **target get_sensorState**

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

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

voltage→get_serialNumber() YVoltage
voltage→serialNumber()

Retourne le numéro de série du module, préprogrammé en usine.

```
js      function get_serialNumber( )
cpp     string get_serialNumber( )
m       -(NSString*) serialNumber
pas     string get_serialNumber( ): string
vb      function get_serialNumber( ) As String
cs      string get_serialNumber( )
java    String get_serialNumber( )
uwp     async Task<string> get_serialNumber( )
py      get_serialNumber( )
php     function get_serialNumber( )
ts      async get_serialNumber( ): Promise<string>
es      async get_serialNumber( )
dnp     string get_serialNumber( )
cp      string get_serialNumber( )
cmd     YVoltage target get_serialNumber
```

Retourne :
: une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine.

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

voltage→get_unit() YVoltage
voltage→unit()

Retourne l'unité dans laquelle la tension est exprimée.

```
js    function get_unit( )
cpp   string get_unit( )
m     -(NSString*) unit
pas   string get_unit( ): string
vb    function get_unit( ) As String
cs    string get_unit( )
java  String get_unit( )
uwp   async Task<string> get_unit( )
py    get_unit( )
php   function get_unit( )
ts    async get_unit( ): Promise<string>
es    async get_unit( )
dnp   string get_unit( )
cp    string get_unit( )
cmd   YVoltage target get_unit
```

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

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

voltage→get_userData() YVoltage
voltage→userData()

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

js	function **get_userData**( )
cpp	void * **get_userData**( )
m	-(id) userData
pas	Tobject **get_userData**( ): Tobject
vb	function **get_userData**( ) As Object
cs	object **get_userData**( )
java	Object **get_userData**( )
py	**get_userData**( )
php	function **get_userData**( )
ts	async **get_userData**( ): Promise<object|null>
es	async **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() YVoltage

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

js	function **isOnline**( )
cpp	bool **isOnline**( )
m	-(BOOL) **isOnline**
pas	boolean **isOnline**( ): boolean
vb	function **isOnline**( ) As Boolean
cs	bool **isOnline**( )
java	boolean **isOnline**( )
py	**isOnline**( )
php	function **isOnline**( )
ts	async **isOnline**( ): Promise<boolean>
es	async **isOnline**( )
dnp	bool **isOnline**( )
cp	bool **isOnline**( )

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

Retourne :
`true` si le capteur de tension est joignable, `false` sinon

voltage→isOnline_async() YVoltage

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

```
js function isOnline_async( callback, context)
```

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

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 machine virtuelle Javascript avec une attente active.

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→isReadOnly() YVoltage

Test si la fonction est en lecture seule.

cpp	bool **isReadOnly**( )
m	-(bool) **isReadOnly**
pas	boolean **isReadOnly**( ): boolean
vb	function **isReadOnly**( ) As Boolean
cs	bool **isReadOnly**( )
java	boolean **isReadOnly**( )
uwp	async Task<bool> **isReadOnly**( )
py	**isReadOnly**( )
php	function **isReadOnly**( )
ts	async **isReadOnly**( ): Promise<boolean>
es	async **isReadOnly**( )
dnp	bool **isReadOnly**( )
cp	bool **isReadOnly**( )
cmd	YVoltage **target isReadOnly**

Retourne vrais si la fonction est protégé en ecriture ou que la fontion n'est pas disponible.

Retourne :
`true` si la fonction est protégé en ecriture ou que la fontion n'est pas disponible

voltage→isSensorReady() YVoltage

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

```
cmd YVoltage target isSensorReady
```

Retourne faux si le module n'est pas joignable, ou que le capteur n'a pas de mesure actuelle à communiquer. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si le capteur dispose d'une mesure actuelle, `false` sinon

voltage→load() YVoltage

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

```
js    function load( msValidity)
cpp   YRETCODE load( int msValidity)
m     -(YRETCODE) load : (u64) msValidity
pas   YRETCODE load( msValidity: u64): YRETCODE
vb    function load( ByVal msValidity As Long) As YRETCODE
cs    YRETCODE load( ulong msValidity)
java  int load( long msValidity)
py    load( msValidity)
php   function load( $msValidity)
ts    async load( msValidity: number): Promise<number>
es    async load( msValidity)
```

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

Paramètres :

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

voltage→loadAttribute() YVoltage

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

```
js    function loadAttribute( attrName)
cpp   string loadAttribute( string attrName)
m     -(NSString*) loadAttribute : (NSString*) attrName
pas   string loadAttribute( attrName: string): string
vb    function loadAttribute( ByVal attrName As String) As String
cs    string loadAttribute( string attrName)
java  String loadAttribute( String attrName)
uwp   async Task<string> loadAttribute( string attrName)
py    loadAttribute( attrName)
php   function loadAttribute( $attrName)
ts    async loadAttribute( attrName: string): Promise<string>
es    async loadAttribute( attrName)
dnp   string loadAttribute( string attrName)
cp    string loadAttribute( string attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

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

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

voltage→loadCalibrationPoints() YVoltage

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

```
js   function loadCalibrationPoints( rawValues, refValues)
cpp  int loadCalibrationPoints( vector<double> rawValues,
                                vector<double> refValues)
m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                                  : (NSMutableArray*) refValues
pas  LongInt loadCalibrationPoints( var rawValues: TDoubleArray,
                                    var refValues: TDoubleArray): LongInt
vb   procedure loadCalibrationPoints( ByVal rawValues As List(Of)
cs   int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)
java int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)
uwp  async Task<int> loadCalibrationPoints( List<double> rawValues,
                                            List<double> refValues)
py   loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
ts   async loadCalibrationPoints( rawValues: number[], refValues: number[]): Promise<number>
es   async loadCalibrationPoints( rawValues, refValues)
cmd  YVoltage target loadCalibrationPoints rawValues refValues
```

Paramètres :

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

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

voltage→load_async() YVoltage

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

```
js function load_async( msValidity, callback, context)
```

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 machine virtuelle Javascript avec une attente active.

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→muteValueCallbacks() YVoltage

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

js	function **muteValueCallbacks**( )
cpp	int **muteValueCallbacks**( )
m	-(int) **muteValueCallbacks**
pas	LongInt **muteValueCallbacks**( ): LongInt
vb	function **muteValueCallbacks**( ) As Integer
cs	int **muteValueCallbacks**( )
java	int **muteValueCallbacks**( )
uwp	async Task<int> **muteValueCallbacks**( )
py	**muteValueCallbacks**( )
php	function **muteValueCallbacks**( )
ts	async **muteValueCallbacks**( ): Promise<number>
es	async **muteValueCallbacks**( )
dnp	int **muteValueCallbacks**( )
cp	int **muteValueCallbacks**( )
cmd	YVoltage **target muteValueCallbacks**

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

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

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

voltage→nextVoltage() YVoltage

Continue l'énumération des capteurs de tension commencée à l'aide de `yFirstVoltage()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les capteurs de tension sont retournés.

js	function **nextVoltage**( )
cpp	YVoltage * **nextVoltage**( )
m	-(nullable YVoltage*) **nextVoltage**
pas	TYVoltage **nextVoltage**( ): TYVoltage
vb	function **nextVoltage**( ) As YVoltage
cs	YVoltage **nextVoltage**( )
java	YVoltage **nextVoltage**( )
uwp	YVoltage **nextVoltage**( )
py	**nextVoltage**( )
php	function **nextVoltage**( )
ts	**nextVoltage**( ): YVoltage | null
es	**nextVoltage**( )

Si vous souhaitez retrouver un capteur de tension spécifique, utilisez `Voltage.findVoltage()` avec un hardwareID ou un nom logique.

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

voltage→registerTimedReportCallback() YVoltage

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

js	function **registerTimedReportCallback**(**callback**)
cpp	int **registerTimedReportCallback**(YVoltageTimedReportCallback **callback**)
m	-(int) **registerTimedReportCallback** : (YVoltageTimedReportCallback _Nullable) **callback**
pas	LongInt **registerTimedReportCallback**(**callback**: TYVoltageTimedReportCallback): LongInt
vb	function **registerTimedReportCallback**(ByVal **callback** As YVoltageTimedReportCallback) As Integer
cs	int **registerTimedReportCallback**(TimedReportCallback **callback**)
java	int **registerTimedReportCallback**(TimedReportCallback **callback**)
uwp	async Task<int> **registerTimedReportCallback**(TimedReportCallback **callback**)
py	**registerTimedReportCallback**(**callback**)
php	function **registerTimedReportCallback**(**$callback**)
ts	async **registerTimedReportCallback**(**callback**: YVoltageTimedReportCallback | null): Promise<number>
es	async **registerTimedReportCallback**(**callback**)

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

Paramètres :

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

voltage→registerValueCallback() YVoltage

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

```
js    function registerValueCallback( callback)
cpp   int registerValueCallback( YVoltageValueCallback callback)
m     -(int) registerValueCallback : (YVoltageValueCallback _Nullable) callback
pas   LongInt registerValueCallback( callback: TYVoltageValueCallback): LongInt
vb    function registerValueCallback( ByVal callback As YVoltageValueCallback) As Integer
cs    int registerValueCallback( ValueCallback callback)
java  int registerValueCallback( UpdateCallback callback)
uwp   async Task<int> registerValueCallback( ValueCallback callback)
py    registerValueCallback( callback)
php   function registerValueCallback( $callback)
ts    async registerValueCallback( callback: YVoltageValueCallback | null): Promise<number>
es    async registerValueCallback( callback)
```

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

Paramètres :

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

voltage→set_advMode() YVoltage
voltage→setAdvMode()

Modifie le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

```
js    function set_advMode( newval)
cpp   int set_advMode( Y_ADVMODE_enum newval)
m     -(int) setAdvMode : (Y_ADVMODE_enum) newval
pas   integer set_advMode( newval: Integer): integer
vb    function set_advMode( ByVal newval As Integer) As Integer
cs    int set_advMode( int newval)
java  int set_advMode( int newval)
uwp   async Task<int> set_advMode( int newval)
py    set_advMode( newval)
php   function set_advMode( $newval)
ts    async set_advMode( newval: YSensor_AdvMode): Promise<number>
es    async set_advMode( newval)
dnp   int set_advMode( int newval)
cp    int set_advMode( int newval)
cmd   YVoltage target set_advMode newval
```

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

Paramètres :

newval une valeur parmi `YVoltage.ADVMODE_IMMEDIATE`, `YVoltage.ADVMODE_PERIOD_AVG`, `YVoltage.ADVMODE_PERIOD_MIN` et `YVoltage.ADVMODE_PERIOD_MAX` représentant le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue)

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_enabled() YVoltage
voltage→setEnabled()

Modifie l'état d'activation de le mesure de tension.

```
js    function set_enabled( newval)
cpp   int set_enabled( Y_ENABLED_enum newval)
m     -(int) setEnabled : (Y_ENABLED_enum) newval
pas   integer set_enabled( newval: Integer): integer
vb    function set_enabled( ByVal newval As Integer) As Integer
cs    int set_enabled( int newval)
java  int set_enabled( int newval)
uwp   async Task<int> set_enabled( int newval)
py    set_enabled( newval)
php   function set_enabled( $newval)
ts    async set_enabled( newval: YVoltage_Enabled): Promise<number>
es    async set_enabled( newval)
dnp   int set_enabled( int newval)
cp    int set_enabled( int newval)
cmd   YVoltage target set_enabled newval
```

Lorsque la mesure de tension AC est désactivée, la mesure se fera uniquement en mode DC, et inversément. Lorsque les deux mesures AC et DC sont activées, le module bascule entre le mode AC et DC en fonction de l'amplitude relative des variations AC par rapport à la tension moyenne. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval soit `YVoltage.ENABLED_FALSE`, soit `YVoltage.ENABLED_TRUE`, selon l'état d'activation de le mesure 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_highestValue() YVoltage
voltage→setHighestValue()

Modifie la mémoire de valeur maximale observée.

```
js    function set_highestValue( newval)
cpp   int set_highestValue( double newval)
m     -(int) setHighestValue : (double) newval
pas   integer set_highestValue( newval: double): integer
vb    function set_highestValue( ByVal newval As Double) As Integer
cs    int set_highestValue( double newval)
java  int set_highestValue( double newval)
uwp   async Task<int> set_highestValue( double newval)
py    set_highestValue( newval)
php   function set_highestValue( $newval)
ts    async set_highestValue( newval: number): Promise<number>
es    async set_highestValue( newval)
dnp   int set_highestValue( double newval)
cp    int set_highestValue( double newval)
cmd   YVoltage target set_highestValue newval
```

Utile pour réinitialiser la valeur renvoyée par get_highestValue().

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

voltage→set_logFrequency() YVoltage
voltage→setLogFrequency()

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

```
js    function set_logFrequency( newval)
cpp   int set_logFrequency( string newval)
m     -(int) setLogFrequency : (NSString*) newval
pas   integer set_logFrequency( newval: string): integer
vb    function set_logFrequency( ByVal newval As String) As Integer
cs    int set_logFrequency( string newval)
java  int set_logFrequency( String newval)
uwp   async Task<int> set_logFrequency( string newval)
py    set_logFrequency( newval)
php   function set_logFrequency( $newval)
ts    async set_logFrequency( newval: string): Promise<number>
es    async set_logFrequency( newval)
dnp   int set_logFrequency( string newval)
cp    int set_logFrequency( string newval)
cmd   YVoltage target set_logFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF". Attention il est inutile, voir contre productif, de régler la fréquence d'enregistrement à une valeur supérieure à la fréquence d'échantillonnage native du capteur: ces deux fréquences sont complètement indépendantes. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

voltage→set_logicalName() YVoltage
voltage→setLogicalName()

Modifie le nom logique du capteur de tension.

```
js    function set_logicalName( newval)
cpp   int set_logicalName( string newval)
m     -(int) setLogicalName : (NSString*) newval
pas   integer set_logicalName( newval: string): integer
vb    function set_logicalName( ByVal newval As String) As Integer
cs    int set_logicalName( string newval)
java  int set_logicalName( String newval)
uwp   async Task<int> set_logicalName( string newval)
py    set_logicalName( newval)
php   function set_logicalName( $newval)
ts    async set_logicalName( newval: string): Promise<number>
es    async set_logicalName( newval)
dnp   int set_logicalName( string newval)
cp    int set_logicalName( string newval)
cmd   YVoltage target set_logicalName newval
```

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

Paramètres :

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

Retourne :

`YAPI.SUCCESS` si l'appel se déroule sans erreur.

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

voltage→set_lowestValue() YVoltage
voltage→setLowestValue()

Modifie la mémoire de valeur minimale observée.

```
js    function set_lowestValue( newval)
cpp   int set_lowestValue( double newval)
m     -(int) setLowestValue : (double) newval
pas   integer set_lowestValue( newval: double): integer
vb    function set_lowestValue( ByVal newval As Double) As Integer
cs    int set_lowestValue( double newval)
java  int set_lowestValue( double newval)
uwp   async Task<int> set_lowestValue( double newval)
py    set_lowestValue( newval)
php   function set_lowestValue( $newval)
ts    async set_lowestValue( newval: number): Promise<number>
es    async set_lowestValue( newval)
dnp   int set_lowestValue( double newval)
cp    int set_lowestValue( double newval)
cmd   YVoltage target set_lowestValue newval
```

Utile pour réinitialiser la valeur renvoyée par get_lowestValue().

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

voltage→set_reportFrequency() YVoltage
voltage→setReportFrequency()

Modifie la fréquence de notification périodique des valeurs mesurées.

```
js    function set_reportFrequency( newval)
cpp   int set_reportFrequency( string newval)
m     -(int) setReportFrequency : (NSString*) newval
pas   integer set_reportFrequency( newval: string): integer
vb    function set_reportFrequency( ByVal newval As String) As Integer
cs    int set_reportFrequency( string newval)
java  int set_reportFrequency( String newval)
uwp   async Task<int> set_reportFrequency( string newval)
py    set_reportFrequency( newval)
php   function set_reportFrequency( $newval)
ts    async set_reportFrequency( newval: string): Promise<number>
es    async set_reportFrequency( newval)
dnp   int set_reportFrequency( string newval)
cp    int set_reportFrequency( string newval)
cmd   YVoltage target set_reportFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF". Attention il est inutile, voir contre productif, de régler la fréquence de notification périodique à une valeur supérieure à la fréquence d'échantillonnage native du capteur: ces deux fréquences sont complètement indépendantes. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

voltage→set_resolution() YVoltage
voltage→setResolution()

Modifie la résolution des valeurs physique mesurées.

```
js    function set_resolution( newval)
cpp   int set_resolution( double newval)
m     -(int) setResolution : (double) newval
pas   integer set_resolution( newval: double): integer
vb    function set_resolution( ByVal newval As Double) As Integer
cs    int set_resolution( double newval)
java  int set_resolution( double newval)
uwp   async Task<int> set_resolution( double newval)
py    set_resolution( newval)
php   function set_resolution( $newval)
ts    async set_resolution( newval: number): Promise<number>
es    async set_resolution( newval)
dnp   int set_resolution( double newval)
cp    int set_resolution( double newval)
cmd   YVoltage target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

voltage→set_userData() YVoltage
voltage→setUserData()

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

```
js    function set_userData( data)
cpp   void set_userData( void * data)
m     -(void) setUserData : (id) data
pas   set_userData( data: Tobject)
vb    procedure set_userData( ByVal data As Object)
cs    void set_userData( object data)
java  void set_userData( Object data)
py    set_userData( data)
php   function set_userData( $data)
ts    async set_userData( data: object|null): Promise<void>
es    async set_userData( data)
```

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

Paramètres :

data objet quelconque à mémoriser

voltage→startDataLogger() YVoltage

Démarre l'enregistreur de données du module.

js	function **startDataLogger**( )
cpp	int **startDataLogger**( )
m	-(int) **startDataLogger**
pas	LongInt **startDataLogger**( ): LongInt
vb	function **startDataLogger**( ) As Integer
cs	int **startDataLogger**( )
java	int **startDataLogger**( )
uwp	async Task<int> **startDataLogger**( )
py	**startDataLogger**( )
php	function **startDataLogger**( )
ts	async **startDataLogger**( ): Promise<number>
es	async **startDataLogger**( )
dnp	int **startDataLogger**( )
cp	int **startDataLogger**( )
cmd	YVoltage **target startDataLogger**

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

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

voltage→stopDataLogger() YVoltage

Arrête l'enregistreur de données du module.

js	function **stopDataLogger**( )
cpp	int **stopDataLogger**( )
m	-(int) **stopDataLogger**
pas	LongInt **stopDataLogger**( ): LongInt
vb	function **stopDataLogger**( ) As Integer
cs	int **stopDataLogger**( )
java	int **stopDataLogger**( )
uwp	async Task<int> **stopDataLogger**( )
py	**stopDataLogger**( )
php	function **stopDataLogger**( )
ts	async **stopDataLogger**( ): Promise<number>
es	async **stopDataLogger**( )
dnp	int **stopDataLogger**( )
cp	int **stopDataLogger**( )
cmd	YVoltage **target stopDataLogger**

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

voltage→unmuteValueCallbacks() YVoltage

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

js	function **unmuteValueCallbacks**( )
cpp	int **unmuteValueCallbacks**( )
m	-(int) **unmuteValueCallbacks**
pas	LongInt **unmuteValueCallbacks**( ): LongInt
vb	function **unmuteValueCallbacks**( ) As Integer
cs	int **unmuteValueCallbacks**( )
java	int **unmuteValueCallbacks**( )
uwp	async Task<int> **unmuteValueCallbacks**( )
py	**unmuteValueCallbacks**( )
php	function **unmuteValueCallbacks**( )
ts	async **unmuteValueCallbacks**( ): Promise<number>
es	async **unmuteValueCallbacks**( )
dnp	int **unmuteValueCallbacks**( )
cp	int **unmuteValueCallbacks**( )
cmd	YVoltage **target unmuteValueCallbacks**

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

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

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

voltage→wait_async() YVoltage

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

```
js  function wait_async( callback, context)
ts  wait_async( callback: Function, context: object)
es  wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

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

Retourne :

rien du tout.

24.5. La classe YCurrent

Interface pour intéragir avec les capteurs de courant, disponibles par exemple dans le Yocto-Amp, le Yocto-Motor-DC et le Yocto-Watt

La classe `YCurrent` permet de lire et de configurer les capteurs de courant Yoctopuce. Elle hérite de la classe `YSensor` toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données.

Pour utiliser les fonctions décrites ici, vous devez inclure:

es	in HTML: <script src="../../lib/yocto_current.js"></script> in node.js: require('yoctolib-es2017/yocto_current.js');
js	<script type='text/javascript' src='yocto_current.js'></script>
cpp	#include "yocto_current.h"
m	#import "yocto_current.h"
pas	uses yocto_current;
vb	yocto_current.vb
cs	yocto_current.cs
java	import com.yoctopuce.YoctoAPI.YCurrent;
uwp	import com.yoctopuce.YoctoAPI.YCurrent;
py	from yocto_current import *
php	require_once('yocto_current.php');
ts	in HTML: import { YCurrent } from '../../dist/esm/yocto_current.js'; in Node.js: import { YCurrent } from 'yoctolib-cjs/yocto_current.js';
dnp	import YoctoProxyAPI.YCurrentProxy
cp	#include "yocto_current_proxy.h"
vi	YCurrent.vi
ml	import YoctoProxyAPI.YCurrentProxy

Fonction globales

YCurrent.FindCurrent(**func**)

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

YCurrent.FindCurrentInContext(**yctx**, **func**)

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

YCurrent.FirstCurrent()

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

YCurrent.FirstCurrentInContext(**yctx**)

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

YCurrent.GetSimilarFunctions()

Enumère toutes les fonctions de type Current disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

Propriétés des objets `YCurrentProxy`

current→AdvMode *[modifiable]*

Mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

current→AdvertisedValue *[lecture seule]*

Courte chaîne de caractères représentant l'état courant de la fonction.

current→Enabled *[modifiable]*

état d'activation de cette mesure.

current→FriendlyName *[lecture seule]*

Identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

current→FunctionId *[lecture seule]*

Identifiant matériel du senseur, sans référence au module.

current→HardwareId *[lecture seule]*

Identifiant matériel unique de la fonction au format `SERIAL.FUNCTIONID`.

current→IsOnline *[lecture seule]*

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

current→LogFrequency *[modifiable]*

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

current→LogicalName *[modifiable]*

Nom logique de la fonction.

current→ReportFrequency *[modifiable]*

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

current→Resolution *[modifiable]*

Résolution des valeurs mesurées.

current→SerialNumber *[lecture seule]*

Numéro de série du module, préprogrammé en usine.

Méthodes des objets `YCurrent`

current→calibrateFromPoints(rawValues, refValues)

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

current→clearCache()

Invalide le cache.

current→describe()

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

current→get_advMode()

Retourne le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

current→get_advertisedValue()

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

current→get_currentRawValue()

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

current→get_currentValue()

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

current→get_dataLogger()

Retourne l'objet `YDatalogger` du module qui héberge le senseur.

current→get_enabled()

Retourne l'état d'activation de cette mesure.

current→get_errorMessage()

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

current→get_errorType()

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

current→get_friendlyName()

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

current→get_functionDescriptor()

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

current→get_functionId()

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

current→get_hardwareId()

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

current→get_highestValue()

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

current→get_logFrequency()

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

current→get_logicalName()

Retourne le nom logique du capteur de courant.

current→get_lowestValue()

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

current→get_module()

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

current→get_module_async(callback, context)

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

current→get_recordedData(startTime, endTime)

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

current→get_reportFrequency()

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

current→get_resolution()

Retourne la résolution des valeurs mesurées.

current→get_sensorState()

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

current→get_serialNumber()

Retourne le numéro de série du module, préprogrammé en usine.

current→get_unit()

Retourne l'unité dans laquelle le courant est exprimée.

current→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

current→isOnline()

Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.

current→isOnline_async(callback, context)

Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.

current→isReadOnly()

Test si la fonction est en lecture seule.

current→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

current→load(msValidity)

Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.

current→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

current→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

current→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.

current→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

current→nextCurrent()

Continue l'énumération des capteurs de courant commencée à l'aide de `yFirstCurrent()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les capteurs de courant sont retournés.

current→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

current→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

current→set_advMode(newval)

Modifie le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

current→set_enabled(newval)

Modifie l'état d'activation de le mesure de courant.

current→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

current→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

current→set_logicalName(newval)

Modifie le nom logique du capteur de courant.

current→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

current→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

current→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

current→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

current→startDataLogger()

Démarre l'enregistreur de données du module.

current→stopDataLogger()

Arrête l'enregistreur de données du module.

current→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

current→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YCurrent.FindCurrent() YCurrent

YCurrent.FindCurrent()

Permet de retrouver un capteur de courant d'après un identifiant donné.

```
js    function yFindCurrent( func)
cpp   YCurrent* FindCurrent( string func)
m     +(YCurrent*) FindCurrent : (NSString*) func
pas   TYCurrent yFindCurrent( func: string): TYCurrent
vb    function FindCurrent( ByVal func As String) As YCurrent
cs    static YCurrent FindCurrent( string func)
java  static YCurrent FindCurrent( String func)
uwp   static YCurrent FindCurrent( string func)
py    FindCurrent( func)
php   function FindCurrent( $func)
ts    static FindCurrent( func: string): YCurrent
es    static FindCurrent( func)
dnp   static YCurrentProxy FindCurrent( string func)
cp    static YCurrentProxy * FindCurrent( string func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de courant soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YCurrent.isOnline()` pour tester si le capteur de courant est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le capteur de courant sans ambiguïté, par exemple `YAMPMK01.current1`.

Retourne :

un objet de classe `YCurrent` qui permet ensuite de contrôler le capteur de courant.

YCurrent.FindCurrentInContext() YCurrent

YCurrent.FindCurrentInContext()

Permet de retrouver un capteur de courant d'après un identifiant donné dans un Context YAPI.

```
java  static YCurrent FindCurrentInContext( YAPIContext yctx, String func)
uwp   static YCurrent FindCurrentInContext( YAPIContext yctx, string func)
ts    static FindCurrentInContext( yctx: YAPIContext, func: string): YCurrent
es    static FindCurrentInContext( yctx, func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de courant soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YCurrent.isOnline()` pour tester si le capteur de courant est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de courant sans ambiguïté, par exemple `YAMPMK01.current1`.

Retourne :

un objet de classe `YCurrent` qui permet ensuite de contrôler le capteur de courant.

YCurrent.FirstCurrent() YCurrent

YCurrent.FirstCurrent()

Commence l'énumération des capteurs de courant accessibles par la librairie.

```
js    function yFirstCurrent( )
cpp   YCurrent * FirstCurrent( )
m     +(YCurrent*) FirstCurrent
pas   TYCurrent yFirstCurrent( ): TYCurrent
vb    function FirstCurrent( ) As YCurrent
cs    static YCurrent FirstCurrent( )
java  static YCurrent FirstCurrent( )
uwp   static YCurrent FirstCurrent( )
py    FirstCurrent( )
php   function FirstCurrent( )
ts    static FirstCurrent( ): YCurrent | null
es    static FirstCurrent( )
```

Utiliser la fonction `YCurrent.nextCurrent()` pour itérer sur les autres capteurs de courant.

Retourne :

un pointeur sur un objet `YCurrent`, correspondant au premier capteur de courant accessible en ligne, ou `null` si il n'y a pas de capteurs de courant disponibles.

YCurrent.FirstCurrentInContext() YCurrent

YCurrent.FirstCurrentInContext()

Commence l'énumération des capteurs de courant accessibles par la librairie.

java	static YCurrent **FirstCurrentInContext**(YAPIContext **yctx**)
uwp	static YCurrent **FirstCurrentInContext**(YAPIContext **yctx**)
ts	static **FirstCurrentInContext**(**yctx**: YAPIContext): YCurrent | null
es	static **FirstCurrentInContext**(**yctx**)

Utiliser la fonction `YCurrent.nextCurrent()` pour itérer sur les autres capteurs de courant.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YCurrent`, correspondant au premier capteur de courant accessible en ligne, ou `null` si il n'y a pas de capteurs de courant disponibles.

YCurrent.GetSimilarFunctions() YCurrent

YCurrent.GetSimilarFunctions()

Enumère toutes les fonctions de type Current disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

```
dnp  static new string[] GetSimilarFunctions( )
cp   static vector<string> GetSimilarFunctions( )
```

Chaque chaîne retournée peut être passée en argument à la méthode `YCurrent.FindCurrent` pour obtenir une objet permettant d'intéragir avec le module correspondant.

Retourne :

un tableau de chaînes de caractères, contenant les identifiants matériels de chaque fonction disponible trouvée.

current→AdvMode YCurrent

Mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

dnp int **AdvMode**

Modifiable. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

current→AdvertisedValue YCurrent

Courte chaîne de caractères représentant l'état courant de la fonction.

dnp string **AdvertisedValue**

current→Enabled YCurrent

état d'activation de cette mesure.

dnp int **Enabled**

Modifiable. Modifie l'état d'activation de le mesure de courant. Lorsque la mesure de courant AC est désactivée, la mesure se fera uniquement en mode DC, et inversément. Lorsque les deux mesures AC et DC sont activées, le module bascule entre le mode AC et DC en fonction de l'amplitude relative des variations AC par rapport à la valeur moyenne du signal. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

current→FriendlyName YCurrent

Identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

dnp string **FriendlyName**

Le chaîne retournée utilise soit les noms logiques du module et de la fonction si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel de la fonction (par exemple: `MyCustomName.relay1`)

current→FunctionId YCurrent

Identifiant matériel du senseur, sans référence au module.

dnp string **FunctionId**

Par example `relay1`.

current→HardwareId YCurrent

Identifiant matériel unique de la fonction au format `SERIAL.FUNCTIONID`.

dnp string **HardwareId**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par example `RELAYLO1-123456.relay1`).

current→IsOnline YCurrent

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

dnp 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.

current→LogFrequency YCurrent

Fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

dnp string **LogFrequency**

Modifiable. Modifie la fréquence d'enregistrement des mesures dans le datalogger. La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF". Attention il est inutile, voir contre productif, de régler la fréquence d'enregistrement à une valeur supérieure à la fréquence d'échantillonnage native du capteur: ces deux fréquences sont complètement indépendantes. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

current→LogicalName YCurrent

Nom logique de la fonction.

dnp string **LogicalName**

Modifiable. 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.

current→ReportFrequency YCurrent

Fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

dnp string **ReportFrequency**

Modifiable. Modifie la fréquence de notification périodique des valeurs mesurées. La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF". Attention il est inutile, voir contre productif, de régler la fréquence de notification périodique à une valeur supérieure à la fréquence d'échantillonnage native du capteur: ces deux fréquences sont complètement indépendantes. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

current→Resolution YCurrent

Résolution des valeurs mesurées.

dnp double **Resolution**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Modifiable. Modifie la résolution des valeurs physique mesurées. La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

current→SerialNumber YCurrent

Numéro de série du module, préprogrammé en usine.

dnp string **SerialNumber**

current→calibrateFromPoints() YCurrent

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
js   function calibrateFromPoints( rawValues, refValues)
cpp  int calibrateFromPoints( vector<double> rawValues,
                              vector<double> refValues)
m    -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                                : (NSMutableArray*) refValues
pas  LongInt calibrateFromPoints( rawValues: TDoubleArray,
                                  refValues: TDoubleArray): LongInt
vb   procedure calibrateFromPoints( ByVal rawValues As List(Of)
cs   int calibrateFromPoints( List<double> rawValues,
                              List<double> refValues)
java int calibrateFromPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)
uwp  async Task<int> calibrateFromPoints( List<double> rawValues,
                                          List<double> refValues)
py   calibrateFromPoints( rawValues, refValues)
php  function calibrateFromPoints( $rawValues, $refValues)
ts   async calibrateFromPoints( rawValues: number[], refValues: number[]): Promise<number>
es   async calibrateFromPoints( rawValues, refValues)
dnp  int calibrateFromPoints( double[] rawValues,
                              double[] refValues)
cp   int calibrateFromPoints( vector<double> rawValues,
                              vector<double> refValues)
cmd  YCurrent target calibrateFromPoints rawValues refValues
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→clearCache() YCurrent

Invalide le cache.

js	function **clearCache**( )
cpp	void **clearCache**( )
m	-(void) **clearCache**
pas	**clearCache**( )
vb	procedure **clearCache**( )
cs	void **clearCache**( )
java	void **clearCache**( )
py	**clearCache**( )
php	function **clearCache**( )
ts	async **clearCache**( ): Promise<void>
es	async **clearCache**( )

Invalide le cache des valeurs courantes du capteur de courant. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

current→describe() YCurrent

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de courant au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

js	function **describe**( )
cpp	string **describe**( )
m	-(NSString*) **describe**
pas	string **describe**( ): string
vb	function **describe**( ) As String
cs	string **describe**( )
java	String **describe**( )
py	**describe**( )
php	function **describe**( )
ts	async **describe**( ): Promise<string>
es	async **describe**( )

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilsé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou `"unresolved"` sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

Retourne :
une chaîne de caractères décrivant le capteur de courant (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

current→get_advMode() YCurrent
current→advMode()

Retourne le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

```
js    function get_advMode( )
cpp   Y_ADVMODE_enum get_advMode( )
m     -(Y_ADVMODE_enum) advMode
pas   Integer get_advMode( ): Integer
vb    function get_advMode( ) As Integer
cs    int get_advMode( )
java  int get_advMode( )
uwp   async Task<int> get_advMode( )
py    get_advMode( )
php   function get_advMode( )
ts    async get_advMode( ): Promise<YSensor_AdvMode>
es    async get_advMode( )
dnp   int get_advMode( )
cp    int get_advMode( )
cmd   YCurrent target get_advMode
```

Retourne :

une valeur parmi `YCurrent.ADVMODE_IMMEDIATE`, `YCurrent.ADVMODE_PERIOD_AVG`, `YCurrent.ADVMODE_PERIOD_MIN` et `YCurrent.ADVMODE_PERIOD_MAX` représentant le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue)

En cas d'erreur, déclenche une exception ou retourne `YCurrent.ADVMODE_INVALID`.

current→get_advertisedValue() YCurrent
current→advertisedValue()

Retourne la valeur courante du capteur de courant (pas plus de 6 caractères).

```
js    function get_advertisedValue( )
cpp   string get_advertisedValue( )
m     -(NSString*) advertisedValue
pas   string get_advertisedValue( ): string
vb    function get_advertisedValue( ) As String
cs    string get_advertisedValue( )
java  String get_advertisedValue( )
uwp   async Task<string> get_advertisedValue( )
py    get_advertisedValue( )
php   function get_advertisedValue( )
ts    async get_advertisedValue( ): Promise<string>
es    async get_advertisedValue( )
dnp   string get_advertisedValue( )
cp    string get_advertisedValue( )
cmd   YCurrent target get_advertisedValue
```

Retourne :
une chaîne de caractères représentant la valeur courante du capteur de courant (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne YCurrent.ADVERTISEDVALUE_INVALID.

current→get_currentRawValue() YCurrent
current→currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en mA, sous forme de nombre à virgule.

js	function **get_currentRawValue**( )
cpp	double **get_currentRawValue**( )
m	-(double) currentRawValue
pas	double **get_currentRawValue**( ): double
vb	function **get_currentRawValue**( ) As Double
cs	double **get_currentRawValue**( )
java	double **get_currentRawValue**( )
uwp	async Task<double> **get_currentRawValue**( )
py	**get_currentRawValue**( )
php	function **get_currentRawValue**( )
ts	async **get_currentRawValue**( ): Promise<number>
es	async **get_currentRawValue**( )
dnp	double **get_currentRawValue**( )
cp	double **get_currentRawValue**( )
cmd	YCurrent **target get_currentRawValue**

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en mA, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `YCurrent.CURRENTRAWVALUE_INVALID`.

current→get_currentValue() YCurrent
current→currentValue()

Retourne la valeur actuelle du courant, en mA, sous forme de nombre à virgule.

```
js    function get_currentValue( )
cpp   double get_currentValue( )
m     -(double) currentValue
pas   double get_currentValue( ): double
vb    function get_currentValue( ) As Double
cs    double get_currentValue( )
java  double get_currentValue( )
uwp   async Task<double> get_currentValue( )
py    get_currentValue( )
php   function get_currentValue( )
ts    async get_currentValue( ): Promise<number>
es    async get_currentValue( )
dnp   double get_currentValue( )
cp    double get_currentValue( )
cmd   YCurrent target get_currentValue
```

Notez qu'un appel à get_currentValue() ne déclenche pas une mesure dans le module mais retourne simplement la valeur obtenue lors de la dernière mesure. En effet, en interne, chaque senseur Yoctopuce effectue des mesures en continu à une fréquence qui lui est propre.

Si vous rencontrez des problèmes de performances en utilisant la fonction get_currentValue() fréquemment, il vous faudra basculer sur un modèle de callbacks. Pour plus de détails jetez un coup d'oeil au chapitre "programmation avancée" du manuel de votre module.

Retourne :
une valeur numérique représentant la valeur actuelle du courant, en mA, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `YCurrent.CURRENTVALUE_INVALID`.

current→get_dataLogger() YCurrent
current→dataLogger()

Retourne l'objet `YDatalogger` du module qui héberge le senseur.

js	function **get_dataLogger**( )
cpp	YDataLogger* **get_dataLogger**( )
m	-(YDataLogger*) dataLogger
pas	TYDataLogger **get_dataLogger**( ): TYDataLogger
vb	function **get_dataLogger**( ) As YDataLogger
cs	YDataLogger **get_dataLogger**( )
java	YDataLogger **get_dataLogger**( )
uwp	async Task<YDataLogger> **get_dataLogger**( )
py	**get_dataLogger**( )
php	function **get_dataLogger**( )
ts	async **get_dataLogger**( ): Promise<YDataLogger | null>
es	async **get_dataLogger**( )
dnp	YDataLoggerProxy **get_dataLogger**( )
cp	YDataLoggerProxy* **get_dataLogger**( )

Cette méthode retourne un objet qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet `YDatalogger`, ou null en cas d'erreur.

current→get_enabled() YCurrent
current→enabled()

Retourne l'état d'activation de cette mesure.

```
js    function get_enabled( )
cpp   Y_ENABLED_enum get_enabled( )
m     -(Y_ENABLED_enum) enabled
pas   Integer get_enabled( ): Integer
vb    function get_enabled( ) As Integer
cs    int get_enabled( )
java  int get_enabled( )
uwp   async Task<int> get_enabled( )
py    get_enabled( )
php   function get_enabled( )
ts    async get_enabled( ): Promise<YCurrent_Enabled>
es    async get_enabled( )
dnp   int get_enabled( )
cp    int get_enabled( )
cmd   YCurrent target get_enabled
```

Retourne :

soit `YCurrent.ENABLED_FALSE`, soit `YCurrent.ENABLED_TRUE`, selon l'état d'activation de cette mesure

En cas d'erreur, déclenche une exception ou retourne `YCurrent.ENABLED_INVALID`.

current→get_errorMessage() YCurrent
current→errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

```
js      function get_errorMessage( )
cpp     string get_errorMessage( )
m       -(NSString*) errorMessage
pas     string get_errorMessage( ): string
vb      function get_errorMessage( ) As String
cs      string get_errorMessage( )
java    String get_errorMessage( )
py      get_errorMessage( )
php     function get_errorMessage( )
ts      get_errorMessage( ): string
es      get_errorMessage( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :
une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de courant.

current→get_errorType() YCurrent
current→errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de courant.

```
js    function get_errorType( )
cpp   YRETCODE get_errorType( )
m     -(YRETCODE) errorType
pas   YRETCODE get_errorType( ): YRETCODE
vb    function get_errorType( ) As YRETCODE
cs    YRETCODE get_errorType( )
java  int get_errorType( )
py    get_errorType( )
php   function get_errorType( )
ts    get_errorType( ): number
es    get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :
un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de courant.

current→get_friendlyName() YCurrent
current→friendlyName()

Retourne un identifiant global du capteur de courant au format `NOM_MODULE.NOM_FONCTION`.

```
js    function get_friendlyName( )
cpp   string get_friendlyName( )
m     -(NSString*) friendlyName
cs    string get_friendlyName( )
java  String get_friendlyName( )
py    get_friendlyName( )
php   function get_friendlyName( )
ts    async get_friendlyName( ): Promise<string>
es    async get_friendlyName( )
dnp   string get_friendlyName( )
cp    string get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et du capteur de courant si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du capteur de courant (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le capteur de courant en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `YCurrent.FRIENDLYNAME_INVALID`.

current→get_functionDescriptor() YCurrent
current→functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

```
js    function get_functionDescriptor( )
cpp   YFUN_DESCR get_functionDescriptor( )
m     -(YFUN_DESCR) functionDescriptor
pas   YFUN_DESCR get_functionDescriptor( ): YFUN_DESCR
vb    function get_functionDescriptor( ) As YFUN_DESCR
cs    YFUN_DESCR get_functionDescriptor( )
java  String get_functionDescriptor( )
py    get_functionDescriptor( )
php   function get_functionDescriptor( )
ts    async get_functionDescriptor( ): Promise<string>
es    async 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$CLASSNAME$.FUNCTIONDESCRIPTOR_INVALID

current→get_functionId() YCurrent
current→functionId()

Retourne l'identifiant matériel du capteur de courant, sans référence au module.

```
js    function get_functionId( )
cpp   string get_functionId( )
m     -(NSString*) functionId
vb    function get_functionId( ) As String
cs    string get_functionId( )
java  String get_functionId( )
py    get_functionId( )
php   function get_functionId( )
ts    async get_functionId( ): Promise<string>
es    async get_functionId( )
dnp   string get_functionId( )
cp    string get_functionId( )
```

Par example `relay1`.

Retourne :
une chaîne de caractères identifiant le capteur de courant (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `YCurrent.FUNCTIONID_INVALID`.

current→get_hardwareId() YCurrent
current→hardwareId()

Retourne l'identifiant matériel unique du capteur de courant au format `SERIAL.FUNCTIONID`.

js	function **get_hardwareId**( )
cpp	string **get_hardwareId**( )
m	-(NSString*) hardwareId
vb	function **get_hardwareId**( ) As String
cs	string **get_hardwareId**( )
java	String **get_hardwareId**( )
py	**get_hardwareId**( )
php	function **get_hardwareId**( )
ts	async **get_hardwareId**( ): Promise<string>
es	async **get_hardwareId**( )
dnp	string **get_hardwareId**( )
cp	string **get_hardwareId**( )

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de courant (par example `RELAYLO1-123456.relay1`).

Retourne :
une chaîne de caractères identifiant le capteur de courant (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `YCurrent.HARDWAREID_INVALID`.

current→get_highestValue() YCurrent
current→highestValue()

Retourne la valeur maximale observée pour le courant depuis le démarrage du module.

```
js    function get_highestValue( )
cpp   double get_highestValue( )
m     -(double) highestValue
pas   double get_highestValue( ): double
vb    function get_highestValue( ) As Double
cs    double get_highestValue( )
java  double get_highestValue( )
uwp   async Task<double> get_highestValue( )
py    get_highestValue( )
php   function get_highestValue( )
ts    async get_highestValue( ): Promise<number>
es    async get_highestValue( )
dnp   double get_highestValue( )
cp    double get_highestValue( )
cmd   YCurrent target get_highestValue
```

Peut être réinitialisé à une valeur arbitraire grâce à set_highestValue().

Retourne :
une valeur numérique représentant la valeur maximale observée pour le courant depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `YCurrent.HIGHESTVALUE_INVALID`.

current→get_logFrequency() YCurrent
current→logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
js    function get_logFrequency( )
cpp   string get_logFrequency( )
m     -(NSString*) logFrequency
pas   string get_logFrequency( ): string
vb    function get_logFrequency( ) As String
cs    string get_logFrequency( )
java  String get_logFrequency( )
uwp   async Task<string> get_logFrequency( )
py    get_logFrequency( )
php   function get_logFrequency( )
ts    async get_logFrequency( ): Promise<string>
es    async get_logFrequency( )
dnp   string get_logFrequency( )
cp    string get_logFrequency( )
cmd   YCurrent target get_logFrequency
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne `YCurrent.LOGFREQUENCY_INVALID`.

current→get_logicalName() YCurrent
current→logicalName()

Retourne le nom logique du capteur de courant.

```
js    function get_logicalName( )
cpp   string get_logicalName( )
m     -(NSString*) logicalName
pas   string get_logicalName( ): string
vb    function get_logicalName( ) As String
cs    string get_logicalName( )
java  String get_logicalName( )
uwp   async Task<string> get_logicalName( )
py    get_logicalName( )
php   function get_logicalName( )
ts    async get_logicalName( ): Promise<string>
es    async get_logicalName( )
dnp   string get_logicalName( )
cp    string get_logicalName( )
cmd   YCurrent target get_logicalName
```

Retourne :
une chaîne de caractères représentant le nom logique du capteur de courant.

En cas d'erreur, déclenche une exception ou retourne `YCurrent.LOGICALNAME_INVALID`.

current→get_lowestValue() current→lowestValue() YCurrent

Retourne la valeur minimale observée pour le courant depuis le démarrage du module.

js	function **get_lowestValue**( )
cpp	double **get_lowestValue**( )
m	-(double) lowestValue
pas	double **get_lowestValue**( ): double
vb	function **get_lowestValue**( ) As Double
cs	double **get_lowestValue**( )
java	double **get_lowestValue**( )
uwp	async Task<double> **get_lowestValue**( )
py	**get_lowestValue**( )
php	function **get_lowestValue**( )
ts	async **get_lowestValue**( ): Promise<number>
es	async **get_lowestValue**( )
dnp	double **get_lowestValue**( )
cp	double **get_lowestValue**( )
cmd	YCurrent **target get_lowestValue**

Peut être réinitialisé à une valeur arbitraire grâce à set_lowestValue().

Retourne :
une valeur numérique représentant la valeur minimale observée pour le courant depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `YCurrent.LOWESTVALUE_INVALID`.

current→get_module() YCurrent
current→module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

js	function **get_module**( )
cpp	YModule * **get_module**( )
m	-(YModule*) module
pas	TYModule **get_module**( ): TYModule
vb	function **get_module**( ) As YModule
cs	YModule **get_module**( )
java	YModule **get_module**( )
py	**get_module**( )
php	function **get_module**( )
ts	async **get_module**( ): Promise<YModule>
es	async **get_module**( )
dnp	YModuleProxy **get_module**( )
cp	YModuleProxy * **get_module**( )

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :
une instance de `YModule`

current→get_module_async() YCurrent
current→module_async()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js  function get_module_async( callback, context)
```

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_recordedData() YCurrent
current→recordedData()

Retourne un objet `YDataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
js    function get_recordedData( startTime, endTime)
cpp   YDataSet get_recordedData( double startTime, double endTime)
m     -(YDataSet*) recordedData : (double) startTime
                                : (double) endTime
pas   TYDataSet get_recordedData( startTime: double, endTime: double): TYDataSet
vb    function get_recordedData( ByVal startTime As Double,
                                 ByVal endTime As Double) As YDataSet
cs    YDataSet get_recordedData( double startTime, double endTime)
java  YDataSet get_recordedData( double startTime, double endTime)
uwp   async Task<YDataSet> get_recordedData( double startTime,
                                             double endTime)
py    get_recordedData( startTime, endTime)
php   function get_recordedData( $startTime, $endTime)
ts    async get_recordedData( startTime: number, endTime: number): Promise<YDataSet>
es    async get_recordedData( startTime, endTime)
dnp   YDataSetProxy get_recordedData( double startTime,
                                      double endTime)
cp    YDataSetProxy* get_recordedData( double startTime,
                                       double endTime)
cmd   YCurrent target get_recordedData startTime endTime
```

Veuillez vous référer à la documentation de la classe `YDataSet` pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets `YDataSet` ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la find de l'intercalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de `YDataSet`, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

current→get_reportFrequency() YCurrent
current→reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
js    function get_reportFrequency( )
cpp   string get_reportFrequency( )
m     -(NSString*) reportFrequency
pas   string get_reportFrequency( ): string
vb    function get_reportFrequency( ) As String
cs    string get_reportFrequency( )
java  String get_reportFrequency( )
uwp   async Task<string> get_reportFrequency( )
py    get_reportFrequency( )
php   function get_reportFrequency( )
ts    async get_reportFrequency( ): Promise<string>
es    async get_reportFrequency( )
dnp   string get_reportFrequency( )
cp    string get_reportFrequency( )
cmd   YCurrent target get_reportFrequency
```

Retourne :

une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne YCurrent.REPORTFREQUENCY_INVALID.

current→get_resolution() YCurrent
current→resolution()

Retourne la résolution des valeurs mesurées.

```
js    function get_resolution( )
cpp   double get_resolution( )
m     -(double) resolution
pas   double get_resolution( ): double
vb    function get_resolution( ) As Double
cs    double get_resolution( )
java  double get_resolution( )
uwp   async Task<double> get_resolution( )
py    get_resolution( )
php   function get_resolution( )
ts    async get_resolution( ): Promise<number>
es    async get_resolution( )
dnp   double get_resolution( )
cp    double get_resolution( )
cmd   YCurrent target get_resolution
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `YCurrent.RESOLUTION_INVALID`.

current→get_sensorState() YCurrent
current→sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

```
js    function get_sensorState( )
cpp   int get_sensorState( )
m     -(int) sensorState
pas   LongInt get_sensorState( ): LongInt
vb    function get_sensorState( ) As Integer
cs    int get_sensorState( )
java  int get_sensorState( )
uwp   async Task<int> get_sensorState( )
py    get_sensorState( )
php   function get_sensorState( )
ts    async get_sensorState( ): Promise<number>
es    async get_sensorState( )
dnp   int get_sensorState( )
cp    int get_sensorState( )
cmd   YCurrent target get_sensorState
```

Retourne :

un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne `YCurrent.SENSORSTATE_INVALID`.

current→get_serialNumber() YCurrent
current→serialNumber()

Retourne le numéro de série du module, préprogrammé en usine.

```
js    function get_serialNumber( )
cpp   string get_serialNumber( )
m     -(NSString*) serialNumber
pas   string get_serialNumber( ): string
vb    function get_serialNumber( ) As String
cs    string get_serialNumber( )
java  String get_serialNumber( )
uwp   async Task<string> get_serialNumber( )
py    get_serialNumber( )
php   function get_serialNumber( )
ts    async get_serialNumber( ): Promise<string>
es    async get_serialNumber( )
dnp   string get_serialNumber( )
cp    string get_serialNumber( )
cmd   YCurrent target get_serialNumber
```

Retourne :
: une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine.

En cas d'erreur, déclenche une exception ou retourne YFunction.SERIALNUMBER_INVALID.

current→get_unit() YCurrent
current→unit()

Retourne l'unité dans laquelle le courant est exprimée.

```
js    function get_unit( )
cpp   string get_unit( )
m     -(NSString*) unit
pas   string get_unit( ): string
vb    function get_unit( ) As String
cs    string get_unit( )
java  String get_unit( )
uwp   async Task<string> get_unit( )
py    get_unit( )
php   function get_unit( )
ts    async get_unit( ): Promise<string>
es    async get_unit( )
dnp   string get_unit( )
cp    string get_unit( )
cmd   YCurrent target get_unit
```

Retourne :
une chaîne de caractères représentant l'unité dans laquelle le courant est exprimée

En cas d'erreur, déclenche une exception ou retourne `YCurrent.UNIT_INVALID`.

current→get_userData() YCurrent
current→userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

js	function **get_userData**( )
cpp	void * **get_userData**( )
m	-(id) userData
pas	Tobject **get_userData**( ): Tobject
vb	function **get_userData**( ) As Object
cs	object **get_userData**( )
java	Object **get_userData**( )
py	**get_userData**( )
php	function **get_userData**( )
ts	async **get_userData**( ): Promise<object|null>
es	async **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() YCurrent

Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.

```
js    function isOnline( )
cpp   bool isOnline( )
m     -(BOOL) isOnline
pas   boolean isOnline( ): boolean
vb    function isOnline( ) As Boolean
cs    bool isOnline( )
java  boolean isOnline( )
py    isOnline( )
php   function isOnline( )
ts    async isOnline( ): Promise<boolean>
es    async isOnline( )
dnp   bool isOnline( )
cp    bool isOnline( )
```

Si les valeurs des attributs en cache du capteur de courant sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si le capteur de courant est joignable, `false` sinon

current→isOnline_async() YCurrent

Vérifie si le module hébergeant le capteur de courant est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du capteur de courant sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

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 machine virtuelle Javascript avec une attente active.

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→isReadOnly() YCurrent

Test si la fonction est en lecture seule.

cpp	bool **isReadOnly**( )
m	-(bool) **isReadOnly**
pas	boolean **isReadOnly**( ): boolean
vb	function **isReadOnly**( ) As Boolean
cs	bool **isReadOnly**( )
java	boolean **isReadOnly**( )
uwp	async Task<bool> **isReadOnly**( )
py	**isReadOnly**( )
php	function **isReadOnly**( )
ts	async **isReadOnly**( ): Promise<boolean>
es	async **isReadOnly**( )
dnp	bool **isReadOnly**( )
cp	bool **isReadOnly**( )
cmd	YCurrent **target isReadOnly**

Retourne vrais si la fonction est protégé en ecriture ou que la fontion n'est pas disponible.

Retourne :
`true` si la fonction est protégé en ecriture ou que la fontion n'est pas disponible

current→isSensorReady() YCurrent

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

```
cmd YCurrent target isSensorReady
```

Retourne faux si le module n'est pas joignable, ou que le capteur n'a pas de mesure actuelle à communiquer. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si le capteur dispose d'une mesure actuelle, `false` sinon

current→load() YCurrent

Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.

```
js    function load( msValidity)
cpp   YRETCODE load( int msValidity)
m     -(YRETCODE) load : (u64) msValidity
pas   YRETCODE load( msValidity: u64): YRETCODE
vb    function load( ByVal msValidity As Long) As YRETCODE
cs    YRETCODE load( ulong msValidity)
java  int load( long msValidity)
py    load( msValidity)
php   function load( $msValidity)
ts    async load( msValidity: number): Promise<number>
es    async load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→loadAttribute() YCurrent

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
js    function loadAttribute( attrName)
cpp   string loadAttribute( string attrName)
m     -(NSString*) loadAttribute : (NSString*) attrName
pas   string loadAttribute( attrName: string): string
vb    function loadAttribute( ByVal attrName As String) As String
cs    string loadAttribute( string attrName)
java  String loadAttribute( String attrName)
uwp   async Task<string> loadAttribute( string attrName)
py    loadAttribute( attrName)
php   function loadAttribute( $attrName)
ts    async loadAttribute( attrName: string): Promise<string>
es    async loadAttribute( attrName)
dnp   string loadAttribute( string attrName)
cp    string loadAttribute( string attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

current→loadCalibrationPoints() YCurrent

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
js   function loadCalibrationPoints( rawValues, refValues)
cpp  int loadCalibrationPoints( vector<double> rawValues,
                                vector<double> refValues)
m    -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                                  : (NSMutableArray*) refValues
pas  LongInt loadCalibrationPoints( var rawValues: TDoubleArray,
                                    var refValues: TDoubleArray): LongInt
vb   procedure loadCalibrationPoints( ByVal rawValues As List(Of)
cs   int loadCalibrationPoints( List<double> rawValues,
                                List<double> refValues)
java int loadCalibrationPoints( ArrayList<Double> rawValues,
                                ArrayList<Double> refValues)
uwp  async Task<int> loadCalibrationPoints( List<double> rawValues,
                                            List<double> refValues)
py   loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
ts   async loadCalibrationPoints( rawValues: number[], refValues: number[]): Promise<number>
es   async loadCalibrationPoints( rawValues, refValues)
cmd  YCurrent target loadCalibrationPoints rawValues refValues
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→load_async() YCurrent

Met en cache les valeurs courantes du capteur de courant, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
```

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 machine virtuelle Javascript avec une attente active.

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→muteValueCallbacks() YCurrent

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

js	function **muteValueCallbacks**( )
cpp	int **muteValueCallbacks**( )
m	-(int) **muteValueCallbacks**
pas	LongInt **muteValueCallbacks**( ): LongInt
vb	function **muteValueCallbacks**( ) As Integer
cs	int **muteValueCallbacks**( )
java	int **muteValueCallbacks**( )
uwp	async Task<int> **muteValueCallbacks**( )
py	**muteValueCallbacks**( )
php	function **muteValueCallbacks**( )
ts	async **muteValueCallbacks**( ): Promise<number>
es	async **muteValueCallbacks**( )
dnp	int **muteValueCallbacks**( )
cp	int **muteValueCallbacks**( )
cmd	YCurrent **target muteValueCallbacks**

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→nextCurrent() YCurrent

Continue l'énumération des capteurs de courant commencée à l'aide de `yFirstCurrent()`. Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les capteurs de courant sont retournés.

js	function **nextCurrent**( )
cpp	YCurrent * **nextCurrent**( )
m	-(nullable YCurrent*) **nextCurrent**
pas	TYCurrent **nextCurrent**( ): TYCurrent
vb	function **nextCurrent**( ) As YCurrent
cs	YCurrent **nextCurrent**( )
java	YCurrent **nextCurrent**( )
uwp	YCurrent **nextCurrent**( )
py	**nextCurrent**( )
php	function **nextCurrent**( )
ts	**nextCurrent**( ): YCurrent | null
es	**nextCurrent**( )

Si vous souhaitez retrouver un capteur de courant spécifique, utilisez `Current.findCurrent()` avec un hardwareID ou un nom logique.

Retourne :
un pointeur sur un objet `YCurrent` accessible en ligne, ou `null` lorsque l'énumération est terminée.

current→registerTimedReportCallback() YCurrent

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

js	function **registerTimedReportCallback**(**callback**)
cpp	int **registerTimedReportCallback**(YCurrentTimedReportCallback **callback**)
m	-(int) **registerTimedReportCallback** : (YCurrentTimedReportCallback _Nullable) **callback**
pas	LongInt **registerTimedReportCallback**(**callback**: TYCurrentTimedReportCallback): LongInt
vb	function **registerTimedReportCallback**(ByVal **callback** As YCurrentTimedReportCallback) As Integer
cs	int **registerTimedReportCallback**(TimedReportCallback **callback**)
java	int **registerTimedReportCallback**(TimedReportCallback **callback**)
uwp	async Task<int> **registerTimedReportCallback**(TimedReportCallback **callback**)
py	**registerTimedReportCallback**(**callback**)
php	function **registerTimedReportCallback**(**$callback**)
ts	async **registerTimedReportCallback**(**callback**: YCurrentTimedReportCallback | null): Promise<number>
es	async **registerTimedReportCallback**(**callback**)

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

current→registerValueCallback() YCurrent

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
js    function registerValueCallback( callback)
cpp   int registerValueCallback( YCurrentValueCallback callback)
m     -(int) registerValueCallback : (YCurrentValueCallback _Nullable) callback
pas   LongInt registerValueCallback( callback: TYCurrentValueCallback): LongInt
vb    function registerValueCallback( ByVal callback As YCurrentValueCallback) As Integer
cs    int registerValueCallback( ValueCallback callback)
java  int registerValueCallback( UpdateCallback callback)
uwp   async Task<int> registerValueCallback( ValueCallback callback)
py    registerValueCallback( callback)
php   function registerValueCallback( $callback)
ts    async registerValueCallback( callback: YCurrentValueCallback | null): Promise<number>
es    async registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

current→set_advMode() YCurrent
current→setAdvMode()

Modifie le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

```
js    function set_advMode( newval)
cpp   int set_advMode( Y_ADVMODE_enum newval)
m     -(int) setAdvMode : (Y_ADVMODE_enum) newval
pas   integer set_advMode( newval: Integer): integer
vb    function set_advMode( ByVal newval As Integer) As Integer
cs    int set_advMode( int newval)
java  int set_advMode( int newval)
uwp   async Task<int> set_advMode( int newval)
py    set_advMode( newval)
php   function set_advMode( $newval)
ts    async set_advMode( newval: YSensor_AdvMode): Promise<number>
es    async set_advMode( newval)
dnp   int set_advMode( int newval)
cp    int set_advMode( int newval)
cmd   YCurrent target set_advMode newval
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur parmi `YCurrent.ADVMODE_IMMEDIATE`, `YCurrent.ADVMODE_PERIOD_AVG`, `YCurrent.ADVMODE_PERIOD_MIN` et `YCurrent.ADVMODE_PERIOD_MAX` représentant le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue)

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_enabled() YCurrent
current→setEnabled()

Modifie l'état d'activation de le mesure de courant.

```
js    function set_enabled( newval)
cpp   int set_enabled( Y_ENABLED_enum newval)
m     -(int) setEnabled : (Y_ENABLED_enum) newval
pas   integer set_enabled( newval: Integer): integer
vb    function set_enabled( ByVal newval As Integer) As Integer
cs    int set_enabled( int newval)
java  int set_enabled( int newval)
uwp   async Task<int> set_enabled( int newval)
py    set_enabled( newval)
php   function set_enabled( $newval)
ts    async set_enabled( newval: YCurrent_Enabled): Promise<number>
es    async set_enabled( newval)
dnp   int set_enabled( int newval)
cp    int set_enabled( int newval)
cmd   YCurrent target set_enabled newval
```

Lorsque la mesure de courant AC est désactivée, la mesure se fera uniquement en mode DC, et inversément. Lorsque les deux mesures AC et DC sont activées, le module bascule entre le mode AC et DC en fonction de l'amplitude relative des variations AC par rapport à la valeur moyenne du signal. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval soit `YCurrent.ENABLED_FALSE`, soit `YCurrent.ENABLED_TRUE`, selon l'état d'activation de le mesure 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_highestValue() YCurrent
current→setHighestValue()

Modifie la mémoire de valeur maximale observée.

```
js    function set_highestValue( newval)
cpp   int set_highestValue( double newval)
m     -(int) setHighestValue : (double) newval
pas   integer set_highestValue( newval: double): integer
vb    function set_highestValue( ByVal newval As Double) As Integer
cs    int set_highestValue( double newval)
java  int set_highestValue( double newval)
uwp   async Task<int> set_highestValue( double newval)
py    set_highestValue( newval)
php   function set_highestValue( $newval)
ts    async set_highestValue( newval: number): Promise<number>
es    async set_highestValue( newval)
dnp   int set_highestValue( double newval)
cp    int set_highestValue( double newval)
cmd   YCurrent target set_highestValue newval
```

Utile pour réinitialiser la valeur renvoyée par get_highestValue().

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→set_logFrequency() YCurrent
current→setLogFrequency()

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
js      function set_logFrequency( newval)
cpp     int set_logFrequency( string newval)
m       -(int) setLogFrequency : (NSString*) newval
pas     integer set_logFrequency( newval: string): integer
vb      function set_logFrequency( ByVal newval As String) As Integer
cs      int set_logFrequency( string newval)
java    int set_logFrequency( String newval)
uwp     async Task<int> set_logFrequency( string newval)
py      set_logFrequency( newval)
php     function set_logFrequency( $newval)
ts      async set_logFrequency( newval: string): Promise<number>
es      async set_logFrequency( newval)
dnp     int set_logFrequency( string newval)
cp      int set_logFrequency( string newval)
cmd     YCurrent target set_logFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF". Attention il est inutile, voir contre productif, de régler la fréquence d'enregistrement à une valeur supérieure à la fréquence d'échantillonnage native du capteur: ces deux fréquences sont complètement indépendantes. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→set_logicalName() YCurrent
current→setLogicalName()

Modifie le nom logique du capteur de courant.

```
js    function set_logicalName( newval)
cpp   int set_logicalName( string newval)
m     -(int) setLogicalName : (NSString*) newval
pas   integer set_logicalName( newval: string): integer
vb    function set_logicalName( ByVal newval As String) As Integer
cs    int set_logicalName( string newval)
java  int set_logicalName( String newval)
uwp   async Task<int> set_logicalName( string newval)
py    set_logicalName( newval)
php   function set_logicalName( $newval)
ts    async set_logicalName( newval: string): Promise<number>
es    async set_logicalName( newval)
dnp   int set_logicalName( string newval)
cp    int set_logicalName( string newval)
cmd   YCurrent target set_logicalName newval
```

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de courant.

Retourne :

`YAPI.SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→set_lowestValue() YCurrent
current→setLowestValue()

Modifie la mémoire de valeur minimale observée.

```
js    function set_lowestValue( newval)
cpp   int set_lowestValue( double newval)
m     -(int) setLowestValue : (double) newval
pas   integer set_lowestValue( newval: double): integer
vb    function set_lowestValue( ByVal newval As Double) As Integer
cs    int set_lowestValue( double newval)
java  int set_lowestValue( double newval)
uwp   async Task<int> set_lowestValue( double newval)
py    set_lowestValue( newval)
php   function set_lowestValue( $newval)
ts    async set_lowestValue( newval: number): Promise<number>
es    async set_lowestValue( newval)
dnp   int set_lowestValue( double newval)
cp    int set_lowestValue( double newval)
cmd   YCurrent target set_lowestValue newval
```

Utile pour réinitialiser la valeur renvoyée par get_lowestValue().

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→set_reportFrequency() YCurrent
current→setReportFrequency()

Modifie la fréquence de notification périodique des valeurs mesurées.

```
js    function set_reportFrequency( newval)
cpp   int set_reportFrequency( string newval)
m     -(int) setReportFrequency : (NSString*) newval
pas   integer set_reportFrequency( newval: string): integer
vb    function set_reportFrequency( ByVal newval As String) As Integer
cs    int set_reportFrequency( string newval)
java  int set_reportFrequency( String newval)
uwp   async Task<int> set_reportFrequency( string newval)
py    set_reportFrequency( newval)
php   function set_reportFrequency( $newval)
ts    async set_reportFrequency( newval: string): Promise<number>
es    async set_reportFrequency( newval)
dnp   int set_reportFrequency( string newval)
cp    int set_reportFrequency( string newval)
cmd   YCurrent target set_reportFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF". Attention il est inutile, voir contre productif, de régler la fréquence de notification périodique à une valeur supérieure à la fréquence d'échantillonnage native du capteur: ces deux fréquences sont complètement indépendantes. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→set_resolution() YCurrent
current→setResolution()

Modifie la résolution des valeurs physique mesurées.

```
js    function set_resolution( newval)
cpp   int set_resolution( double newval)
m     -(int) setResolution : (double) newval
pas   integer set_resolution( newval: double): integer
vb    function set_resolution( ByVal newval As Double) As Integer
cs    int set_resolution( double newval)
java  int set_resolution( double newval)
uwp   async Task<int> set_resolution( double newval)
py    set_resolution( newval)
php   function set_resolution( $newval)
ts    async set_resolution( newval: number): Promise<number>
es    async set_resolution( newval)
dnp   int set_resolution( double newval)
cp    int set_resolution( double newval)
cmd   YCurrent target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→set_userData() YCurrent
current→setUserData()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

```
js    function set_userData( data)
cpp   void set_userData( void * data)
m     -(void) setUserData : (id) data
pas   set_userData( data: Tobject)
vb    procedure set_userData( ByVal data As Object)
cs    void set_userData( object data)
java  void set_userData( Object data)
py    set_userData( data)
php   function set_userData( $data)
ts    async set_userData( data: object|null): Promise<void>
es    async set_userData( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

current→startDataLogger() YCurrent

Démarre l'enregistreur de données du module.

js	function **startDataLogger**( )
cpp	int **startDataLogger**( )
m	-(int) **startDataLogger**
pas	LongInt **startDataLogger**( ): LongInt
vb	function **startDataLogger**( ) As Integer
cs	int **startDataLogger**( )
java	int **startDataLogger**( )
uwp	async Task<int> **startDataLogger**( )
py	**startDataLogger**( )
php	function **startDataLogger**( )
ts	async **startDataLogger**( ): Promise<number>
es	async **startDataLogger**( )
dnp	int **startDataLogger**( )
cp	int **startDataLogger**( )
cmd	YCurrent **target startDataLogger**

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
`YAPI.SUCCESS` si l'opération se déroule sans erreur.

current→stopDataLogger() YCurrent

Arrête l'enregistreur de données du module.

js	function **stopDataLogger**( )
cpp	int **stopDataLogger**( )
m	-(int) **stopDataLogger**
pas	LongInt **stopDataLogger**( ): LongInt
vb	function **stopDataLogger**( ) As Integer
cs	int **stopDataLogger**( )
java	int **stopDataLogger**( )
uwp	async Task<int> **stopDataLogger**( )
py	**stopDataLogger**( )
php	function **stopDataLogger**( )
ts	async **stopDataLogger**( ): Promise<number>
es	async **stopDataLogger**( )
dnp	int **stopDataLogger**( )
cp	int **stopDataLogger**( )
cmd	YCurrent **target stopDataLogger**

Retourne :
`YAPI.SUCCESS` si l'opération se déroule sans erreur.

current→unmuteValueCallbacks() YCurrent

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

js	function **unmuteValueCallbacks**( )
cpp	int **unmuteValueCallbacks**( )
m	-(int) **unmuteValueCallbacks**
pas	LongInt **unmuteValueCallbacks**( ): LongInt
vb	function **unmuteValueCallbacks**( ) As Integer
cs	int **unmuteValueCallbacks**( )
java	int **unmuteValueCallbacks**( )
uwp	async Task<int> **unmuteValueCallbacks**( )
py	**unmuteValueCallbacks**( )
php	function **unmuteValueCallbacks**( )
ts	async **unmuteValueCallbacks**( ): Promise<number>
es	async **unmuteValueCallbacks**( )
dnp	int **unmuteValueCallbacks**( )
cp	int **unmuteValueCallbacks**( )
cmd	YCurrent **target unmuteValueCallbacks**

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

current→wait_async() YCurrent

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js  function wait_async( callback, context)
ts  wait_async( callback: Function, context: object)
es  wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout.

24.6. La classe YTemperature

Interface pour intéragir avec les capteurs de température, disponibles par exemple dans le Yocto-Meteo-V2, le Yocto-PT100, le Yocto-Temperature et le Yocto-Thermocouple

La classe `YTemperature` permet de lire et de configurer les capteurs de température Yoctopuce. Elle hérite de la classe `YSensor` toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, elle permet de configurer les paramètres spécifiques de certains types de capteur (type de connection, table d'étalonnage).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_temperature.js'></script>
cpp	#include "yocto_temperature.h"
m	#import "yocto_temperature.h"
pas	uses yocto_temperature;
vb	yocto_temperature.vb
cs	yocto_temperature.cs
java	import com.yoctopuce.YoctoAPI.YTemperature;
uwp	import com.yoctopuce.YoctoAPI.YTemperature;
py	from yocto_temperature import *
php	require_once('yocto_temperature.php');
ts	in HTML: import { YTemperature } from '../../dist/esm/yocto_temperature.js'; in Node.js: import { YTemperature } from 'yoctolib-cjs/yocto_temperature.js';
es	in HTML: <script src="../../lib/yocto_temperature.js"></script> in node.js: require('yoctolib-es2017/yocto_temperature.js');
dnp	import YoctoProxyAPI.YTemperatureProxy
cp	#include "yocto_temperature_proxy.h"
vi	YTemperature.vi
ml	import YoctoProxyAPI.YTemperatureProxy

Fonction globales

YTemperature.FindTemperature(**func**)

Permet de retrouver un capteur de température d'après un identifiant donné.

YTemperature.FindTemperatureInContext(**yctx**, **func**)

Permet de retrouver un capteur de température d'après un identifiant donné dans un Context YAPI.

YTemperature.FirstTemperature()

Commence l'énumération des capteurs de température accessibles par la librairie.

YTemperature.FirstTemperatureInContext(**yctx**)

Commence l'énumération des capteurs de température accessibles par la librairie.

YTemperature.GetSimilarFunctions()

Enumère toutes les fonctions de type Temperature disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

Propriétés des objets `YTemperatureProxy`

temperature→AdvMode *[modifiable]*

Mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

temperature→AdvertisedValue *[lecture seule]*

Courte chaîne de caractères représentant l'état courant de la fonction.

temperature→FriendlyName *[lecture seule]*

Identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

temperature→FunctionId *[lecture seule]*

Identifiant matériel du senseur, sans référence au module.

temperature→HardwareId *[lecture seule]*

Identifiant matériel unique de la fonction au format `SERIAL.FUNCTIONID`.

temperature→IsOnline *[lecture seule]*

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

temperature→LogFrequency *[modifiable]*

Fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

temperature→LogicalName *[modifiable]*

Nom logique de la fonction.

temperature→ReportFrequency *[modifiable]*

Fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

temperature→Resolution *[modifiable]*

Résolution des valeurs mesurées.

temperature→SensorType *[modifiable]*

Type de capteur de température utilisé par le module **Modifiable**.

temperature→SerialNumber *[lecture seule]*

Numéro de série du module, préprogrammé en usine.

temperature→SignalUnit *[lecture seule]*

Unité du signal électrique utilisé par le capteur.

Méthodes des objets `YTemperature`

temperature→calibrateFromPoints(rawValues, refValues)

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

temperature→clearCache()

Invalide le cache.

temperature→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de température au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

temperature→get_advMode()

Retourne le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

temperature→get_advertisedValue()

Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

temperature→get_currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés Celsius, sous forme de nombre à virgule.

temperature→get_currentValue()

Retourne la valeur actuelle de la température, en degrés Celsius, sous forme de nombre à virgule.

temperature→get_dataLogger()

Retourne l'objet `YDatalogger` du module qui héberge le senseur.

temperature→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

temperature→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

temperature→get_friendlyName()

Retourne un identifiant global du capteur de température au format `NOM_MODULE.NOM_FONCTION`.

temperature→get_functionDescriptor()

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

temperature→get_functionId()

Retourne l'identifiant matériel du capteur de température, sans référence au module.

temperature→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de température au format `SERIAL.FUNCTIONID`.

temperature→get_highestValue()

Retourne la valeur maximale observée pour la température depuis le démarrage du module.

temperature→get_logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

temperature→get_logicalName()

Retourne le nom logique du capteur de température.

temperature→get_lowestValue()

Retourne la valeur minimale observée pour la température depuis le démarrage du module.

temperature→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

temperature→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

temperature→get_recordedData(startTime, endTime)

Retourne un objet `YDataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

temperature→get_reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

temperature→get_resolution()

Retourne la résolution des valeurs mesurées.

temperature→get_sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

temperature→get_sensorType()

Retourne le type de capteur de température utilisé par le module

temperature→get_serialNumber()

Retourne le numéro de série du module, préprogrammé en usine.

temperature→get_signalUnit()

Retourne l'unité du signal électrique utilisé par le capteur.

temperature→get_signalValue()

Retourne la valeur actuelle du signal électrique mesuré par le capteur.

temperature→get_unit()

Retourne l'unité dans laquelle la température est exprimée.

temperature→get_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

temperature→isOnline()

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

temperature→isOnline_async(callback, context)

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

temperature→isReadOnly()

Test si la fonction est en lecture seule.

temperature→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

temperature→load(msValidity)

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

temperature→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

temperature→loadCalibrationPoints(rawValues, refValues)

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

temperature→loadThermistorResponseTable(tempValues, resValues)

Récupère la table de réponse d'un thermistor précédemment enregistrée à l'aide de la fonction `set_thermistorResponseTable`.

temperature→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

temperature→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

temperature→nextTemperature()

Continue l'énumération des capteurs de température commencée à l'aide de `yFirstTemperature()`
Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les capteurs de température sont retournés.

temperature→registerTimedReportCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

temperature→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

temperature→set_advMode(newval)

Modifie le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

temperature→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

temperature→set_logFrequency(newval)

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

temperature→set_logicalName(newval)

Modifie le nom logique du capteur de température.

temperature→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

temperature→set_ntcParameters(res25, beta)

Configure les paramètres d'un thermistor NTC pour calculer correctement la température sur la base de la résistance mesurée.

temperature→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

temperature→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

temperature→set_sensorType(newval)

Modifie le type de senseur utilisé par le module.

temperature→set_thermistorResponseTable(tempValues, resValues)

Enregistre la table de réponse d'un thermistor, afin de pouvoir interpoler la température sur la base de la résistance mesurée.

temperature→set_unit(newval)

Modifie l'unité dans laquelle la température mesurée est exprimée.

temperature→set_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

temperature→startDataLogger()

Démarre l'enregistreur de données du module.

temperature→stopDataLogger()

Arrête l'enregistreur de données du module.

temperature→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

temperature→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YTemperature.FindTemperature() YTemperature

YTemperature.FindTemperature()

Permet de retrouver un capteur de température d'après un identifiant donné.

```
js    function yFindTemperature( func)
cpp   YTemperature* FindTemperature( string func)
m     +(YTemperature*) FindTemperature : (NSString*) func
pas   TYTemperature yFindTemperature( func: string): TYTemperature
vb    function FindTemperature( ByVal func As String) As YTemperature
cs    static YTemperature FindTemperature( string func)
java  static YTemperature FindTemperature( String func)
uwp   static YTemperature FindTemperature( string func)
py    FindTemperature( func)
php   function FindTemperature( $func)
ts    static FindTemperature( func: string): YTemperature
es    static FindTemperature( func)
dnp   static YTemperatureProxy FindTemperature( string func)
cp    static YTemperatureProxy * FindTemperature( string func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de température soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YTemperature.isOnline()` pour tester si le capteur de température est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Si un appel à la méthode is_online() de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler registerHub() à l'initialisation de de l'application.

Paramètres :

func une chaîne de caractères qui référence le capteur de température sans ambiguïté, par exemple `METEOMK2.temperature`.

Retourne :

un objet de classe `YTemperature` qui permet ensuite de contrôler le capteur de température.

YTemperature.FindTemperatureInContext() YTemperature

YTemperature.FindTemperatureInContext()

Permet de retrouver un capteur de température d'après un identifiant donné dans un Context YAPI.

```
java  static YTemperature FindTemperatureInContext( YAPIContext yctx,
                                                    String func)
uwp   static YTemperature FindTemperatureInContext( YAPIContext yctx,
                                                    string func)
ts    static FindTemperatureInContext( yctx: YAPIContext, func: string): YTemperature
es    static FindTemperatureInContext( yctx, func)
```

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le capteur de température soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YTemperature.isOnline()` pour tester si le capteur de température est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence le capteur de température sans ambiguïté, par exemple `METEOMK2.temperature`.

Retourne :

un objet de classe `YTemperature` qui permet ensuite de contrôler le capteur de température.

YTemperature.FirstTemperature() YTemperature

YTemperature.FirstTemperature()

Commence l'énumération des capteurs de température accessibles par la librairie.

js	function **yFirstTemperature**( )
cpp	YTemperature * **FirstTemperature**( )
m	+(YTemperature*) **FirstTemperature**
pas	TYTemperature **yFirstTemperature**( ): TYTemperature
vb	function **FirstTemperature**( ) As YTemperature
cs	static YTemperature **FirstTemperature**( )
java	static YTemperature **FirstTemperature**( )
uwp	static YTemperature **FirstTemperature**( )
py	**FirstTemperature**( )
php	function **FirstTemperature**( )
ts	static **FirstTemperature**( ): YTemperature | null
es	static **FirstTemperature**( )

Utiliser la fonction `YTemperature.nextTemperature()` pour itérer sur les autres capteurs de température.

Retourne :

un pointeur sur un objet `YTemperature`, correspondant au premier capteur de température accessible en ligne, ou `null` si il n'y a pas de capteurs de température disponibles.

YTemperature.FirstTemperatureInContext() YTemperature YTemperature.FirstTemperatureInContext()

Commence l'énumération des capteurs de température accessibles par la librairie.

```
java  static YTemperature FirstTemperatureInContext( YAPIContext yctx)
uwp   static YTemperature FirstTemperatureInContext( YAPIContext yctx)
ts    static FirstTemperatureInContext( yctx: YAPIContext): YTemperature | null
es    static FirstTemperatureInContext( yctx)
```

Utiliser la fonction `YTemperature.nextTemperature()` pour itérer sur les autres capteurs de température.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YTemperature`, correspondant au premier capteur de température accessible en ligne, ou `null` si il n'y a pas de capteurs de température disponibles.

YTemperature.GetSimilarFunctions() YTemperature

YTemperature.GetSimilarFunctions()

Enumère toutes les fonctions de type Temperature disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

dnp	static new string[] **GetSimilarFunctions**( )
cp	static vector<string> **GetSimilarFunctions**( )

Chaque chaîne retournée peut être passée en argument à la méthode `YTemperature.FindTemperature` pour obtenir une objet permettant d'intéragir avec le module correspondant.

Retourne :
un tableau de chaînes de caractères, contenant les identifiants matériels de chaque fonction disponible trouvée.

temperature→AdvMode YTemperature

Mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

dnp int **AdvMode**

Modifiable. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

temperature→AdvertisedValue YTemperature

Courte chaîne de caractères représentant l'état courant de la fonction.

dnp string **AdvertisedValue**

temperature→FriendlyName YTemperature

Identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

dnp string **FriendlyName**

Le chaîne retournée utilise soit les noms logiques du module et de la fonction si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel de la fonction (par exemple: `MyCustomName.relay1`)

temperature→FunctionId YTemperature

Identifiant matériel du senseur, sans référence au module.

dnp string **FunctionId**

Par example `relay1`.

temperature→HardwareId YTemperature

Identifiant matériel unique de la fonction au format `SERIAL.FUNCTIONID`.

dnp string **HardwareId**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par example `RELAYLO1-123456.relay1`).

temperature→IsOnline YTemperature

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

dnp 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.

temperature→LogFrequency YTemperature

Fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

dnp string **LogFrequency**

Modifiable. Modifie la fréquence d'enregistrement des mesures dans le datalogger. La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF". Attention il est inutile, voir contre productif, de régler la fréquence d'enregistrement à une valeur supérieure à la fréquence d'échantillonnage native du capteur: ces deux fréquences sont complètement indépendantes. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

temperature→LogicalName YTemperature

Nom logique de la fonction.

dnp string **LogicalName**

Modifiable. 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.

temperature→ReportFrequency YTemperature

Fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

dnp string **ReportFrequency**

Modifiable. Modifie la fréquence de notification périodique des valeurs mesurées. La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF". Attention il est inutile, voir contre productif, de régler la fréquence de notification périodique à une valeur supérieure à la fréquence d'échantillonnage native du capteur: ces deux fréquences sont complètement indépendantes. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

temperature→Resolution YTemperature

Résolution des valeurs mesurées.

dnp double **Resolution**

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Modifiable. Modifie la résolution des valeurs physique mesurées. La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

temperature→SensorType YTemperature

Type de capteur de température utilisé par le module **Modifiable**.

dnp int **SensorType**

Modifie le type de senseur utilisé par le module. Cette function sert à spécifier le type de thermocouple (K,E, etc..) raccordé au module. Cette fonction n'aura pas d'effet si le module utilise un capteur digital ou un thermistor. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

temperature→SerialNumber YTemperature

Numéro de série du module, préprogrammé en usine.

dnp string **SerialNumber**

temperature→SignalUnit YTemperature

Unité du signal électrique utilisé par le capteur.

dnp string **SignalUnit**

temperature→calibrateFromPoints() YTemperature

Enregistre des points de correction de mesure, typiquement pour compenser l'effet d'un boîtier sur les mesures rendues par le capteur.

```
js   function calibrateFromPoints( rawValues, refValues)
cpp  int calibrateFromPoints( vector<double> rawValues,
                              vector<double> refValues)
m    -(int) calibrateFromPoints : (NSMutableArray*) rawValues
                                : (NSMutableArray*) refValues
pas  LongInt calibrateFromPoints( rawValues: TDoubleArray,
                                  refValues: TDoubleArray): LongInt
vb   procedure calibrateFromPoints( ByVal rawValues As List(Of)
cs   int calibrateFromPoints( List<double> rawValues,
                              List<double> refValues)
java int calibrateFromPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)
uwp  async Task<int> calibrateFromPoints( List<double> rawValues,
                                          List<double> refValues)
py   calibrateFromPoints( rawValues, refValues)
php  function calibrateFromPoints( $rawValues, $refValues)
ts   async calibrateFromPoints( rawValues: number[], refValues: number[]): Promise<number>
es   async calibrateFromPoints( rawValues, refValues)
dnp  int calibrateFromPoints( double[] rawValues,
                              double[] refValues)
cp   int calibrateFromPoints( vector<double> rawValues,
                              vector<double> refValues)
cmd  YTemperature target calibrateFromPoints rawValues refValues
```

Il est possible d'enregistrer jusqu'à cinq points de correction. Les points de correction doivent être fournis en ordre croissant, et dans la plage valide du capteur. Le module effectue automatiquement une interpolation linéaire de l'erreur entre les points spécifiés. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Pour plus de plus amples possibilités d'appliquer une surcalibration aux capteurs, veuillez contacter support@yoctopuce.com.

Paramètres :

rawValues tableau de nombres flottants, correspondant aux valeurs brutes rendues par le capteur pour les points de correction.

refValues tableau de nombres flottants, correspondant aux valeurs corrigées désirées pour les points de correction.

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→clearCache() YTemperature

Invalide le cache.

js	function **clearCache**( )
cpp	void **clearCache**( )
m	-(void) **clearCache**
pas	**clearCache**( )
vb	procedure **clearCache**( )
cs	void **clearCache**( )
java	void **clearCache**( )
py	**clearCache**( )
php	function **clearCache**( )
ts	async **clearCache**( ): Promise<void>
es	async **clearCache**( )

Invalide le cache des valeurs courantes du capteur de température. Force le prochain appel à une méthode get_xxx() ou loadxxx() pour charger les les données depuis le module.

temperature→describe() YTemperature

Retourne un court texte décrivant de manière non-ambigüe l'instance du capteur de température au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

js	function **describe**( )
cpp	string **describe**( )
m	-(NSString*) **describe**
pas	string **describe**( ): string
vb	function **describe**( ) As String
cs	string **describe**( )
java	String **describe**( )
py	**describe**( )
php	function **describe**( )
ts	async **describe**( ): Promise<string>
es	async **describe**( )

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilsé lors du premier accès a la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou `"unresolved"` sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomeName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

Retourne :
une chaîne de caractères décrivant le capteur de température (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

temperature→get_advMode() YTemperature
temperature→advMode()

Retourne le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

```
js    function get_advMode( )
cpp   Y_ADVMODE_enum get_advMode( )
m     -(Y_ADVMODE_enum) advMode
pas   Integer get_advMode( ): Integer
vb    function get_advMode( ) As Integer
cs    int get_advMode( )
java  int get_advMode( )
uwp   async Task<int> get_advMode( )
py    get_advMode( )
php   function get_advMode( )
ts    async get_advMode( ): Promise<YSensor_AdvMode>
es    async get_advMode( )
dnp   int get_advMode( )
cp    int get_advMode( )
cmd   YTemperature target get_advMode
```

Retourne :

une valeur parmi `YTemperature.ADVMODE_IMMEDIATE`, `YTemperature.ADVMODE_PERIOD_AVG`, `YTemperature.ADVMODE_PERIOD_MIN` et `YTemperature.ADVMODE_PERIOD_MAX` représentant le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue)

En cas d'erreur, déclenche une exception ou retourne `YTemperature.ADVMODE_INVALID`.

temperature→get_advertisedValue() YTemperature
temperature→advertisedValue()

Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

js	function **get_advertisedValue**( )
cpp	string **get_advertisedValue**( )
m	-(NSString*) advertisedValue
pas	string **get_advertisedValue**( ): string
vb	function **get_advertisedValue**( ) As String
cs	string **get_advertisedValue**( )
java	String **get_advertisedValue**( )
uwp	async Task<string> **get_advertisedValue**( )
py	**get_advertisedValue**( )
php	function **get_advertisedValue**( )
ts	async **get_advertisedValue**( ): Promise<string>
es	async **get_advertisedValue**( )
dnp	string **get_advertisedValue**( )
cp	string **get_advertisedValue**( )
cmd	YTemperature **target get_advertisedValue**

Retourne :

une chaîne de caractères représentant la valeur courante du capteur de température (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `YTemperature.ADVERTISEDVALUE_INVALID`.

temperature→get_currentRawValue() YTemperature
temperature→currentRawValue()

Retourne la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés Celsius, sous forme de nombre à virgule.

js	function **get_currentRawValue**( )
cpp	double **get_currentRawValue**( )
m	-(double) currentRawValue
pas	double **get_currentRawValue**( ): double
vb	function **get_currentRawValue**( ) As Double
cs	double **get_currentRawValue**( )
java	double **get_currentRawValue**( )
uwp	async Task<double> **get_currentRawValue**( )
py	**get_currentRawValue**( )
php	function **get_currentRawValue**( )
ts	async **get_currentRawValue**( ): Promise<number>
es	async **get_currentRawValue**( )
dnp	double **get_currentRawValue**( )
cp	double **get_currentRawValue**( )
cmd	YTemperature **target get_currentRawValue**

Retourne :

une valeur numérique représentant la valeur brute retournée par le capteur (sans arrondi ni calibration), en degrés Celsius, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `YTemperature.CURRENTRAWVALUE_INVALID`.

temperature→get_currentValue() YTemperature
temperature→currentValue()

Retourne la valeur actuelle de la température, en degrés Celsius, sous forme de nombre à virgule.

```
js    function get_currentValue( )
cpp   double get_currentValue( )
m     -(double) currentValue
pas   double get_currentValue( ): double
vb    function get_currentValue( ) As Double
cs    double get_currentValue( )
java  double get_currentValue( )
uwp   async Task<double> get_currentValue( )
py    get_currentValue( )
php   function get_currentValue( )
ts    async get_currentValue( ): Promise<number>
es    async get_currentValue( )
dnp   double get_currentValue( )
cp    double get_currentValue( )
cmd   YTemperature target get_currentValue
```

Notez qu'un appel à get_currentValue() ne déclenche pas une mesure dans le module mais retourne simplement la valeur obtenue lors de la dernière mesure. En effet, en interne, chaque senseur Yoctopuce effectue des mesures en continu à une fréquence qui lui est propre.

Si vous rencontrez des problèmes de performances en utilisant la fonction get_currentValue() fréquemment, il vous faudra basculer sur un modèle de callbacks. Pour plus de détails jetez un coup d'oeil au chapitre "programmation avancée" du manuel de votre module.

Retourne :
une valeur numérique représentant la valeur actuelle de la température, en degrés Celsius, sous forme de nombre à virgule

En cas d'erreur, déclenche une exception ou retourne `YTemperature.CURRENTVALUE_INVALID`.

temperature→get_dataLogger() YTemperature
temperature→dataLogger()

Retourne l'objet `YDatalogger` du module qui héberge le senseur.

js	function **get_dataLogger**( )
cpp	YDataLogger* **get_dataLogger**( )
m	-(YDataLogger*) dataLogger
pas	TYDataLogger **get_dataLogger**( ): TYDataLogger
vb	function **get_dataLogger**( ) As YDataLogger
cs	YDataLogger **get_dataLogger**( )
java	YDataLogger **get_dataLogger**( )
uwp	async Task<YDataLogger> **get_dataLogger**( )
py	**get_dataLogger**( )
php	function **get_dataLogger**( )
ts	async **get_dataLogger**( ): Promise<YDataLogger | null>
es	async **get_dataLogger**( )
dnp	YDataLoggerProxy **get_dataLogger**( )
cp	YDataLoggerProxy* **get_dataLogger**( )

Cette méthode retourne un objet qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :
un objet `YDatalogger`, ou null en cas d'erreur.

temperature→get_errorMessage() YTemperature
temperature→errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

js	function **get_errorMessage**( )
cpp	string **get_errorMessage**( )
m	-(NSString*) errorMessage
pas	string **get_errorMessage**( ): string
vb	function **get_errorMessage**( ) As String
cs	string **get_errorMessage**( )
java	String **get_errorMessage**( )
py	**get_errorMessage**( )
php	function **get_errorMessage**( )
ts	**get_errorMessage**( ): string
es	**get_errorMessage**( )

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :
une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du capteur de température.

temperature→get_errorType() YTemperature
temperature→errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de température.

```
js    function get_errorType( )
cpp   YRETCODE get_errorType( )
m     -(YRETCODE) errorType
pas   YRETCODE get_errorType( ): YRETCODE
vb    function get_errorType( ) As YRETCODE
cs    YRETCODE get_errorType( )
java  int get_errorType( )
py    get_errorType( )
php   function get_errorType( )
ts    get_errorType( ): number
es    get_errorType( )
```

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :
un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du capteur de température.

temperature→get_friendlyName() YTemperature
temperature→friendlyName()

Retourne un identifiant global du capteur de température au format NOM_MODULE.NOM_FONCTION.

js	function **get_friendlyName**( )
cpp	string **get_friendlyName**( )
m	-(NSString*) friendlyName
cs	string **get_friendlyName**( )
java	String **get_friendlyName**( )
py	**get_friendlyName**( )
php	function **get_friendlyName**( )
ts	async **get_friendlyName**( ): Promise<string>
es	async **get_friendlyName**( )
dnp	string **get_friendlyName**( )
cp	string **get_friendlyName**( )

Le chaîne retournée utilise soit les noms logiques du module et du capteur de température si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel du capteur de température (par exemple: `MyCustomName.relay1`)

Retourne :
une chaîne de caractères identifiant le capteur de température en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `YTemperature.FRIENDLYNAME_INVALID`.

temperature→get_functionDescriptor() YTemperature
temperature→functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

```
js    function get_functionDescriptor( )
cpp   YFUN_DESCR get_functionDescriptor( )
m     -(YFUN_DESCR) functionDescriptor
pas   YFUN_DESCR get_functionDescriptor( ): YFUN_DESCR
vb    function get_functionDescriptor( ) As YFUN_DESCR
cs    YFUN_DESCR get_functionDescriptor( )
java  String get_functionDescriptor( )
py    get_functionDescriptor( )
php   function get_functionDescriptor( )
ts    async get_functionDescriptor( ): Promise<string>
es    async 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$CLASSNAME$.FUNCTIONDESCRIPTOR_INVALID

temperature→get_functionId() YTemperature
temperature→functionId()

Retourne l'identifiant matériel du capteur de température, sans référence au module.

js	function **get_functionId**( )
cpp	string **get_functionId**( )
m	-(NSString*) functionId
vb	function **get_functionId**( ) As String
cs	string **get_functionId**( )
java	String **get_functionId**( )
py	**get_functionId**( )
php	function **get_functionId**( )
ts	async **get_functionId**( ): Promise<string>
es	async **get_functionId**( )
dnp	string **get_functionId**( )
cp	string **get_functionId**( )

Par example `relay1`.

Retourne :
une chaîne de caractères identifiant le capteur de température (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `YTemperature.FUNCTIONID_INVALID`.

temperature→get_hardwareId() YTemperature
temperature→hardwareId()

Retourne l'identifiant matériel unique du capteur de température au format `SERIAL.FUNCTIONID`.

js	function **get_hardwareId**( )
cpp	string **get_hardwareId**( )
m	-(NSString*) hardwareId
vb	function **get_hardwareId**( ) As String
cs	string **get_hardwareId**( )
java	String **get_hardwareId**( )
py	**get_hardwareId**( )
php	function **get_hardwareId**( )
ts	async **get_hardwareId**( ): Promise<string>
es	async **get_hardwareId**( )
dnp	string **get_hardwareId**( )
cp	string **get_hardwareId**( )

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de température (par example `RELAYLO1-123456.relay1`).

Retourne :
une chaîne de caractères identifiant le capteur de température (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `YTemperature.HARDWAREID_INVALID`.

temperature→get_highestValue() YTemperature
temperature→highestValue()

Retourne la valeur maximale observée pour la température depuis le démarrage du module.

```
js    function get_highestValue( )
cpp   double get_highestValue( )
m     -(double) highestValue
pas   double get_highestValue( ): double
vb    function get_highestValue( ) As Double
cs    double get_highestValue( )
java  double get_highestValue( )
uwp   async Task<double> get_highestValue( )
py    get_highestValue( )
php   function get_highestValue( )
ts    async get_highestValue( ): Promise<number>
es    async get_highestValue( )
dnp   double get_highestValue( )
cp    double get_highestValue( )
cmd   YTemperature target get_highestValue
```

Peut être réinitialisé à une valeur arbitraire grâce à set_highestValue().

Retourne :
une valeur numérique représentant la valeur maximale observée pour la température depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne `YTemperature.HIGHESTVALUE_INVALID`.

temperature→get_logFrequency() YTemperature
temperature→logFrequency()

Retourne la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.

```
js    function get_logFrequency( )
cpp   string get_logFrequency( )
m     -(NSString*) logFrequency
pas   string get_logFrequency( ): string
vb    function get_logFrequency( ) As String
cs    string get_logFrequency( )
java  String get_logFrequency( )
uwp   async Task<string> get_logFrequency( )
py    get_logFrequency( )
php   function get_logFrequency( )
ts    async get_logFrequency( ): Promise<string>
es    async get_logFrequency( )
dnp   string get_logFrequency( )
cp    string get_logFrequency( )
cmd   YTemperature target get_logFrequency
```

Retourne :

une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données

En cas d'erreur, déclenche une exception ou retourne YTemperature.LOGFREQUENCY_INVALID.

temperature→get_logicalName() YTemperature
temperature→logicalName()

Retourne le nom logique du capteur de température.

```
js    function get_logicalName( )
cpp   string get_logicalName( )
m     -(NSString*) logicalName
pas   string get_logicalName( ): string
vb    function get_logicalName( ) As String
cs    string get_logicalName( )
java  String get_logicalName( )
uwp   async Task<string> get_logicalName( )
py    get_logicalName( )
php   function get_logicalName( )
ts    async get_logicalName( ): Promise<string>
es    async get_logicalName( )
dnp   string get_logicalName( )
cp    string get_logicalName( )
cmd   YTemperature target get_logicalName
```

Retourne :
une chaîne de caractères représentant le nom logique du capteur de température.

En cas d'erreur, déclenche une exception ou retourne `YTemperature.LOGICALNAME_INVALID`.

temperature→get_lowestValue() YTemperature
temperature→lowestValue()

Retourne la valeur minimale observée pour la température depuis le démarrage du module.

```
js    function get_lowestValue( )
cpp   double get_lowestValue( )
m     -(double) lowestValue
pas   double get_lowestValue( ): double
vb    function get_lowestValue( ) As Double
cs    double get_lowestValue( )
java  double get_lowestValue( )
uwp   async Task<double> get_lowestValue( )
py    get_lowestValue( )
php   function get_lowestValue( )
ts    async get_lowestValue( ): Promise<number>
es    async get_lowestValue( )
dnp   double get_lowestValue( )
cp    double get_lowestValue( )
cmd   YTemperature target get_lowestValue
```

Peut être réinitialisé à une valeur arbitraire grâce à set_lowestValue().

Retourne :
une valeur numérique représentant la valeur minimale observée pour la température depuis le démarrage du module

En cas d'erreur, déclenche une exception ou retourne YTemperature.LOWESTVALUE_INVALID.

temperature→get_module() YTemperature
temperature→module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

js	function **get_module**( )
cpp	YModule * **get_module**( )
m	-(YModule*) module
pas	TYModule **get_module**( ): TYModule
vb	function **get_module**( ) As YModule
cs	YModule **get_module**( )
java	YModule **get_module**( )
py	**get_module**( )
php	function **get_module**( )
ts	async **get_module**( ): Promise<YModule>
es	async **get_module**( )
dnp	YModuleProxy **get_module**( )
cp	YModuleProxy * **get_module**( )

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :
une instance de `YModule`

temperature→get_module_async() YTemperature
temperature→module_async()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js  function get_module_async( callback, context)
```

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_recordedData() YTemperature
temperature→recordedData()

Retourne un objet `YDataSet` représentant des mesures de ce capteur précédemment enregistrées à l'aide du DataLogger, pour l'intervalle de temps spécifié.

```
js    function get_recordedData( startTime, endTime)
cpp   YDataSet get_recordedData( double startTime, double endTime)
m     -(YDataSet*) recordedData : (double) startTime
                                : (double) endTime
pas   TYDataSet get_recordedData( startTime: double, endTime: double): TYDataSet
vb    function get_recordedData( ByVal startTime As Double,
                                 ByVal endTime As Double) As YDataSet
cs    YDataSet get_recordedData( double startTime, double endTime)
java  YDataSet get_recordedData( double startTime, double endTime)
uwp   async Task<YDataSet> get_recordedData( double startTime,
                                             double endTime)
py    get_recordedData( startTime, endTime)
php   function get_recordedData( $startTime, $endTime)
ts    async get_recordedData( startTime: number, endTime: number): Promise<YDataSet>
es    async get_recordedData( startTime, endTime)
dnp   YDataSetProxy get_recordedData( double startTime,
                                      double endTime)
cp    YDataSetProxy* get_recordedData( double startTime,
                                       double endTime)
cmd   YTemperature target get_recordedData startTime endTime
```

Veuillez vous référer à la documentation de la classe `YDataSet` pour plus plus d'informations sur la manière d'obtenir un aperçu des mesures pour la période, et comment charger progressivement une grande quantité de mesures depuis le dataLogger.

Cette méthode ne fonctionne que si le module utilise un firmware récent, car les objets `YDataSet` ne sont pas supportés par les firmwares antérieurs à la révision 13000.

Paramètres :

startTime le début de l'intervalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite sur le début des mesures.

endTime la find de l'intercalle de mesure désiré, c'est à dire en nombre de secondes depuis le 1er janvier 1970 UTC. La valeur 0 peut être utilisée pour ne poser aucune limite de fin.

Retourne :

une instance de `YDataSet`, dont les méthodes permettent de d'accéder aux données historiques souhaitées.

temperature→get_reportFrequency() YTemperature
temperature→reportFrequency()

Retourne la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.

```
js    function get_reportFrequency( )
cpp   string get_reportFrequency( )
m     -(NSString*) reportFrequency
pas   string get_reportFrequency( ): string
vb    function get_reportFrequency( ) As String
cs    string get_reportFrequency( )
java  String get_reportFrequency( )
uwp   async Task<string> get_reportFrequency( )
py    get_reportFrequency( )
php   function get_reportFrequency( )
ts    async get_reportFrequency( ): Promise<string>
es    async get_reportFrequency( )
dnp   string get_reportFrequency( )
cp    string get_reportFrequency( )
cmd   YTemperature target get_reportFrequency
```

Retourne :
une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction

En cas d'erreur, déclenche une exception ou retourne YTemperature.REPORTFREQUENCY_INVALID.

temperature→get_resolution() YTemperature
temperature→resolution()

Retourne la résolution des valeurs mesurées.

```
js    function get_resolution( )
cpp   double get_resolution( )
m     -(double) resolution
pas   double get_resolution( ): double
vb    function get_resolution( ) As Double
cs    double get_resolution( )
java  double get_resolution( )
uwp   async Task<double> get_resolution( )
py    get_resolution( )
php   function get_resolution( )
ts    async get_resolution( ): Promise<number>
es    async get_resolution( )
dnp   double get_resolution( )
cp    double get_resolution( )
cmd   YTemperature target get_resolution
```

La résolution correspond à la précision numérique de la représentation des mesures. Elle n'est pas forcément identique à la précision réelle du capteur. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne `YTemperature.RESOLUTION_INVALID`.

temperature→get_sensorState() YTemperature
temperature→sensorState()

Retourne le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment.

```
js    function get_sensorState( )
cpp   int get_sensorState( )
m     -(int) sensorState
pas   LongInt get_sensorState( ): LongInt
vb    function get_sensorState( ) As Integer
cs    int get_sensorState( )
java  int get_sensorState( )
uwp   async Task<int> get_sensorState( )
py    get_sensorState( )
php   function get_sensorState( )
ts    async get_sensorState( ): Promise<number>
es    async get_sensorState( )
dnp   int get_sensorState( )
cp    int get_sensorState( )
cmd   YTemperature target get_sensorState
```

Retourne :

un entier représentant le code d'état du capteur, qui vaut zéro lorsqu'une mesure actuelle est disponible, ou un code positif si le capteur n'est pas en mesure de fournir une valeur en ce moment

En cas d'erreur, déclenche une exception ou retourne YTemperature.SENSORSTATE_INVALID.

temperature→get_sensorType() YTemperature
temperature→sensorType()

Retourne le type de capteur de température utilisé par le module

```
js    function get_sensorType( )
cpp   Y_SENSORTYPE_enum get_sensorType( )
m     -(Y_SENSORTYPE_enum) sensorType
pas   Integer get_sensorType( ): Integer
vb    function get_sensorType( ) As Integer
cs    int get_sensorType( )
java  int get_sensorType( )
uwp   async Task<int> get_sensorType( )
py    get_sensorType( )
php   function get_sensorType( )
ts    async get_sensorType( ): Promise<YTemperature_SensorType>
es    async get_sensorType( )
dnp   int get_sensorType( )
cp    int get_sensorType( )
cmd   YTemperature target get_sensorType
```

Retourne :

une valeur parmi YTemperature.SENSORTYPE_DIGITAL, YTemperature.SENSORTYPE_TYPE_K, YTemperature.SENSORTYPE_TYPE_E, YTemperature.SENSORTYPE_TYPE_J, YTemperature.SENSORTYPE_TYPE_N, YTemperature.SENSORTYPE_TYPE_R, YTemperature.SENSORTYPE_TYPE_S, YTemperature.SENSORTYPE_TYPE_T, YTemperature.SENSORTYPE_PT100_4WIRES, YTemperature.SENSORTYPE_PT100_3WIRES, YTemperature.SENSORTYPE_PT100_2WIRES, YTemperature.SENSORTYPE_RES_OHM, YTemperature.SENSORTYPE_RES_NTC, YTemperature.SENSORTYPE_RES_LINEAR, YTemperature.SENSORTYPE_RES_INTERNAL, YTemperature.SENSORTYPE_IR, YTemperature.SENSORTYPE_RES_PT1000 et YTemperature.SENSORTYPE_CHANNEL_OFF représentant le type de capteur de température utilisé par le module

En cas d'erreur, déclenche une exception ou retourne YTemperature.SENSORTYPE_INVALID.

temperature→get_serialNumber() YTemperature
temperature→serialNumber()

Retourne le numéro de série du module, préprogrammé en usine.

```
js    function get_serialNumber( )
cpp   string get_serialNumber( )
m     -(NSString*) serialNumber
pas   string get_serialNumber( ): string
vb    function get_serialNumber( ) As String
cs    string get_serialNumber( )
java  String get_serialNumber( )
uwp   async Task<string> get_serialNumber( )
py    get_serialNumber( )
php   function get_serialNumber( )
ts    async get_serialNumber( ): Promise<string>
es    async get_serialNumber( )
dnp   string get_serialNumber( )
cp    string get_serialNumber( )
cmd   YTemperature target get_serialNumber
```

Retourne :
: une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine.

En cas d'erreur, déclenche une exception ou retourne YFunction.SERIALNUMBER_INVALID.

temperature→get_signalUnit() YTemperature
temperature→signalUnit()

Retourne l'unité du signal électrique utilisé par le capteur.

```
js    function get_signalUnit( )
cpp   string get_signalUnit( )
m     -(NSString*) signalUnit
pas   string get_signalUnit( ): string
vb    function get_signalUnit( ) As String
cs    string get_signalUnit( )
java  String get_signalUnit( )
uwp   async Task<string> get_signalUnit( )
py    get_signalUnit( )
php   function get_signalUnit( )
ts    async get_signalUnit( ): Promise<string>
es    async get_signalUnit( )
dnp   string get_signalUnit( )
cp    string get_signalUnit( )
cmd   YTemperature target get_signalUnit
```

Retourne :
une chaîne de caractères représentant l'unité du signal électrique utilisé par le capteur

En cas d'erreur, déclenche une exception ou retourne `YTemperature.SIGNALUNIT_INVALID`.

temperature→get_signalValue() YTemperature
temperature→signalValue()

Retourne la valeur actuelle du signal électrique mesuré par le capteur.

```
js    function get_signalValue( )
cpp   double get_signalValue( )
m     -(double) signalValue
pas   double get_signalValue( ): double
vb    function get_signalValue( ) As Double
cs    double get_signalValue( )
java  double get_signalValue( )
uwp   async Task<double> get_signalValue( )
py    get_signalValue( )
php   function get_signalValue( )
ts    async get_signalValue( ): Promise<number>
es    async get_signalValue( )
dnp   double get_signalValue( )
cp    double get_signalValue( )
cmd   YTemperature target get_signalValue
```

Retourne :
une valeur numérique représentant la valeur actuelle du signal électrique mesuré par le capteur

En cas d'erreur, déclenche une exception ou retourne `YTemperature.SIGNALVALUE_INVALID`.

temperature→get_unit() YTemperature
temperature→unit()

Retourne l'unité dans laquelle la température est exprimée.

```
js    function get_unit( )
cpp   string get_unit( )
m     -(NSString*) unit
pas   string get_unit( ): string
vb    function get_unit( ) As String
cs    string get_unit( )
java  String get_unit( )
uwp   async Task<string> get_unit( )
py    get_unit( )
php   function get_unit( )
ts    async get_unit( ): Promise<string>
es    async get_unit( )
dnp   string get_unit( )
cp    string get_unit( )
cmd   YTemperature target get_unit
```

Retourne :
une chaîne de caractères représentant l'unité dans laquelle la température est exprimée

En cas d'erreur, déclenche une exception ou retourne `YTemperature.UNIT_INVALID`.

temperature→get_userData() YTemperature
temperature→userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

js	function **get_userData**( )
cpp	void * **get_userData**( )
m	-(id) userData
pas	Tobject **get_userData**( ): Tobject
vb	function **get_userData**( ) As Object
cs	object **get_userData**( )
java	Object **get_userData**( )
py	**get_userData**( )
php	function **get_userData**( )
ts	async **get_userData**( ): Promise<object|null>
es	async **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() YTemperature

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

js	function **isOnline**( )
cpp	bool **isOnline**( )
m	-(BOOL) **isOnline**
pas	boolean **isOnline**( ): boolean
vb	function **isOnline**( ) As Boolean
cs	bool **isOnline**( )
java	boolean **isOnline**( )
py	**isOnline**( )
php	function **isOnline**( )
ts	async **isOnline**( ): Promise<boolean>
es	async **isOnline**( )
dnp	bool **isOnline**( )
cp	bool **isOnline**( )

Si les valeurs des attributs en cache du capteur de température sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si le capteur de température est joignable, `false` sinon

temperature→isOnline_async() YTemperature

Vérifie si le module hébergeant le capteur de température est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du capteur de température sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

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 machine virtuelle Javascript avec une attente active.

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→isReadOnly() YTemperature

Test si la fonction est en lecture seule.

cpp	bool **isReadOnly**( )
m	-(bool) **isReadOnly**
pas	boolean **isReadOnly**( ): boolean
vb	function **isReadOnly**( ) As Boolean
cs	bool **isReadOnly**( )
java	boolean **isReadOnly**( )
uwp	async Task<bool> **isReadOnly**( )
py	**isReadOnly**( )
php	function **isReadOnly**( )
ts	async **isReadOnly**( ): Promise<boolean>
es	async **isReadOnly**( )
dnp	bool **isReadOnly**( )
cp	bool **isReadOnly**( )
cmd	YTemperature **target isReadOnly**

Retourne vrais si la fonction est protégé en ecriture ou que la fontion n'est pas disponible.

Retourne :
`true` si la fonction est protégé en ecriture ou que la fontion n'est pas disponible

temperature→isSensorReady() YTemperature

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

```
cmd YTemperature target isSensorReady
```

Retourne faux si le module n'est pas joignable, ou que le capteur n'a pas de mesure actuelle à communiquer. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si le capteur dispose d'une mesure actuelle, `false` sinon

temperature→load() YTemperature

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

js	function **load**(**msValidity**)
cpp	YRETCODE **load**(int **msValidity**)
m	-(YRETCODE) **load** : (u64) **msValidity**
pas	YRETCODE **load**(**msValidity**: u64): YRETCODE
vb	function **load**(ByVal **msValidity** As Long) As YRETCODE
cs	YRETCODE **load**(ulong **msValidity**)
java	int **load**(long **msValidity**)
py	**load**(**msValidity**)
php	function **load**(**$msValidity**)
ts	async **load**(**msValidity**: number): Promise<number>
es	async **load**(**msValidity**)

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→loadAttribute() YTemperature

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

```
js    function loadAttribute( attrName)
cpp   string loadAttribute( string attrName)
m     -(NSString*) loadAttribute : (NSString*) attrName
pas   string loadAttribute( attrName: string): string
vb    function loadAttribute( ByVal attrName As String) As String
cs    string loadAttribute( string attrName)
java  String loadAttribute( String attrName)
uwp   async Task<string> loadAttribute( string attrName)
py    loadAttribute( attrName)
php   function loadAttribute( $attrName)
ts    async loadAttribute( attrName: string): Promise<string>
es    async loadAttribute( attrName)
dnp   string loadAttribute( string attrName)
cp    string loadAttribute( string attrName)
```

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

temperature→loadCalibrationPoints() YTemperature

Récupère les points de correction de mesure précédemment enregistrés à l'aide de la méthode `calibrateFromPoints`.

```
js    function loadCalibrationPoints( rawValues, refValues)
cpp   int loadCalibrationPoints( vector<double> rawValues,
                                 vector<double> refValues)
m     -(int) loadCalibrationPoints : (NSMutableArray*) rawValues
                                   : (NSMutableArray*) refValues
pas   LongInt loadCalibrationPoints( var rawValues: TDoubleArray,
                                     var refValues: TDoubleArray): LongInt
vb    procedure loadCalibrationPoints( ByVal rawValues As List(Of)
cs    int loadCalibrationPoints( List<double> rawValues,
                                 List<double> refValues)
java  int loadCalibrationPoints( ArrayList<Double> rawValues,
                                 ArrayList<Double> refValues)
uwp   async Task<int> loadCalibrationPoints( List<double> rawValues,
                                             List<double> refValues)
py    loadCalibrationPoints( rawValues, refValues)
php   function loadCalibrationPoints( &$rawValues, &$refValues)
ts    async loadCalibrationPoints( rawValues: number[], refValues: number[]): Promise<number>
es    async loadCalibrationPoints( rawValues, refValues)
cmd   YTemperature target loadCalibrationPoints rawValues refValues
```

Paramètres :

rawValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs brutes des points de correction.

refValues tableau de nombres flottants, qui sera rempli par la fonction avec les valeurs désirées des points de correction.

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→loadThermistorResponseTable() YTemperature

Récupère la table de réponse d'un thermistor précédemment enregistrée à l'aide de la fonction `set_thermistorResponseTable`.

```
js    function loadThermistorResponseTable( tempValues, resValues)
cpp   int loadThermistorResponseTable( vector<double> tempValues,
                                       vector<double> resValues)
m     -(int) loadThermistorResponseTable : (NSMutableArray*) tempValues
                                         : (NSMutableArray*) resValues
pas   LongInt loadThermistorResponseTable( var tempValues: TDoubleArray,
                                           var resValues: TDoubleArray): LongInt
vb    procedure loadThermistorResponseTable( ByVal tempValues As List(Of)
cs    int loadThermistorResponseTable( List<double> tempValues,
                                       List<double> resValues)
java  int loadThermistorResponseTable( ArrayList<Double> tempValues,
                                       ArrayList<Double> resValues)
uwp   async Task<int> loadThermistorResponseTable( List<double> tempValues,
                                                   List<double> resValues)
py    loadThermistorResponseTable( tempValues, resValues)
php   function loadThermistorResponseTable( &$tempValues, &$resValues)
ts    async loadThermistorResponseTable( tempValues: number[], resValues: number[]):
                                         Promise<number>
es    async loadThermistorResponseTable( tempValues, resValues)
cmd   YTemperature target loadThermistorResponseTable tempValues resValues
```

Cette fonction ne peut être utilisée qu'avec les capteurs de température basés sur un thermistor.

Paramètres :

tempValues tableau de nombres flottants, qui sera rempli par la fonction avec les différentes températures (en degrés Celsius) pour lesquelles la résistance du thermistor est spécifiée.

resValues tableau de nombres flottants, qui sera rempli par la fonction avec les résistances (en Ohms) pour chacun des points de température, index par index.

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→load_async() YTemperature

Met en cache les valeurs courantes du capteur de température, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
```

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 machine virtuelle Javascript avec une attente active.

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→muteValueCallbacks() YTemperature

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

js	function **muteValueCallbacks**( )
cpp	int **muteValueCallbacks**( )
m	-(int) **muteValueCallbacks**
pas	LongInt **muteValueCallbacks**( ): LongInt
vb	function **muteValueCallbacks**( ) As Integer
cs	int **muteValueCallbacks**( )
java	int **muteValueCallbacks**( )
uwp	async Task<int> **muteValueCallbacks**( )
py	**muteValueCallbacks**( )
php	function **muteValueCallbacks**( )
ts	async **muteValueCallbacks**( ): Promise<number>
es	async **muteValueCallbacks**( )
dnp	int **muteValueCallbacks**( )
cp	int **muteValueCallbacks**( )
cmd	YTemperature **target muteValueCallbacks**

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclanchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→nextTemperature() YTemperature

Continue l'énumération des capteurs de température commencée à l'aide de `yFirstTemperature()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les capteurs de température sont retournés.

js	function **nextTemperature**( )
cpp	YTemperature * **nextTemperature**( )
m	-(nullable YTemperature*) **nextTemperature**
pas	TYTemperature **nextTemperature**( ): TYTemperature
vb	function **nextTemperature**( ) As YTemperature
cs	YTemperature **nextTemperature**( )
java	YTemperature **nextTemperature**( )
uwp	YTemperature **nextTemperature**( )
py	**nextTemperature**( )
php	function **nextTemperature**( )
ts	**nextTemperature**( ): YTemperature | null
es	**nextTemperature**( )

Si vous souhaitez retrouver un capteur de température spécifique, utilisez `Temperature.findTemperature()` avec un hardwareID ou un nom logique.

Retourne :

un pointeur sur un objet `YTemperature` accessible en ligne, ou `null` lorsque l'énumération est terminée.

temperature→registerTimedReportCallback() YTemperature

Enregistre la fonction de callback qui est appelée à chaque notification périodique.

js	function **registerTimedReportCallback**(**callback**)
cpp	int **registerTimedReportCallback**(YTemperatureTimedReportCallback **callback**)
m	-(int) **registerTimedReportCallback** : (YTemperatureTimedReportCallback _Nullable) **callback**
pas	LongInt **registerTimedReportCallback**(**callback**: TYTemperatureTimedReportCallback): LongInt
vb	function **registerTimedReportCallback**(ByVal **callback** As YTemperatureTimedReportCallback) As Integer
cs	int **registerTimedReportCallback**(TimedReportCallback **callback**)
java	int **registerTimedReportCallback**(TimedReportCallback **callback**)
uwp	async Task<int> **registerTimedReportCallback**(TimedReportCallback **callback**)
py	**registerTimedReportCallback**(**callback**)
php	function **registerTimedReportCallback**(**$callback**)
ts	async **registerTimedReportCallback**(**callback**: YTemperatureTimedReportCallback | null): Promise<number>
es	async **registerTimedReportCallback**(**callback**)

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callbacks peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callbacks ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et un objet `YMeasure` décrivant la nouvelle valeur publiée.

temperature→registerValueCallback() YTemperature

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

```
js    function registerValueCallback( callback)
cpp   int registerValueCallback( YTemperatureValueCallback callback)
m     -(int) registerValueCallback : (YTemperatureValueCallback _Nullable) callback
pas   LongInt registerValueCallback( callback: TYTemperatureValueCallback): LongInt
vb    function registerValueCallback( ByVal callback As YTemperatureValueCallback) As Integer
cs    int registerValueCallback( ValueCallback callback)
java  int registerValueCallback( UpdateCallback callback)
uwp   async Task<int> registerValueCallback( ValueCallback callback)
py    registerValueCallback( callback)
php   function registerValueCallback( $callback)
ts    async registerValueCallback( callback: YTemperatureValueCallback | null): Promise<number>
es    async registerValueCallback( callback)
```

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

temperature→set_advMode() YTemperature
temperature→setAdvMode()

Modifie le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

```
js    function set_advMode( newval)
cpp   int set_advMode( Y_ADVMODE_enum newval)
m     -(int) setAdvMode : (Y_ADVMODE_enum) newval
pas   integer set_advMode( newval: Integer): integer
vb    function set_advMode( ByVal newval As Integer) As Integer
cs    int set_advMode( int newval)
java  int set_advMode( int newval)
uwp   async Task<int> set_advMode( int newval)
py    set_advMode( newval)
php   function set_advMode( $newval)
ts    async set_advMode( newval: YSensor_AdvMode): Promise<number>
es    async set_advMode( newval)
dnp   int set_advMode( int newval)
cp    int set_advMode( int newval)
cmd   YTemperature target set_advMode newval
```

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur parmi `YTemperature.ADVMODE_IMMEDIATE`, `YTemperature.ADVMODE_PERIOD_AVG`, `YTemperature.ADVMODE_PERIOD_MIN` et `YTemperature.ADVMODE_PERIOD_MAX` représentant le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue)

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_highestValue() YTemperature
temperature→setHighestValue()

Modifie la mémoire de valeur maximale observée.

```
js    function set_highestValue( newval)
cpp   int set_highestValue( double newval)
m     -(int) setHighestValue : (double) newval
pas   integer set_highestValue( newval: double): integer
vb    function set_highestValue( ByVal newval As Double) As Integer
cs    int set_highestValue( double newval)
java  int set_highestValue( double newval)
uwp   async Task<int> set_highestValue( double newval)
py    set_highestValue( newval)
php   function set_highestValue( $newval)
ts    async set_highestValue( newval: number): Promise<number>
es    async set_highestValue( newval)
dnp   int set_highestValue( double newval)
cp    int set_highestValue( double newval)
cmd   YTemperature target set_highestValue newval
```

Utile pour réinitialiser la valeur renvoyée par get_highestValue().

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_logFrequency() YTemperature
temperature→setLogFrequency()

Modifie la fréquence d'enregistrement des mesures dans le datalogger.

```
js    function set_logFrequency( newval)
cpp   int set_logFrequency( string newval)
m     -(int) setLogFrequency : (NSString*) newval
pas   integer set_logFrequency( newval: string): integer
vb    function set_logFrequency( ByVal newval As String) As Integer
cs    int set_logFrequency( string newval)
java  int set_logFrequency( String newval)
uwp   async Task<int> set_logFrequency( string newval)
py    set_logFrequency( newval)
php   function set_logFrequency( $newval)
ts    async set_logFrequency( newval: string): Promise<number>
es    async set_logFrequency( newval)
dnp   int set_logFrequency( string newval)
cp    int set_logFrequency( string newval)
cmd   YTemperature target set_logFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver l'enregistrement des mesures de cette fonction, utilisez la valeur "OFF". Attention il est inutile, voir contre productif, de régler la fréquence d'enregistrement à une valeur supérieure à la fréquence d'échantillonnage native du capteur: ces deux fréquences sont complètement indépendantes. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant la fréquence d'enregistrement des mesures dans le datalogger

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_logicalName() YTemperature
temperature→setLogicalName()

Modifie le nom logique du capteur de température.

js	function **set_logicalName**(**newval**)
cpp	int **set_logicalName**(string **newval**)
m	-(int) setLogicalName : (NSString*) **newval**
pas	integer **set_logicalName**(**newval**: string): integer
vb	function **set_logicalName**(ByVal **newval** As String) As Integer
cs	int **set_logicalName**(string **newval**)
java	int **set_logicalName**(String **newval**)
uwp	async Task<int> **set_logicalName**(string **newval**)
py	**set_logicalName**(**newval**)
php	function **set_logicalName**(**$newval**)
ts	async **set_logicalName**(**newval**: string): Promise<number>
es	async **set_logicalName**(**newval**)
dnp	int **set_logicalName**(string **newval**)
cp	int **set_logicalName**(string **newval**)
cmd	YTemperature **target set_logicalName newval**

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de température.

Retourne :

`YAPI.SUCCESS` si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_lowestValue() YTemperature
temperature→setLowestValue()

Modifie la mémoire de valeur minimale observée.

```
js    function set_lowestValue( newval)
cpp   int set_lowestValue( double newval)
m     -(int) setLowestValue : (double) newval
pas   integer set_lowestValue( newval: double): integer
vb    function set_lowestValue( ByVal newval As Double) As Integer
cs    int set_lowestValue( double newval)
java  int set_lowestValue( double newval)
uwp   async Task<int> set_lowestValue( double newval)
py    set_lowestValue( newval)
php   function set_lowestValue( $newval)
ts    async set_lowestValue( newval: number): Promise<number>
es    async set_lowestValue( newval)
dnp   int set_lowestValue( double newval)
cp    int set_lowestValue( double newval)
cmd   YTemperature target set_lowestValue newval
```

Utile pour réinitialiser la valeur renvoyée par get_lowestValue().

Paramètres :

newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_ntcParameters() YTemperature
temperature→setNtcParameters()

Configure les paramètres d'un thermistor NTC pour calculer correctement la température sur la base de la résistance mesurée.

js	function **set_ntcParameters**(**res25**, **beta**)
cpp	int **set_ntcParameters**(double **res25**, double **beta**)
m	-(int) setNtcParameters : (double) **res25** : (double) **beta**
pas	LongInt **set_ntcParameters**(**res25**: double, **beta**: double): LongInt
vb	function **set_ntcParameters**(ByVal **res25** As Double, ByVal **beta** As Double) As Integer
cs	int **set_ntcParameters**(double **res25**, double **beta**)
java	int **set_ntcParameters**(double **res25**, double **beta**)
uwp	async Task<int> **set_ntcParameters**(double **res25**, double **beta**)
py	**set_ntcParameters**(**res25**, **beta**)
php	function **set_ntcParameters**(**$res25**, **$beta**)
ts	async **set_ntcParameters**(**res25**: number, **beta**: number): Promise<number>
es	async **set_ntcParameters**(**res25**, **beta**)
dnp	int **set_ntcParameters**(double **res25**, double **beta**)
cp	int **set_ntcParameters**(double **res25**, double **beta**)
cmd	YTemperature **target set_ntcParameters res25 beta**

Pour plus de précision, vous pouvez saisir une table complète à l'aide de la fonction set_thermistorResponseTable. Cette fonction ne peut être utilisée qu'avec les capteurs de température basés sur un thermistor.

Paramètres :

res25 résistance à 25 degrés Celsius

beta coefficient Beta

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_reportFrequency() YTemperature
temperature→setReportFrequency()

Modifie la fréquence de notification périodique des valeurs mesurées.

```
js    function set_reportFrequency( newval)
cpp   int set_reportFrequency( string newval)
m     -(int) setReportFrequency : (NSString*) newval
pas   integer set_reportFrequency( newval: string): integer
vb    function set_reportFrequency( ByVal newval As String) As Integer
cs    int set_reportFrequency( string newval)
java  int set_reportFrequency( String newval)
uwp   async Task<int> set_reportFrequency( string newval)
py    set_reportFrequency( newval)
php   function set_reportFrequency( $newval)
ts    async set_reportFrequency( newval: string): Promise<number>
es    async set_reportFrequency( newval)
dnp   int set_reportFrequency( string newval)
cp    int set_reportFrequency( string newval)
cmd   YTemperature target set_reportFrequency newval
```

La fréquence peut être spécifiée en mesures par secondes, en mesures par minutes (par exemple "15/m") ou en mesures par heure (par exemple "4/h"). Pour désactiver les notifications périodiques pour cette fonction, utilisez la valeur "OFF". Attention il est inutile, voir contre productif, de régler la fréquence de notification périodique à une valeur supérieure à la fréquence d'échantillonnage native du capteur: ces deux fréquences sont complètement indépendantes. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant la fréquence de notification périodique des valeurs mesurées

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_resolution() YTemperature
temperature→setResolution()

Modifie la résolution des valeurs physique mesurées.

```
js    function set_resolution( newval)
cpp   int set_resolution( double newval)
m     -(int) setResolution : (double) newval
pas   integer set_resolution( newval: double): integer
vb    function set_resolution( ByVal newval As Double) As Integer
cs    int set_resolution( double newval)
java  int set_resolution( double newval)
uwp   async Task<int> set_resolution( double newval)
py    set_resolution( newval)
php   function set_resolution( $newval)
ts    async set_resolution( newval: number): Promise<number>
es    async set_resolution( newval)
dnp   int set_resolution( double newval)
cp    int set_resolution( double newval)
cmd   YTemperature target set_resolution newval
```

La résolution correspond à la précision de l'affichage des mesures. Elle ne change pas la précision de la mesure elle-même. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_sensorType() YTemperature
temperature→setSensorType()

Modifie le type de senseur utilisé par le module.

```
js    function set_sensorType( newval)
cpp   int set_sensorType( Y_SENSORTYPE_enum newval)
m     -(int) setSensorType : (Y_SENSORTYPE_enum) newval
pas   integer set_sensorType( newval: Integer): integer
vb    function set_sensorType( ByVal newval As Integer) As Integer
cs    int set_sensorType( int newval)
java  int set_sensorType( int newval)
uwp   async Task<int> set_sensorType( int newval)
py    set_sensorType( newval)
php   function set_sensorType( $newval)
ts    async set_sensorType( newval: YTemperature_SensorType): Promise<number>
es    async set_sensorType( newval)
dnp   int set_sensorType( int newval)
cp    int set_sensorType( int newval)
cmd   YTemperature target set_sensorType newval
```

Cette function sert à spécifier le type de thermocouple (K,E, etc..) raccordé au module. Cette fonction n'aura pas d'effet si le module utilise un capteur digital ou un thermistor. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval `une valeur parmi YTemperature.SENSORTYPE_DIGITAL`, `YTemperature.SENSORTYPE_TYPE_K`, `YTemperature.SENSORTYPE_TYPE_E`, `YTemperature.SENSORTYPE_TYPE_J`, `YTemperature.SENSORTYPE_TYPE_N`, `YTemperature.SENSORTYPE_TYPE_R`, `YTemperature.SENSORTYPE_TYPE_S`, `YTemperature.SENSORTYPE_TYPE_T`, `YTemperature.SENSORTYPE_PT100_4WIRES`, `YTemperature.SENSORTYPE_PT100_3WIRES`, `YTemperature.SENSORTYPE_PT100_2WIRES`, `YTemperature.SENSORTYPE_RES_OHM`, `YTemperature.SENSORTYPE_RES_NTC`, `YTemperature.SENSORTYPE_RES_LINEAR`, `YTemperature.SENSORTYPE_RES_INTERNAL`, `YTemperature.SENSORTYPE_IR`, `YTemperature.SENSORTYPE_RES_PT1000` et `YTemperature.SENSORTYPE_CHANNEL_OFF` représentant le type de senseur 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.

temperature→set_thermistorResponseTable() YTemperature
temperature→setThermistorResponseTable()

Enregistre la table de réponse d'un thermistor, afin de pouvoir interpoler la température sur la base de la résistance mesurée.

```
js    function set_thermistorResponseTable( tempValues, resValues)
cpp   int set_thermistorResponseTable( vector<double> tempValues,
                                        vector<double> resValues)
m     -(int) setThermistorResponseTable : (NSMutableArray*) tempValues
                                        : (NSMutableArray*) resValues
pas   LongInt set_thermistorResponseTable( tempValues: TDoubleArray,
                                           resValues: TDoubleArray): LongInt
vb    procedure set_thermistorResponseTable( ByVal tempValues As List(Of)
cs    int set_thermistorResponseTable( List<double> tempValues,
                                       List<double> resValues)
java  int set_thermistorResponseTable( ArrayList<Double> tempValues,
                                       ArrayList<Double> resValues)
uwp   async Task<int> set_thermistorResponseTable( List<double> tempValues,
                                                   List<double> resValues)
py    set_thermistorResponseTable( tempValues, resValues)
php   function set_thermistorResponseTable( $tempValues, $resValues)
ts    async set_thermistorResponseTable( tempValues: number[], resValues: number[]):
                                         Promise<number>
es    async set_thermistorResponseTable( tempValues, resValues)
dnp   int set_thermistorResponseTable( double[] tempValues,
                                       double[] resValues)
cp    int set_thermistorResponseTable( vector<double> tempValues,
                                       vector<double> resValues)
cmd   YTemperature target set_thermistorResponseTable tempValues resValues
```

Cette fonction ne peut être utilisée qu'avec les capteurs de température basés sur un thermistor.

Paramètres :

tempValues tableau de nombres flottants, correspondant aux différentes températures (en degrés Celsius) pour lesquelles la résistance du thermistor est spécifiée.

resValues tableau de nombres flottants, correspondant aux résistances (en Ohms) pour chacun des points de température, index par index.

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_unit() YTemperature
temperature→setUnit()

Modifie l'unité dans laquelle la température mesurée est exprimée.

```
js    function set_unit( newval)
cpp   int set_unit( string newval)
m     -(int) setUnit : (NSString*) newval
pas   integer set_unit( newval: string): integer
vb    function set_unit( ByVal newval As String) As Integer
cs    int set_unit( string newval)
java  int set_unit( String newval)
uwp   async Task<int> set_unit( string newval)
py    set_unit( newval)
php   function set_unit( $newval)
ts    async set_unit( newval: string): Promise<number>
es    async set_unit( newval)
dnp   int set_unit( string newval)
cp    int set_unit( string newval)
cmd   YTemperature target set_unit newval
```

Cette unité est une chaîne de charactère. Si cette chaîne de caractère se termine par un F les valeurs mesurée seront rendues en degrés Fahrenheit, si elle se termine par un K, les valeurs de température seront rendues en degrés Kelvin. Si elle se termine par un C, les valeurs de température seront rendues en degrés Celsius. Si elle ne setermine n'importe quel autre caractère, le changement est ignoré. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé. Attention: si une calibration spécifique est définie pour la fonction temperature, un changement d'unité a toutes les chances de la fausser.

Paramètres :

newval une chaîne de caractères représentant l'unité dans laquelle la température mesurée est exprimée

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→set_userData() YTemperature
temperature→setUserData()

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

```
js    function set_userData( data)
cpp   void set_userData( void * data)
m     -(void) setUserData : (id) data
pas   set_userData( data: Tobject)
vb    procedure set_userData( ByVal data As Object)
cs    void set_userData( object data)
java  void set_userData( Object data)
py    set_userData( data)
php   function set_userData( $data)
ts    async set_userData( data: object|null): Promise<void>
es    async set_userData( data)
```

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

temperature→startDataLogger() YTemperature

Démarre l'enregistreur de données du module.

js	function **startDataLogger**( )
cpp	int **startDataLogger**( )
m	-(int) **startDataLogger**
pas	LongInt **startDataLogger**( ): LongInt
vb	function **startDataLogger**( ) As Integer
cs	int **startDataLogger**( )
java	int **startDataLogger**( )
uwp	async Task<int> **startDataLogger**( )
py	**startDataLogger**( )
php	function **startDataLogger**( )
ts	async **startDataLogger**( ): Promise<number>
es	async **startDataLogger**( )
dnp	int **startDataLogger**( )
cp	int **startDataLogger**( )
cmd	YTemperature **target startDataLogger**

Attention, l'enregistreur ne sauvera les mesures de ce capteur que si la fréquence d'enregistrement (logFrequency) n'est pas sur "OFF".

Retourne :
`YAPI.SUCCESS` si l'opération se déroule sans erreur.

temperature→stopDataLogger() YTemperature

Arrête l'enregistreur de données du module.

js	function **stopDataLogger**( )
cpp	int **stopDataLogger**( )
m	-(int) **stopDataLogger**
pas	LongInt **stopDataLogger**( ): LongInt
vb	function **stopDataLogger**( ) As Integer
cs	int **stopDataLogger**( )
java	int **stopDataLogger**( )
uwp	async Task<int> **stopDataLogger**( )
py	**stopDataLogger**( )
php	function **stopDataLogger**( )
ts	async **stopDataLogger**( ): Promise<number>
es	async **stopDataLogger**( )
dnp	int **stopDataLogger**( )
cp	int **stopDataLogger**( )
cmd	YTemperature **target stopDataLogger**

Retourne :
`YAPI.SUCCESS` si l'opération se déroule sans erreur.

temperature→unmuteValueCallbacks() YTemperature

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

js	function **unmuteValueCallbacks**( )
cpp	int **unmuteValueCallbacks**( )
m	-(int) **unmuteValueCallbacks**
pas	LongInt **unmuteValueCallbacks**( ): LongInt
vb	function **unmuteValueCallbacks**( ) As Integer
cs	int **unmuteValueCallbacks**( )
java	int **unmuteValueCallbacks**( )
uwp	async Task<int> **unmuteValueCallbacks**( )
py	**unmuteValueCallbacks**( )
php	function **unmuteValueCallbacks**( )
ts	async **unmuteValueCallbacks**( ): Promise<number>
es	async **unmuteValueCallbacks**( )
dnp	int **unmuteValueCallbacks**( )
cp	int **unmuteValueCallbacks**( )
cmd	YTemperature **target unmuteValueCallbacks**

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :
`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

temperature→wait_async() YTemperature

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

```
js  function wait_async( callback, context)
ts  wait_async( callback: Function, context: object)
es  wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout.

25. Problèmes courants

25.1. Par où commencer ?

Si c'est la première fois que vous utilisez un module Yoctopuce et ne savez pas trop par où commencer, allez donc jeter un coup d'œil sur le blog de Yoctopuce. Il y a une section dédiée aux débutants [1].

25.2. Linux et USB

Pour fonctionner correctement sous Linux la librairie a besoin d'avoir accès en écriture à tous les périphériques USB Yoctopuce. Or, par défaut, sous Linux les droits d'accès des utilisateurs non-root à USB sont limités à la lecture. Afin d'éviter de devoir lancer les exécutables en tant que root, il faut créer une nouvelle règle *udev* pour autoriser un ou plusieurs utilisateurs à accéder en écriture aux périphériques Yoctopuce.

Pour ajouter une règle *udev* à votre installation, il faut ajouter un fichier avec un nom au format "`##-nomArbitraire.rules`" dans le répertoire "`/etc/udev/rules.d`". Lors du démarrage du système, *udev* va lire tous les fichiers avec l'extension "`.rules`" de ce répertoire en respectant l'ordre alphabétique (par exemple, le fichier "`51-custom.rules`" sera interprété APRES le fichier "`50-udev-default.rules`").

Le fichier "`50-udev-default`" contient les règles *udev* par défaut du système. Pour modifier le comportement par défaut du système, il faut donc créer un fichier qui commence par un nombre plus grand que 50, qui définira un comportement plus spécifique que le défaut du système. Notez que pour ajouter une règle vous aurez besoin d'avoir un accès root sur le système.

Dans le répertoire `udev_conf` de l'archive du *VirtualHub*[2] pour Linux, vous trouverez deux exemples de règles qui vous éviterons de devoir partir de rien.

Exemple 1: 51-yoctopuce.rules

Cette règle va autoriser tous les utilisateurs à accéder en lecture et en écriture aux périphériques Yoctopuce USB. Les droits d'accès pour tous les autres périphériques ne seront pas modifiés. Si ce scénario vous convient il suffit de copier le fichier "`51-yoctopuce_all.rules`" dans le répertoire "`/etc/udev/rules.d`" et de redémarrer votre système.

[1] voir: http://www.yoctopuce.com/FR/blog_by_categories/pour-les-debutants
[2] http://www.yoctopuce.com/EN/virtualhub.php

```
# udev rules to allow write access to all users
# for Yoctopuce USB devices
SUBSYSTEM=="usb", ATTR{idVendor}=="24e0", MODE="0666"
```

Exemple 2: 51-yoctopuce_group.rules

Cette règle va autoriser le groupe "yoctogroup" à accéder en lecture et écriture aux périphériques Yoctopuce USB. Les droits d'accès pour tous les autres périphériques ne seront pas modifiés. Si ce scénario vous convient il suffit de copier le fichier "`51-yoctopuce_group.rules`" dans le répertoire "`/etc/udev/rules.d`" et de redémarrer votre système.

```
# udev rules to allow write access to all users of "yoctogroup"
# for Yoctopuce USB devices
SUBSYSTEM=="usb", ATTR{idVendor}=="24e0", MODE="0664", GROUP="yoctogroup"
```

25.3. Plateformes ARM: HF et EL

Sur ARM il existe deux grandes familles d'executables: HF (Hard Float) et EL (EABI Little Endian). Ces deux familles ne sont absolument pas compatibles entre elles. La capacité d'une machine ARM à faire tourner des exécutables de l'une ou l'autre de ces familles dépend du hardware et du système d'exploitation. Les problèmes de compatibilité entre ArmHL et ArmEL sont assez difficiles à diagnostiquer, souvent même l'OS se révèle incapable de distinguer un exécutable HF d'un exécutable EL.

Tous les binaires Yoctopuce pour ARM sont fournis pré-compilée pour ArmHF et ArmEL, si vous ne savez à quelle famille votre machine ARM apartient, essayez simplement de lancer un exécutable de chaque famille.

25.4. Les exemples de programmation n'ont pas l'air de marcher

La plupart des exemples de programmation de l'API Yoctopuce sont des programmes en ligne de commande et ont besoin de quelques paramètres pour fonctionner. Vous devez les lancer depuis l'invite de commande de votre système d'exploitation ou configurer votre IDE pour qu'il passe les paramètres corrects au programme [3].

25.5. Module alimenté mais invisible pour l'OS

Si votre Yocto-Motor-DC est branché par USB et que sa LED bleue s'allume, mais que le module n'est pas vu par le système d'exploitation, vérifiez que vous utilisez bien un vrai câble USB avec les fils pour les données, et non pas un câble de charge. Les câbles de charge n'ont que les fils d'alimentation.

25.6. Another process named xxx is already using yAPI

Si lors de l'initialisation de l'API Yoctopuce, vous obtenez le message d'erreur "*Another process named xxx is already using yAPI*", cela signifie qu'une autre application est déjà en train d'utiliser les modules Yoctopuce USB. Sur une même machine, un seul processus à la fois peut accéder aux modules Yoctopuce par USB. Cette limitation peut facilement être contournée en utilisant un VirtualHub et le mode réseau [4].

[3] voir: http://www.yoctopuce.com/FR/article/a-propos-des-programmes-d-exemples
[4] voir: http://www.yoctopuce.com/FR/article/message-d-erreur-another-process-is-already-using-yapi

25.7. Déconnexions, comportement erratique

Si votre Yocto-Motor-DC se comporte de manière erratique et/ou se déconnecte du bus USB sans raison apparente, vérifiez qu'il est alimenté correctement. Evitez les câbles d'une longueur supérieure à 2 mètres. Au besoin, intercalez un hub USB alimenté [5] [6].

25.8. RegisterHub d'un VirtualHub déconnecte le précédent

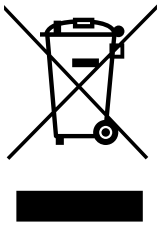
Si lorsque vous faire un YAPI.RegisterHub d'un VirtualHub la connexion avec un autre virtualHub précédement enregistré tombe, vérifiez que les machines qui hébergent ces VirtualHubs on bien un *hostname* différent. Ce cas de figure est très courant avec les machines dont le système d'exploitation est installé avec une image monolithique, comme les Raspberry-PI par exemple. L'API Yoctopuce utilise les numéros de série Yoctopuce pour communiquer et le numéro de série d'un VirtualHub est créé à la volée à partir du *hostname* de la machine qui l'héberge.

25.9. Commandes ignorées

Si vous avez l'impression que des commandes envoyées à un module Yoctopuce sont ignorées, typiquement lorsque vous avez écrit un programme qui sert à configurer ce modules Yoctopuce et qui envoie donc beaucoup de commandes, vérifiez que vous avez bien mis un YAPI.FreeAPI() à la fin du programme. Les commandes sont envoyées aux modules de manière asynchrone grâce à un processus qui tourne en arrière plan. Lorsque le programme se termine, ce processus est tué, même s'il n'a pas eu le temps de tout envoyer. En revanche API.FreeAPI() attend que la file d'attente des commandes à envoyer soit vide avant de libérer les ressources utilisées par l'API et rendre la main.

25.10. Module endommagé

Yoctopuce s'efforce de réduire la production de déchets électroniques. Si vous avez l'impression que votre Yocto-Motor-DC ne fonctionne plus, commencez par contacter le support Yoctopuce par e-mail pour poser un diagnostic. Même si c'est suite à une mauvaise manipulation que le module a été endommagé, il se peut que Yoctopuce puisse le réparer, et ainsi éviter de créer un déchet électronique.



Déchets d'équipements électriques et électroniques (DEEE) Si voulez vraiment vous débarasser de votre Yocto-Motor-DC, ne le jetez pas à la poubelle, mais ramenez-le à l'un des points de collecte proposé dans votre région afin qu'il soit envoyé à un centre de recyclage ou de traitement spécialisé.

[5] voir: http://www.yoctopuce.com/FR/article/cables-usb-la-taille-compte
[6] voir: http://www.yoctopuce.com/FR/article/combien-de-capteurs-usb-peut-on-connecter

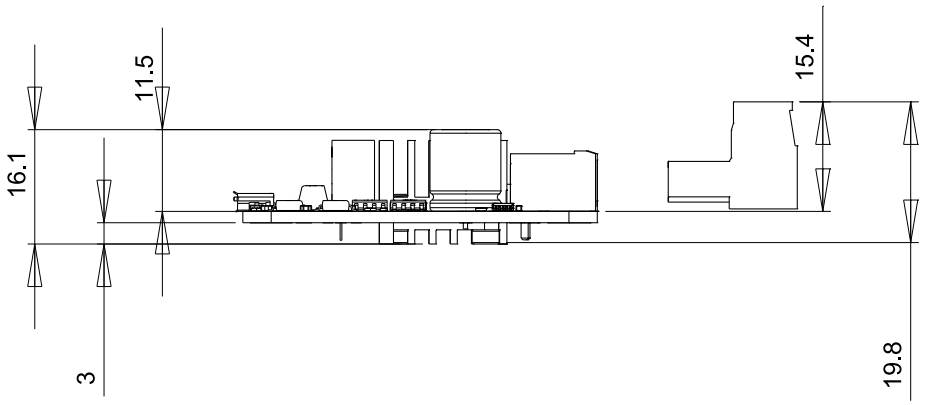
26. Caractéristiques

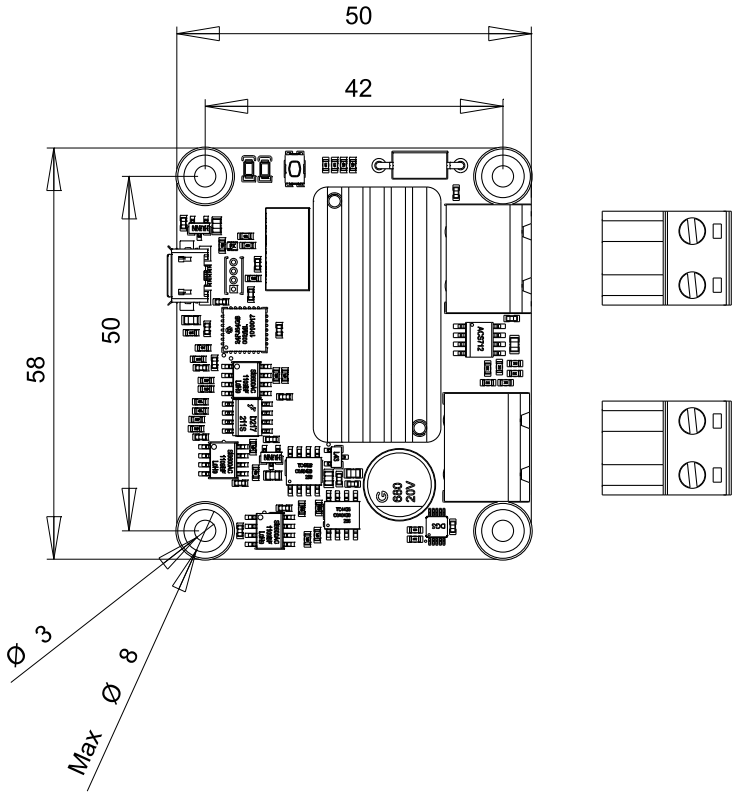
Vous trouverez résumées ci-dessous les principales caractéristiques techniques de votre module Yocto-Motor-DC

Identifiant produit	MOTORCTL
Révision matérielle†	
Connecteur USB	micro-B
Largeur	50 mm
Longueur	58 mm
Poids	36 g
Courant Max (avec ventilation)	20 A
Courant Max (en pointe)	30 A
Courant Max (sans ventilation)	10 A
Fréquence	0.25 ... 32 KHz
Classe de protection selon IEC 61140	classe III
Isolation USB, tension retenue (1 min.)	0.5 kV
Temp. de fonctionnement normale	5...40 °C
Temp. de fonctionnement étendue‡	-30...70 °C
Conformité RoHS	RoHS III (2011/65/UE+2015/863)
USB Vendor ID	0x24E0
USB Device ID	0x0016
Code tarifaire harmonisé	9032.9000
Fabriqué en	Suisse

† Ces spécifications correspondent à la révision matérielle actuelle du produit. Les spécifications des versions antérieures peuvent être inférieures.

‡ La plage de température étendue est définie d'après les spécifications des composants et testée sur une durée limitée (1h). En cas d'utilisation prolongée hors de la plage de température standard, il est recommandé procéder à des tests extensifs avant la mise en production.





All dimensions are in mm
Toutes les dimensions sont en mm

Yocto-MotorDC

A4

Scale
1:1
Echelle