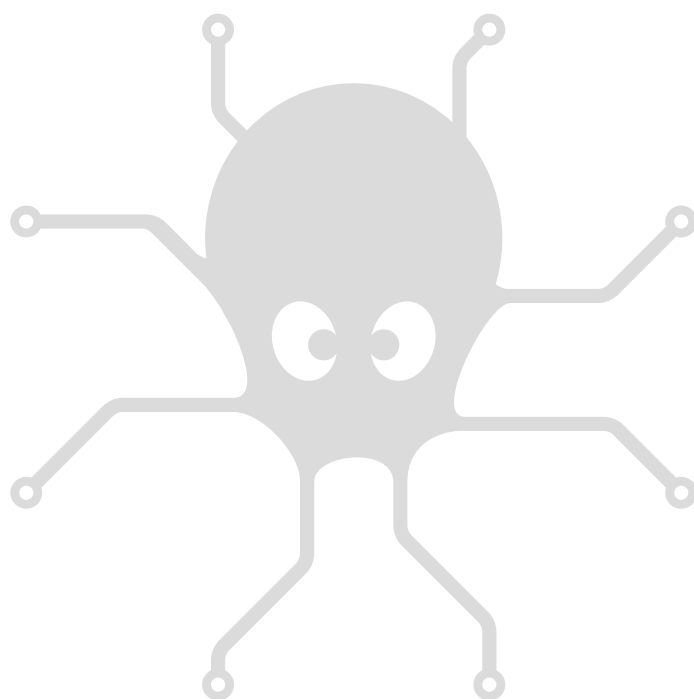




Yocto-Altimeter

Mode d'emploi

Table des matières

1. Introduction	1
1.1. Informations de sécurité	2
1.2. Conditions environnementales	2
2. Présentation	5
2.1. Les éléments communs	5
2.2. Les éléments spécifiques	6
2.3. Accessoires optionnels	7
3. Fonctionnement d'un altimètre barométrique	9
4. Premiers pas	11
4.1. Prérequis	11
4.2. Test de la connectivité USB	12
4.3. Localisation	13
4.4. Test du module	13
4.5. Configuration	14
5. Montage et connectique	17
5.1. Fixation	17
5.2. Déporter le capteur	18
5.3. Contraintes d'alimentation par USB	19
6. Programmation, concepts généraux	21
6.1. Paradigme de programmation	21
6.2. Le module Yocto-Altimeter	23
6.3. Interface de contrôle du module	24
6.4. Interface de la fonction Altitude	25
6.5. Interface de la fonction Pressure	27
6.6. Interface de la fonction Temperature	28
6.7. Interface de la fonction DataLogger	29
6.8. Quelle interface: Native, DLL ou Service?	30
6.9. Programmation, par où commencer?	32

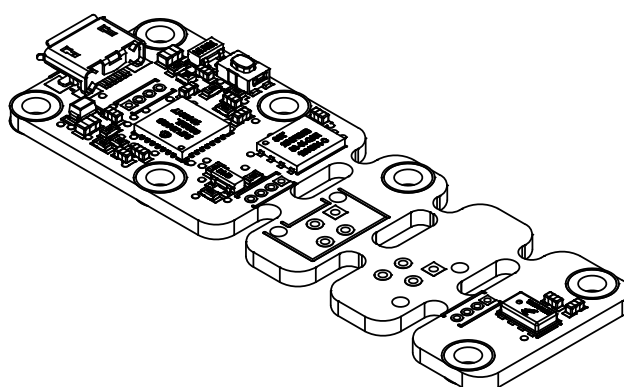
7. Utilisation du Yocto-Altimeter en ligne de commande	35
7.1. <i>Installation</i>	35
7.2. <i>Utilisation: description générale</i>	35
7.3. <i>Contrôle de la fonction Altitude</i>	36
7.4. <i>Contrôle de la partie module</i>	37
7.5. <i>Limitations</i>	37
8. Utilisation du Yocto-Altimeter en JavaScript / EcmaScript	39
8.1. <i>Fonctions bloquantes et fonctions asynchrones en JavaScript</i>	40
8.2. <i>Utiliser la librairie Yoctopuce pour JavaScript / EcmaScript 2017</i>	41
8.3. <i>Contrôle de la fonction Altitude</i>	43
8.4. <i>Contrôle de la partie module</i>	46
8.5. <i>Gestion des erreurs</i>	48
9. Utilisation du Yocto-Altimeter en PHP	51
9.1. <i>Préparation</i>	51
9.2. <i>Contrôle de la fonction Altitude</i>	51
9.3. <i>Contrôle de la partie module</i>	53
9.4. <i>API par callback HTTP et filtres NAT</i>	56
9.5. <i>Gestion des erreurs</i>	59
10. Utilisation du Yocto-Altimeter en C++	61
10.1. <i>Contrôle de la fonction Altitude</i>	61
10.2. <i>Contrôle de la partie module</i>	64
10.3. <i>Gestion des erreurs</i>	66
10.4. <i>Intégration de la librairie Yoctopuce en C++</i>	67
11. Utilisation du Yocto-Altimeter en Objective-C	69
11.1. <i>Contrôle de la fonction Altitude</i>	69
11.2. <i>Contrôle de la partie module</i>	71
11.3. <i>Gestion des erreurs</i>	73
12. Utilisation du Yocto-Altimeter en VisualBasic .NET	75
12.1. <i>Installation</i>	75
12.2. <i>Utilisation l'API yoctopuce dans un projet Visual Basic</i>	75
12.3. <i>Contrôle de la fonction Altitude</i>	76
12.4. <i>Contrôle de la partie module</i>	78
12.5. <i>Gestion des erreurs</i>	80
13. Utilisation du Yocto-Altimeter en C#	83
13.1. <i>Installation</i>	83
13.2. <i>Utilisation l'API yoctopuce dans un projet Visual C#</i>	83
13.3. <i>Contrôle de la fonction Altitude</i>	84
13.4. <i>Contrôle de la partie module</i>	86
13.5. <i>Gestion des erreurs</i>	89
14. Utilisation du Yocto-Altimeter avec Universal Windows Platform	91
14.1. <i>Fonctions bloquantes et fonctions asynchrones</i>	91
14.2. <i>Installation</i>	92
14.3. <i>Utilisation l'API Yoctopuce dans un projet Visual Studio</i>	92
14.4. <i>Contrôle de la fonction Altitude</i>	93

14.5. Un exemple concret	94
14.6. Contrôle de la partie module	95
14.7. Gestion des erreurs	97
15. Utilisation du Yocto-Altimeter en Delphi	99
15.1. Préparation	99
15.2. Contrôle de la fonction Altitude	99
15.3. Contrôle de la partie module	101
15.4. Gestion des erreurs	104
16. Utilisation du Yocto-Altimeter en Python	107
16.1. Fichiers sources	107
16.2. Librairie dynamique	107
16.3. Contrôle de la fonction Altitude	108
16.4. Contrôle de la partie module	109
16.5. Gestion des erreurs	111
17. Utilisation du Yocto-Altimeter en Java	113
17.1. Préparation	113
17.2. Contrôle de la fonction Altitude	113
17.3. Contrôle de la partie module	115
17.4. Gestion des erreurs	118
18. Utilisation du Yocto-Altimeter avec Android	119
18.1. Accès Natif et Virtual Hub.	119
18.2. Préparation	119
18.3. Compatibilité	119
18.4. Activer le port USB sous Android	120
18.5. Contrôle de la fonction Altitude	122
18.6. Contrôle de la partie module	124
18.7. Gestion des erreurs	129
19. Mise à jour du firmware	131
19.1. Le VirtualHub ou le YoctoHub	131
19.2. La librairie ligne de commandes	131
19.3. L'application Android Yocto-Firmware	131
19.4. La librairie de programmation	132
19.5. Le mode "mise à jour"	134
20. Utilisation avec des langages non supportés	135
20.1. Ligne de commande	135
20.2. Virtual Hub et HTTP GET	135
20.3. Utilisation des librairies dynamiques	137
20.4. Port de la librairie haut niveau	140
21. Programmation avancée	141
21.1. Programmation par événements	141
21.2. L'enregistreur de données	144
21.3. Calibration des senseurs	147
22. Référence de l'API de haut niveau	151
22.1. Fonctions générales	152

22.2. Interface de contrôle du module	187
22.3. Interface de la fonction Altitude	253
22.4. Interface de la fonction Pressure	314
22.5. Interface de la fonction Temperature	370
22.6. Interface de la fonction DataLogger	435
22.7. Séquence de données enregistrées	479
22.8. Valeur mesurée	493
23. Problèmes courants	499
23.1. Par où commencer ?	499
23.2. Linux et USB	499
23.3. Plateformes ARM: HF et EL	500
23.4. Les exemples de programmation n'ont pas l'air de marcher	500
23.5. Module alimenté mais invisible pour l'OS	500
23.6. Another process named xxx is already using yAPI	500
23.7. Déconnexions, comportement erratique	501
23.8. Module endommagé	501
24. Caractéristiques	503

1. Introduction

Le Yocto-Altimeter est un module électronique de 60x20mm qui permet d'estimer l'altitude à laquelle il se trouve sur la base d'une mesure de pression atmosphérique. Le module Yocto-Altimeter indique aussi la pression et la température ambiante. La précision de l'estimation d'altitude relative est de l'ordre de 25cm.



Le module Yocto-Altimeter

Le Yocto-Altimeter 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-Altimeter et espère sincèrement qu'il vous donnera entière satisfaction. Les ingénieurs Yoctopuce se sont donné beaucoup de mal pour que votre Yocto-Altimeter 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-Altimeter 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-Altimeter 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 dégât 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-Altimeter, 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'eau et au frottement usuel qui peut survenir durant un entretien usuel.

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

1.2. Conditions environnementales

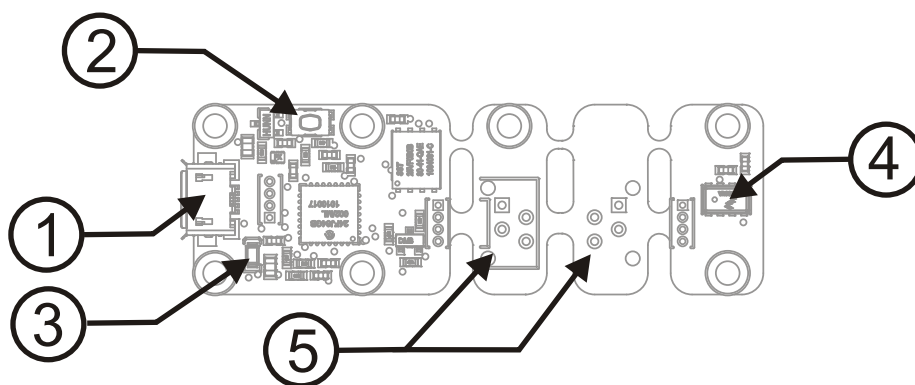
Les produits Yoctopuce sont conçus pour une utilisation intérieure dans un environnement usuel de bureau ou de laboratoire (*degré de pollution 2* selon IEC 60664): la pollution de l'air doit être faible et essentiellement non conductrice. L'humidité relative prévue est de 10% à 90% RH, sans condensation. 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-Altimeter est -10...85°C. Cette plage de température a été déterminée en fonction des recommandations officielles des fabricants des composants utilisés dans le Yocto-Altimeter, et par des tests de durée limitée (1h) dans les

conditions extrêmes, en environnement contrôlé. Si vous envisagez d'utiliser le Yocto-Altimeter 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 4: Capteur de pression
2: Yocto-bouton 5: Emplacements pour les embases Picoflex
3: Yocto-led

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

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 morse¹. 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 Yoctopuce².

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-Altimeter ce numéro commence par YALTIMK1. 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 chaîne 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 mêmes 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 modules 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

Le capteur

Le capteur MPL3115A2 est fabriqué par [Freescale](#). Il permet de mesurer très précisément la pression atmosphérique entre 500 et 1100 millibars³, ce qui correspond en atmosphère standard à une plage de -2300m à 5500m, mais il continuera à fonctionner correctement jusqu'à 200 millibars, soit environ 11500 mètres...

¹ court-court-court long-long-long court-court-court

² support@yoctopuce.com

³ 1000 millibar = 1000 hPa = ~14.5 psi = ~29.53 inHg

2.3. Accessoires optionnels

Les accessoires ci-dessous ne sont pas nécessaires à l'utilisation du module Yocto-Altimeter, 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-Altimeter à un support, vous pouvez placer des petites vis de 2.5mm avec une tête de 4.5mm 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-Altimeter 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-Altimeter à un Micro-hub USB ou à un YoctoHub en évitant l'encombrement d'un vrai câble USB, vous pouvez utiliser les 4 pads au pas 1.27mm juste derrière le connecteur USB. Vous avez alors deux possibilités.

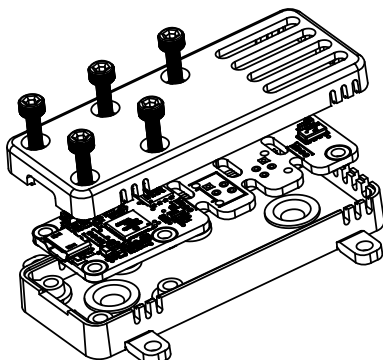
Vous pouvez monter directement le module sur le hub à l'aide d'un jeu de vis et entretoises, et les connecter à l'aide de connecteurs board-to-board au pas 1.27mm. Pour éviter les court-circuits, soudez de préférence le connecteur femelle sur le hub et le connecteur mâle sur le Yocto-Altimeter.

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-Altimeter a été conçu pour pouvoir être installé tel quel dans votre projet. Néanmoins Yoctopuce commercialise des boîtiers spécialement conçus pour les modules Yoctopuce. Ces boîtiers sont munis de pattes de fixation amovibles et d'aimants de fixation. Vous trouverez plus d'informations à propos de ces boîtiers sur le site de Yoctopuce⁴. Le boîtier recommandé pour votre Yocto-Altimeter est le modèle YoctoBox-Long-Thin-Black-Vents

⁴ <http://www.yoctopuce.com/EN/products/category/enclosures>



Vous pouvez installer votre Yocto-Altimeter dans un boîtier optionnel.

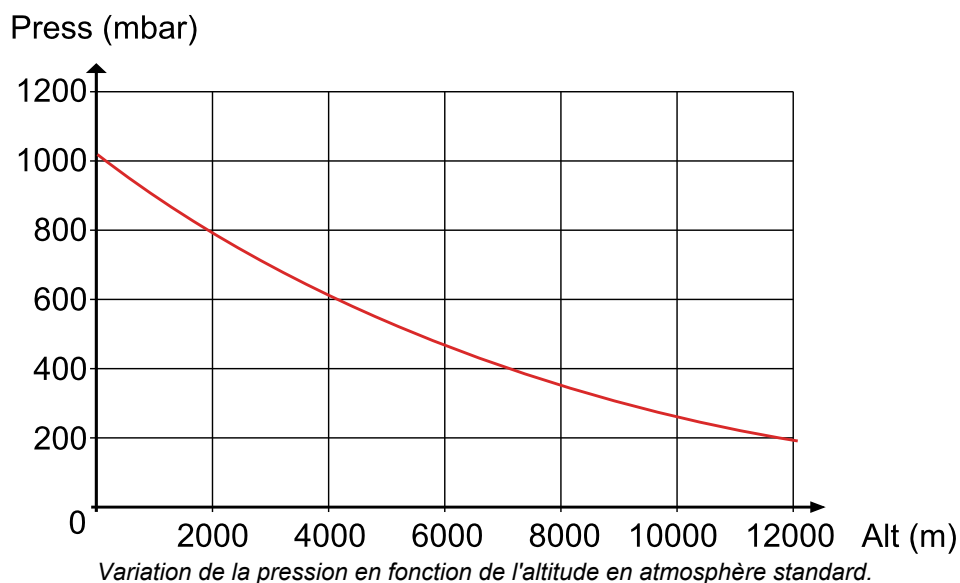
Connecteurs Picoflex et câble nappe souple

Si vous désirez déporter la partie capteur du module Yocto-Altimeter à l'aide d'un câble à connecteur enfichable, vous aurez besoin de câble nappe souple à 4 fils espacés de 1.27mm et de connecteurs Picoflex.⁵ Vous trouverez plus de détail à ce sujet dans le chapitre concernant le montage et la connectique.

⁵ Embases Molex ref 90325-3004 ou 90325-0004, disponibles chez la plupart des fournisseurs de composants électroniques (www.mouser.com, www.digikey.com, www.farnell.com, www.distrelec.ch...). S'utilise avec les connecteurs ref 90327-3304 ou 90327-0304

3. Fonctionnement d'un altimètre barométrique

Un altimètre barométrique mesure la pression atmosphérique à l'endroit où il se trouve, calcule la différence par rapport à la pression au niveau de la mer puis en déduit l'altitude. En effet, plus on monte haut dans l'atmosphère, plus la pression baisse.



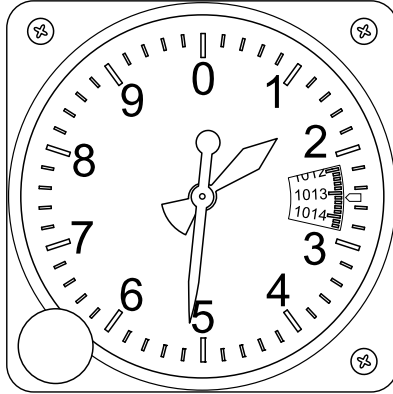
L'atmosphère considérée comme standard au niveau de la mer est à une pression 1013.25 millibars à 15°C. Mais en réalité cette pression de référence au niveau de la mer subit des variations significatives en fonction des conditions météo.

C'est pourquoi, pour être précis, un altimètre doit régulièrement être réglé. Deux méthodes peuvent être utilisées en fonction des informations disponibles:

- L'altitude actuelle est connue: on calibre l'altimètre sur cette altitude, et tout changement d'altitude ultérieur sera alors mesuré précisément. Ainsi, à la mise en route, les avions règlent toujours leur altimètre à l'altitude de l'aéroport où ils se trouvent.
- La pression au niveau de la mer est connue. Une fois cette pression de référence donnée à l'altimètre, il sera capable de calculer précisément l'altitude. En aviation, cette pression de référence est appelée le QNH. Il très utilisé, au point que chaque aéroport le communique à chaque avion qui entre en contact avec lui. Un avion en vol peut ainsi recalibrer régulièrement son altimètre, ce qui est indispensable en raison des changements de météo et donc de pression d'un endroit à l'autre. Par conséquent, si vous désirez connaître le QNH de l'endroit

ou vous vous trouvez, vous pourrez l'obtenir grâce aux informations météorologiques de l'aéroport le plus proche (cherchez sur Google le code de l'aéroport suivi de QNH).

Pour résumer l'altitude, la pression ambiante et le QNH sont interdépendants: lorsque l'altitude et la pression ambiante sont connues, on peut en déduire le QNH, de même lorsque le QNH et la pression ambiante sont connus, on peut en déduire l'altitude.



Un altimètre d'avion affiche toujours à la fois l'altitude et le QNH.

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-Altimeter, 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 fournit les bibliothèques 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 HID¹ 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 bibliothèque 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.

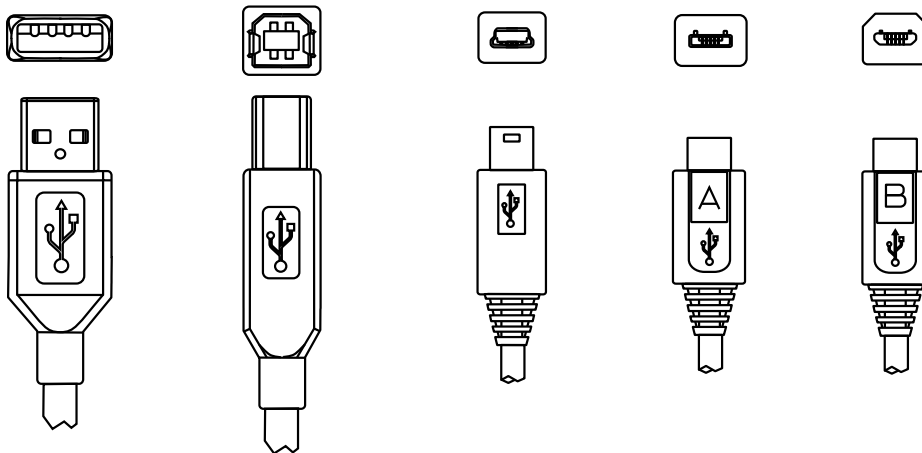
Les versions de Linux supportées sont les kernels 2.6, 3.x et 4.x. D'autres versions du kernel et même d'autres variantes d'Unix sont très susceptibles d'être utilisées sans problème, puisque le support de Linux est fait via l'API standard de la **libusb**, disponible aussi pour FreeBSD par exemple. 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.

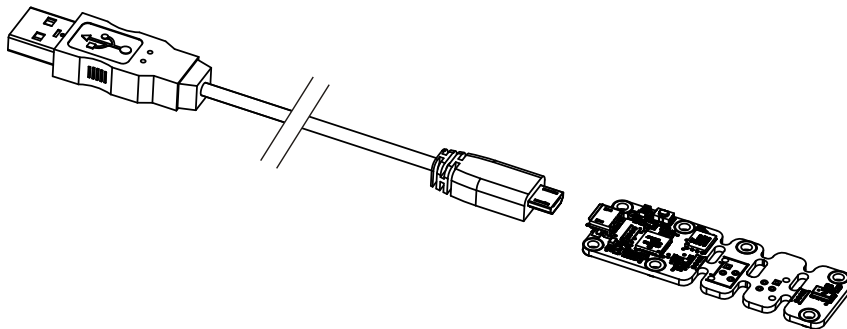
Un câble USB 2.0 de type A-micro B

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-Altimeter à un ordinateur, vous avez besoin d'un câble USB 2.0 de type A-micro B. Vous trouverez ce câble 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-Altimeter à l'aide d'un câble USB 2.0 de type A - micro B

Si vous branchez un hub USB entre l'ordinateur et le module Yocto-Altimeter, 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.

4.2. Test de la connectivité USB

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

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

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 web⁴. 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.

Serial	Logical Name	Description	Action
VIRTHUB0-1521ca755		VirtualHub	configure view log file
YALTIMK1-20DA0		Yocto-Altimeter	configure view log file beacon

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-Altimeter.

YALTIMK1-20DA0



YALTIMK1-20DA0 is a 20x60mm board featuring an altimeter, as well as temperature and pressure sensors.

Kernel

Serial #

Product name:

Logical name:

Firmware:

Consumption:

Beacon:

Luminosity:

YALTIMK1-20DA0

Yocto-Altimeter

15783

25 mA

Inactive

50%

turn on

Sensors

	Altitude	Pressure	Temperature
Current value	468.7 m	968.2 mbar	28.9 °C
Minimum value	468 m	0 mbar	0 °C
Maximum value	469 m	968 mbar	28.9 °C

Misc

Open API browser (pop-up)

Get user manual from yoctopuce.com

Close

Propriétés du module Yocto-Altimeter.

Cette fenêtre vous permet entre autres de jouer avec votre module pour en vérifier son fonctionnement, les valeurs d'altitude, de pression et de température y sont en effet affichées en temps réel.

³ www.yoctopuce.com/FR/virtualhub.php

⁴ L'interface est testée avec Chrome, FireFox, Safari, Edge et IE 11.

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-Altimeter.

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 commande⁶.

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.

Le module Yocto-Altimeter dispose de trois fonctions: "altitude", "temperature" et "pression". Cliquez simplement sur le bouton "rename" correspondant pour leur affecter un nouveau nom logique.

⁵ www.yoctopuce.com/FR/virtualhub.php

⁶ Consultez la documentation du virtual hub pour plus de détails

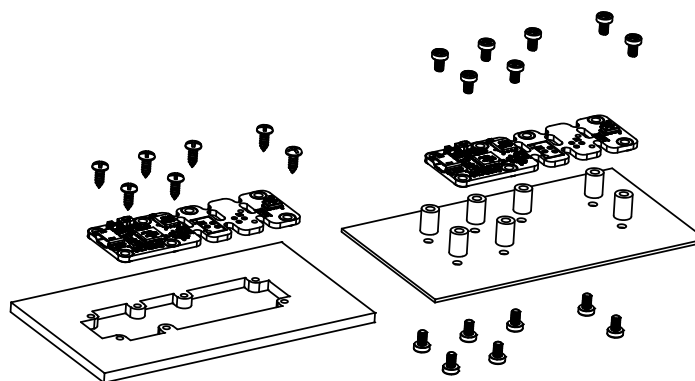
La fenêtre de configuration permet aussi de régler l'altimètre, soit en forçant l'altitude à une valeur connue, ce qui permet au module d'en déduire le QNH; soit en réglant directement QNH. Le QNH est sauvé dans la mémoire flash du module, ce qui signifie que le module s'en rappellera même s'il a été privé de courant.

5. Montage et connectique

Ce chapitre fournit des explications importantes pour utiliser votre module Yocto-Altimeter 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. 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). Une fois votre projet pratiquement terminé il faudra penser à faire en sorte que vos modules ne puissent pas se promener à l'intérieur.



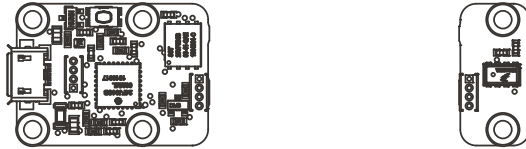
Exemples de montage sur un support.

Le module Yocto-Altimeter dispose de trous de montage 2.5mm. Vous pouvez utiliser ces trous pour y passer des vis. Le diamètre de la tête de ces vis ne devra pas dépasser 4.5mm, sous peine d'endommager les circuits du module. Veillez à 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. Rien ne vous empêche de le fixer au pistolet à colle; ça ne sera pas très joli mais ça tiendra.

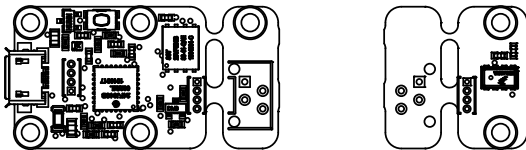
Si vous comptez visser votre module directement contre une paroi conductrice, un châssis métallique par exemple, intercalez une couche isolante entre les deux. Sinon vous allez à coup sûr provoquer un court-circuit: il y a des pads à nu sous votre module. Du simple ruban adhésif isolant devrait faire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

5.2. Déporter le capteur

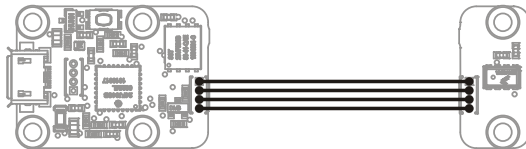
Le module Yocto-Altimeter est conçu pour pouvoir être séparé en deux morceaux afin de vous permettre de déporter le capteur. Vous pouvez les séparer en cassant simplement le circuit, mais vous obtiendrez un meilleur résultat en utilisant une bonne paire de tenailles, ou une grosse pince coupante. Une fois les deux sous-modules séparés vous pouvez poncer sans risque les parties qui dépassent.



Le module Yocto-Altimeter est conçu pour pouvoir être séparé en deux parties.

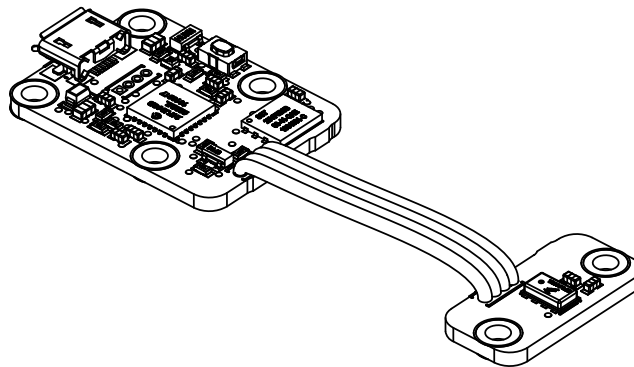


.. et vous avez deux possibilités en fonction du type de connexion que vous désirez utiliser.



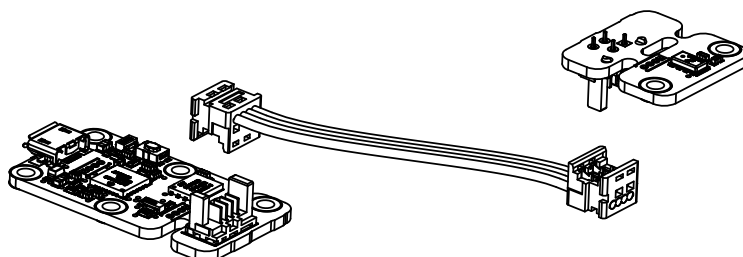
Câblage des sous modules une fois ceux-ci séparés.

Une fois les modules séparés vous allez devoir les re-câbler. Plusieurs solutions s'offrent à vous. Vous pouvez raccorder les sous-modules en soudant des fils électriques tout simples, mais vous obtiendrez un meilleur résultat avec du câble nappe au pas 1.27 mm. Utilisez de préférence du câble avec des conducteurs mono-brin plutôt que du multi-brin: les câbles mono-brin sont un peu moins souples, mais nettement plus facile à souder.



Déport du capteur à l'aide de simple câble nappe.

Vous pouvez aussi utiliser des câbles nappe munis de connecteurs Picoflex, Vous obtiendrez un système un peu plus gros, mais les embases Picoflex sont beaucoup plus faciles à souder que des câbles nappe. De plus le résultat sera démontable.



Déport du capteur à l'aide de connecteurs Picoflex.

Veillez à ne pas placer la partie capteur trop près d'une source de chaleur où dans un courant d'air ce qui pourrait altérer les mesure de manière significative..

Attention, les modules Yoctopuce sécables ont souvent des systèmes de connectique très semblables. Cependant les sous-modules ne sont pas du tout compatibles entre modèles différents. Si vous raccordez un sous module de votre Yocto-Altimeter à un autre type de module, par exemple un Yocto-Temperature, cela ne marchera pas, et vous risquez fort d'endommager votre matériel.

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-Altimeter 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.

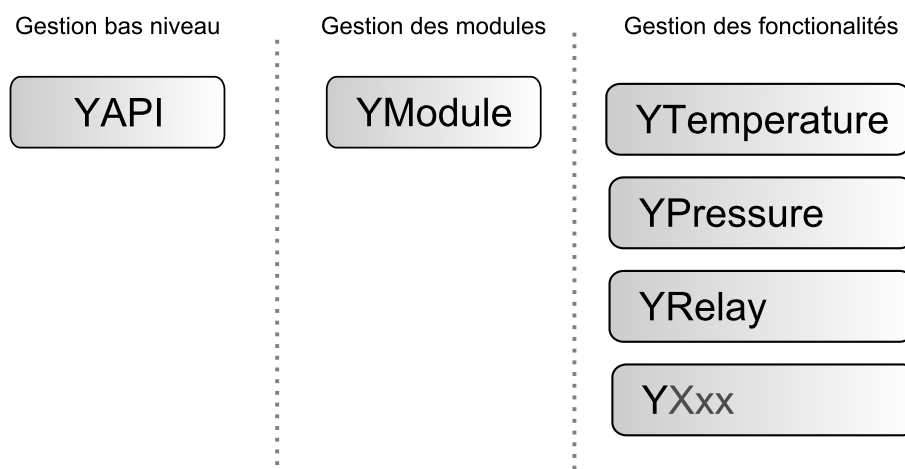
6. Programmation, concepts généraux

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-Altimeter 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 aux 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-Altimeter 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-Altimeter

Le module Yocto-Altimeter offre une instance de la fonction Altitude correspondant à une estimation de l'altitude, avec une sensibilité d'environ 0.25m. Il indique aussi la pression atmosphérique et la température mesurée.

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

altitude : Altitude

attribut	type	modifiable ?
logicalName	Texte	modifiable
advertisedValue	Texte	modifiable
unit	Texte	lecture seule
currentValue	Nombre (virgule fixe)	modifiable
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	Paramètres de calibration	modifiable
resolution	Nombre (virgule fixe)	modifiable
sensorState	Nombre entier	lecture seule
qnh	Nombre (virgule fixe)	modifiable
technology	Texte	lecture seule

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	Paramètres 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

pressure : Pressure

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	Paramètres de calibration	modifiable
resolution	Nombre (virgule fixe)	modifiable
sensorState	Nombre entier	lecture seule

dataLogger : DataLogger

attribut	type	modifiable ?
logicalName	Texte	modifiable
advertisedValue	Texte	modifiable
currentRunIndex	Nombre entier	lecture seule
timeUTC	Heure UTC	modifiable
recording	Type énuméré	modifiable
autoStart	On/Off	modifiable
beaconDriven	On/Off	modifiable
clearHistory	Booléen	modifiable

6.3. Interface de contrôle du module

Cette interface est la même pour tous les modules USB de Yoctopuce. Elle permet de contrôler les paramètres généraux du module, et d'énumérer les fonctions fournies par chaque module.

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-Altimeter, ce numéro de série commence toujours par YALTIMK1. 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 63 en usine.

productRelease

Numéro de révision du module hardware, préprogrammé en usine.

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. Interface de la fonction Altitude

La classe YAltitude permet de lire et de configurer les capteurs d'altitude Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks,

enregistreur de données. De plus, pour les capteurs barométriques, elle permet de faire la configuration de la pression de référence au niveau de la mer (QNH).

logicalName

Chaîne de caractères contenant le nom logique de l'altimètre, 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 directement à l'altimètre. Si deux altimètres 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 de l'altimètre, et qui sera publiée automatiquement jusqu'au hub parent. Pour un altimètre, la valeur publiée est la valeur courante de l'altitude.

unit

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

currentValue

Valeur actuelle de l'altitude, en mètres, sous forme de nombre à virgule.

lowestValue

Valeur minimale de l'altitude, en mètres, sous forme de nombre à virgule.

highestValue

Valeur maximale de l'altitude, en mètres, 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).

qnh

Pression atmosphérique actuelle au niveau de la mer.

technology

Technologie utilisée

6.5. Interface de la fonction Pressure

La classe YPressure permet de lire et de configurer les capteurs de pression Yoctopuce. Elle hérite de la class 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 pression, 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 pression. Si deux capteurs de pression 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 pression, et qui sera publiée automatiquement jusqu'au hub parent. Pour un capteur de pression, la valeur publiée est la valeur courante de la pression.

unit

Courte chaîne de caractères représentant l'unité dans laquelle la pression est exprimée.

currentValue

Valeur actuelle de la pression, en millibar (hPa), sous forme de nombre à virgule.

lowestValue

Valeur minimale de la pression, en millibar (hPa), sous forme de nombre à virgule.

highestValue

Valeur maximale de la pression, en millibar (hPa), 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).

6.6. Interface de la fonction Temperature

La classe YTemperature permet de lire et de configurer les capteurs de température Yoctopuce. Elle hérite de la class 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 caractères 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 ou encore un thermistor.

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 caractères représentant l'unité du signal électrique utilisé par le capteur.

command

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

6.7. Interface de la fonction DataLogger

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

logicalName

Chaîne de caractères contenant le nom logique de l'enregistreur de données, 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 directement à l'enregistreur de données. Si deux enregistreurs de données 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 de l'enregistreur de données, et qui sera publiée automatiquement jusqu'au hub parent. Pour un enregistreur de données, la valeur publiée est son état d'activation (ON ou OFF).

currentRunIndex

Numéro du Run actuel, correspondant au nombre de fois que le module a été mis sous tension avec la fonction d'enregistreur de données active.

timeUTC

Heure UTC courante, lorsque l'on désire associer une référence temporelle absolue aux données enregistrées. Cette heure doit être configurée explicitement par logiciel.

recording

Etat d'activité de l'enregistreur de données. L'enregistreur peut être enclenché ou déclenché à volonté par cet attribut, mais son état à la mise sous tension est déterminé par l'attribut persistant **autoStart**. Lorsque l'enregistreur est enclenché mais qu'il n'est pas encore prêt pour enregistrer, son état est PENDING.

autoStart

Enclenchement automatique de l'enregistreur de données à la mise sous tension. Cet attribut permet d'activer systématiquement l'enregistreur à la mise sous tension, sans devoir l'activer par une commande logicielle.

beaconDriven

Permet de synchroniser l'état de la balise de localisation avec l'état de l'enregistreur de données. Quand cet attribut est activé il est possible de démarrer et arrêter l'enregistrement en utilisant le Yocto-bouton du module ou l'attribut `beacon` de la fonction `YModule`. De la même manière si l'attribut `recording` de la fonction `datalogger` est modifié, l'état de la balise de localisation est mis à jour. Note: quand cet attribut est activé balise de localisation du module clignote deux fois plus lentement.

clearHistory

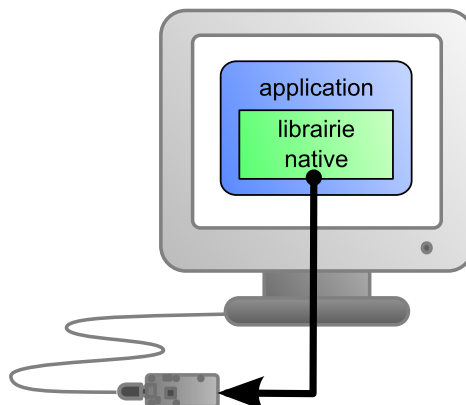
Attribut qui peut être mis à vrai pour effacer l'historique des mesures.

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.

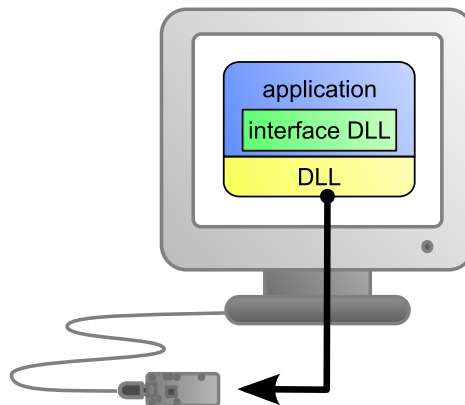


L'application utilise la librairie native pour contrôler le module connecté en local

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 langage 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

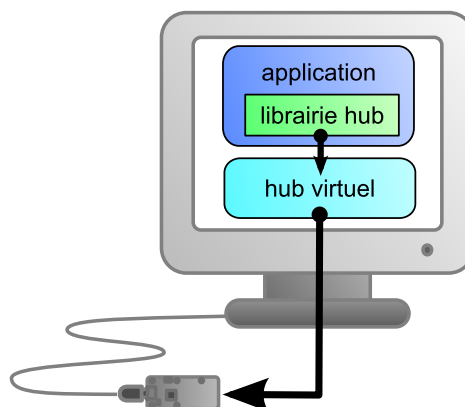
un nouveau langage se limite à coder la petite librairie qui contrôle la DLL. Du point de de l'utilisateur final, il y a peu de différence: il faut simplement être sur que la DLL sera installée sur son ordinateur en même temps que le programme principal.



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

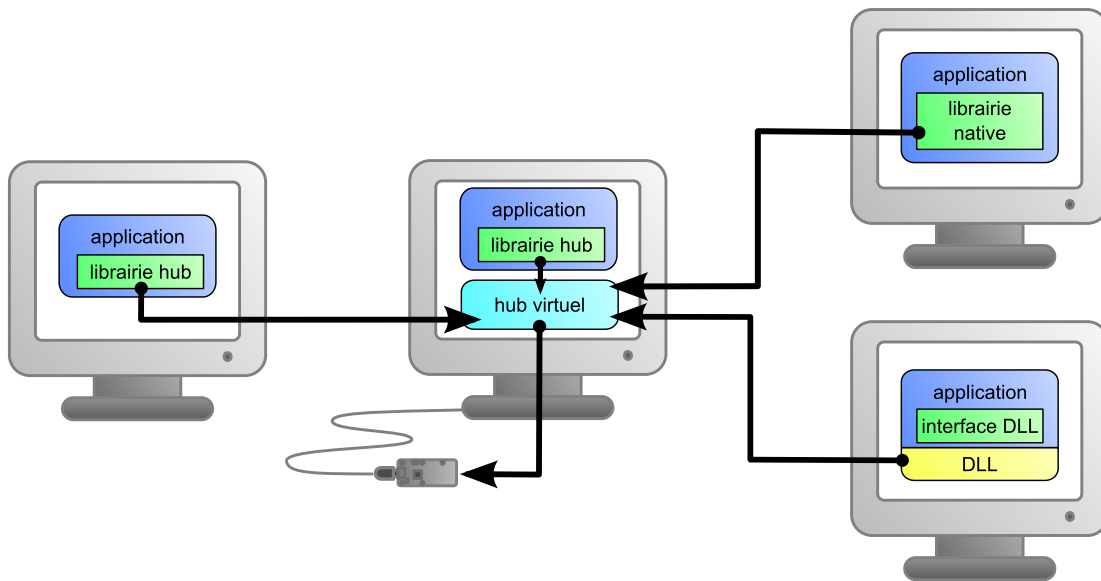
Contrôle par un service

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	Hub virtuel
C++	✓	✓	✓
Objective-C	✓	-	✓
Delphi	-	✓	✓
Python	-	✓	✓
VisualBasic .Net	-	✓	✓
C# .Net	-	✓	✓
C# UWP	✓	-	✓
EcmaScript / JavaScript	-	-	✓
PHP	-	-	✓
Java	-	✓	✓
Java pour Android	✓	-	✓
Ligne de commande	✓	-	✓

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-Altimeter. Il est temps de passer à la pratique. Il vous faut télécharger la librairie Yoctopuce pour votre langage de programmation favori depuis le site web de Yoctopuce¹. 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.

¹ <http://www.yoctopuce.com/FR/libraries.php>

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-Altimeter.

7. Utilisation du Yocto-Altimeter en ligne de commande

Lorsque vous désirez effectuer une opération ponctuelle sur votre Yocto-Altimeter, 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 fournies¹.

7.1. Installation

Téléchargez l'API en ligne de commande². 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-Altimeter, ouvrir un shell et commencer à travailler en tapant par exemple:

```
C:\>YAltitude 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 sauvegarder 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 Altitude

Pour contrôler la fonction Altitude de votre Yocto-Altimeter, vous avez besoin de l'exécutable YAltitude.

Vous pouvez par exemple lancer:

```
C:\>YAltitude any get_currentValue
```

Cet exemple utilise la cible "any" pour signifier que l'on désire travailler sur la première fonction Altitude 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-Altimeter avec le numéro de série *YALTIMK1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction altitude "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté).

```
C:\>YAltitude YALTIMK1-123456.altitude describe
C:\>YAltitude YALTIMK1-123456.MaFonction describe
C:\>YAltitude MonModule.altitude describe
C:\>YAltitude MonModule.MaFonction describe
C:\>YAltitude MaFonction describe
```

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

```
C:\>YAltitude all describe
```

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

```
C:\>YAltitude /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 YALTIMK1-12346 set_logicalName MonPremierModule
C:\>YModule YALTIMK1-12346 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 YALTIMK1-12346 set_logicalName MonPremierModule
C:\>YModule YALTIMK1-12346 saveToFlash
```

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

```
C:\>YModule -s YALTIMK1-12346 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 alentours 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 qu'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 VirtualHub³ 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-Altimeter 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 différencier 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 librairie vers un standard antérieur de JavaScript, comme décrit un peu plus bas.

Nous ne recommandons plus l'utilisation de `jspm 0.17` puisque cet outil est toujours en version Beta après 18 mois, et que solliciter l'utilisation d'un outil supplémentaire pour utiliser notre librairie ne se justifie plus dès lors que `async / await` sont standardisés.

8.1. Fonctions bloquantes et fonctions asynchrones en JavaScript

JavaScript a été conçu pour éviter toute situation de *concurrency* 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. 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.

Lorsqu'on les utilise avec des simples fonctions de callback, comme c'est fait quasi systématiquement dans les bibliothèques Node.js, les opérations asynchrones ont la fâcheuse tendance de rendre 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 promesse retournée par la fonction appelée à une promesse de continuer 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 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 bibliothèque 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 bibliothèque 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.

8.2. 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 2017¹
- Le programme VirtualHub² 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 Babel³ 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 `compat` 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 scripts codés en dur 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 dispose de nombreuses fonctionnalités intéressantes, comme un mode de surveillance qui traduit 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 valeur 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);
```

8.3. Contrôle de la fonction Altitude

Il suffit de quelques lignes de code pour piloter un Yocto-Altimeter. Voici le squelette d'un fragment de code JavaScript qui utilise la fonction Altitude.

```
// En Node.js, on utilise la fonction require()
// En HTML, on utiliserait <script src="...">
require('yoctolib-es2017/yocto_api.js');
require('yoctolib-es2017/yocto_altitude.js');

// On récupère l'objet représentant le module, à travers le VirtualHub local
await YAPI.RegisterHub('127.0.0.1');
var altitude = YAltitude.FindAltitude("YALTIMK1-123456.altitude");

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

Voyons maintenant en détail ce que font ces quelques lignes.

Require de yocto_api et yocto_altitude

Ces deux imports permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api` doit toujours être inclus, `yocto_altitude` est nécessaire pour gérer les modules contenant un altimètre, comme le Yocto-Altimeter. 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 déclenche une exception.

YAltitude.FindAltitude

La méthode `FindAltitude`, permet de retrouver un altimètre 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-Altimeter avec le numéros de série `YALTIMK1-123456` que vous auriez appelé "*MonModule*" et dont vous auriez

nommé la fonction *altitude* "MaFonction", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
altitude = YAltitude.FindAltitude("YALTIMK1-123456.altitude")
altitude = YAltitude.FindAltitude("YALTIMK1-123456.MaFonction")
altitude = YAltitude.FindAltitude("MonModule.altitude")
altitude = YAltitude.FindAltitude("MonModule.MaFonction")
altitude = YAltitude.FindAltitude("MaFonction")
```

`YAltitude.FindAltitude` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'altimètre.

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `YAltitude.FindAltitude` permet d'obtenir l'altitude actuelle mesurée par le capteur. La valeur de retour est un nombre flottant en mètres.

YTemperature.FindTemperature et YPressure.FindPressure

Les fonctions `YTemperature.FindTemperature` et `YPressure.FindPressure` permettent de travailler respectivement avec les mesures de température et de pression, et se manipulent exactement comme `YAltitude.FindAltitude`.

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-Altimeter** 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-Altimeter 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-Altimeter et où vous avez lancé le VirtualHub.

```
"use strict";

require('yoctolib-es2017/yocto_api.js');
require('yoctolib-es2017/yocto_altitude.js');
require('yoctolib-es2017/yocto_temperature.js');
require('yoctolib-es2017/yocto_pressure.js');

let alti, temp, pres;

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 = YAltitude.FirstAltitude();
        if(anysensor) {
            let module = await anysensor.module();
```

```

        serial = await module.get_serialNumber();
    } else {
        console.log('No matching sensor connected, check cable !');
        return;
    }
}
console.log('Using device '+serial);
alti = YAltitude.FindAltitude(serial+".altitude");
temp = YTemperature.FindTemperature(serial+".temperature");
pres = YPressure.FindPressure(serial+".pressure");

refresh();
}

async function refresh()
{
    if (await alti.isOnline()) {
        console.log('Altitude      : +(await alti.get_currentValue()) + ' m ' +
                    ' (QNH = +(await alti.get_qnh())+ ' hPa)');
        console.log('Temperature : +(await temp.get_currentValue()) + ' Â°C');
        console.log('Pressure     : +(await pres.get_currentValue()) + ' hPa');
    } 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-Altimeter**. 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_temperature.js"></script>
  <script src="../../lib/yocto_altitude.js"></script>
  <script src="../../lib/yocto_pressure.js"></script>
  <script>
    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);
      }
      refresh();
    }

    async function refresh()
    {

```

```

let serial = document.getElementById('serial').value;
if(serial == '') {
  // by default use any connected module suitable for the demo
  let anysensor = YAltitude.FirstAltitude();
  if(anysensor) {
    let module = await anysensor.module();
    serial = await module.get_serialNumber();
    document.getElementById('serial').value = serial;
  }
}
let alti = YAltitude.FindAltitude(serial+".altitude");
let temp = YTemperature.FindTemperature(serial+".temperature");
let pres = YPressure.FindPressure(serial+".pressure");

if (await alti.isOnline()) {
  document.getElementById('msg').value = '';
  document.getElementById("qnh").value = await alti.get_qnh();
  document.getElementById("alti").value = await alti.get_currentValue();
  document.getElementById("temp").value = await temp.get_currentValue();
  document.getElementById("pres").value = await pres.get_currentValue();
} else {
  document.getElementById('msg').value = 'Module not connected';
}

setTimeout(refresh, 500);
}

startDemo();
</script>
</head>
<body>
Module to use: <input id='serial'>
<input id='msg' style='color:red;border:none;' readonly><br>
Current altitude : <input id='alti' readonly> m
(QNH: <input id='qnh' readonly></span> hPa)<br>
Current temperature : <input id='temp' readonly> &deg;<br>
Current pressure : <input id='pres' readonly> hPa<br>
</body>
</html>

```

Aucune installation n'est nécessaire pour utiliser cet exemple, il suffit d'ouvrir la page HTML avec un navigateur web.

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.

```

"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 au 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 alentours 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);
}
```

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 `Y_STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `Y_CURRENTVALUE_INVALID`, etc. Dans tous les cas, la valeur retournée sera du type attendu, et ne sera pas un pointeur nul qui risquerait de faire crasher votre programme. Au pire, si vous affichez la valeur sans la tester, elle sera hors du cadre attendu pour la valeur retournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI_SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

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

9. Utilisation du Yocto-Altimeter 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 où 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 souhaitez 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 forward les requêtes TCP sur le port 4444.

9.1. Préparation

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

- La librairie de programmation pour PHP²
- Le programme VirtualHub³ 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.

9.2. Contrôle de la fonction Altitude

Il suffit de quelques lignes de code pour piloter un Yocto-Altimeter. Voici le squelette d'un fragment de code PHP qui utilise la fonction Altitude.

```
include('yocto_api.php');  
include('yocto_altitude.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 récupère l'objet représentant le module, à travers le VirtualHub local
yRegisterHub('http://127.0.0.1:4444/', $errmsg);
$altitude = yFindAltitude("YALTIMK1-123456.altitude");

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.php et yocto_altitude.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_altitude.php` est nécessaire pour gérer les modules contenant un altimètre, comme le Yocto-Altimeter.

yRegisterHub

La fonction `yRegisterHub` permet d'indiquer sur quelle machine se trouve les modules Yoctopuce, ou plus exactement 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.

yFindAltitude

La fonction `yFindAltitude`, permet de retrouver un altimètre 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-Altimeter avec le numéros de série `YALTIMK1-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *altitude* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
$altitude = yFindAltitude("YALTIMK1-123456.altitude");
$altitude = yFindAltitude("YALTIMK1-123456.MaFonction");
$altitude = yFindAltitude("MonModule.altitude");
$altitude = yFindAltitude("MonModule.MaFonction");
$altitude = yFindAltitude("MaFonction");
```

`yFindAltitude` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'altimètre.

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `yFindAltitude` permet d'obtenir l'altitude actuelle mesurée par le capteur. La valeur de retour est un nombre flottant en mètres.

yFindTemperature et yFindPressure

Les fonctions `yFindTemperature` et `yFindPressure` permettent de travailler respectivement avec les mesures de température et de pression, et se manipulent exactement comme `yFindAltitude`.

Un exemple réel

Ouvrez votre éditeur de texte préféré⁴, 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 **Exemples/Doc-GettingStarted-Yocto-Altimeter** 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>
  <?php
    include('yocto_api.php');
    include('yocto_altitude.php');
    include('yocto_pressure.php');
    include('yocto_temperature.php');

    // Use explicit error handling rather than exceptions
    yDisableExceptions();

    // Setup the API to use the VirtualHub on local machine
    if(yRegisterHub('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
        $salt = yFindAltitude("$serial.altitude");
        if (!$salt->isOnline()) {
            die("Module not connected (check serial and USB cable)");
        }
    } else {
        // or use any connected module suitable for the demo
        $salt = yFirstAltitude();
        if(is_null($salt)) {
            die("No module connected (check USB cable)");
        } else {
            $serial = $salt->module()->get_serialnumber();
        }
    }
    Print("Module to use: <input name='serial' value='$serial'><br>");

    // Get altitude and temperature as well
    $salt = yFindAltitude("$serial.altitude");
    $temp = yFindTemperature("$serial.temperature");
    $press = yFindPressure("$serial.pressure");

    $hvalue = $salt->get_currentValue();
    $qvalue = $salt->get_qnh();
    $pvalue = $press->get_currentValue();
    $tvalue = $temp->get_currentValue();
    Print("Altitude: $hvalue m (QNH=$qvalue hPa)<br>");
    Print("Temperature: $tvalue deg C <br>");
    Print("Pressure: $pvalue hPa<br>");
    yFreeAPI();

    // trigger auto-refresh after one second
    Print("<script language='javascript1.5' type='text/JavaScript'>\n");
    Print("setTimeout('window.location.reload()',1000);");
    Print("</script>\n");
  ?>
</BODY>
</HTML>
```

9.3. 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
    yDisableExceptions();

    // Setup the API to use the VirtualHub on local machine
    if(yRegisterHub('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
        $module = yFindModule("$serial");
        if (!$module->isOnline()) {
            die("Module not connected (check serial and USB cable)");
        }
    } else {
        // or use any connected module suitable for the demo
        $module = yFirstModule();
        if($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: %s mA<br>', $module->get_usbCurrent());
    printf('logs:<br><pre>%s</pre>', $module->get_lastLogs());
    yFreeAPI();
?>
<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
    yDisableExceptions();

    // Setup the API to use the VirtualHub on local machine
    if(yRegisterHub('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
      $module = yFindModule("$serial");
      if (!$module->isOnline()) {
        die("Module not connected (check serial and USB cable)");
      }
    } else {
      // or use any connected module suitable for the demo
      $module = yFirstModule();
      if($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>");
    yFreeAPI();
  ?>
  <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 alentours 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');
      yRegisterHub("http://127.0.0.1:4444/");
      $module = yFirstModule();
      while (!is_null($module)) {
        printf("%s (%s)<br>", $module->get_serialNumber(),
              $module->get_productName());
        $module=$module->nextModule();
      }
      yFreeAPI();
    ?>
  </TT>
</BODY>
</HTML>

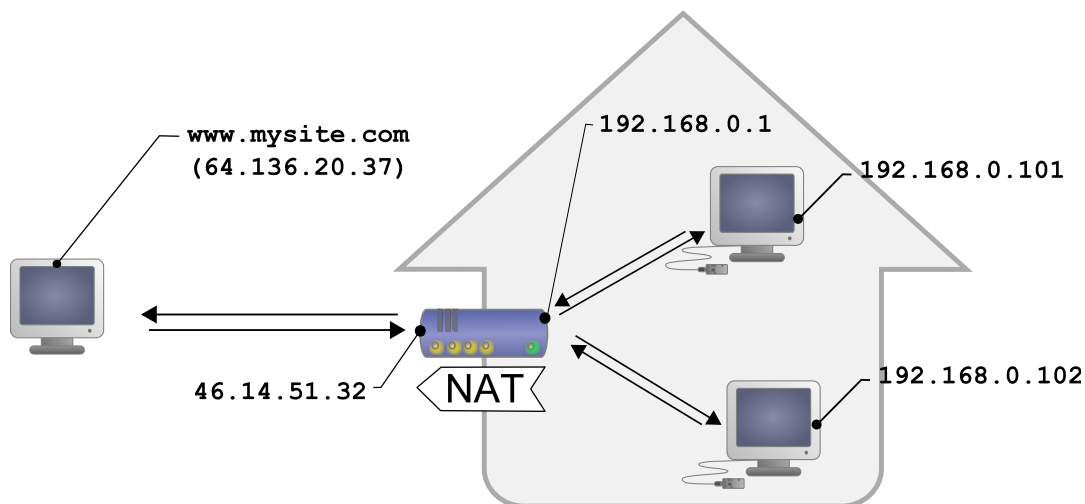
```

9.4. 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 ouvrir un port. L'application typique est le contrôle de modules Yoctopuce situés sur réseau privé depuis un site Web publique.

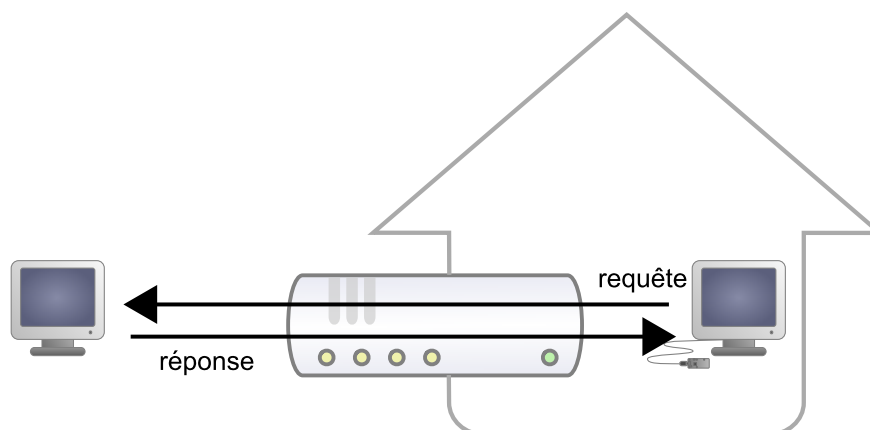
Le filtre NAT, avantages et inconvénients

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.

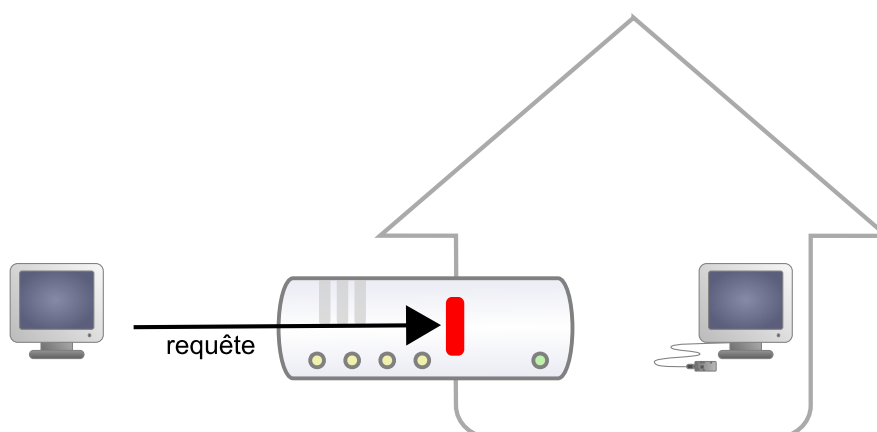


Configuration DSL typique, les machines du LAN sont isolées de l'extérieur par le router DSL

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.



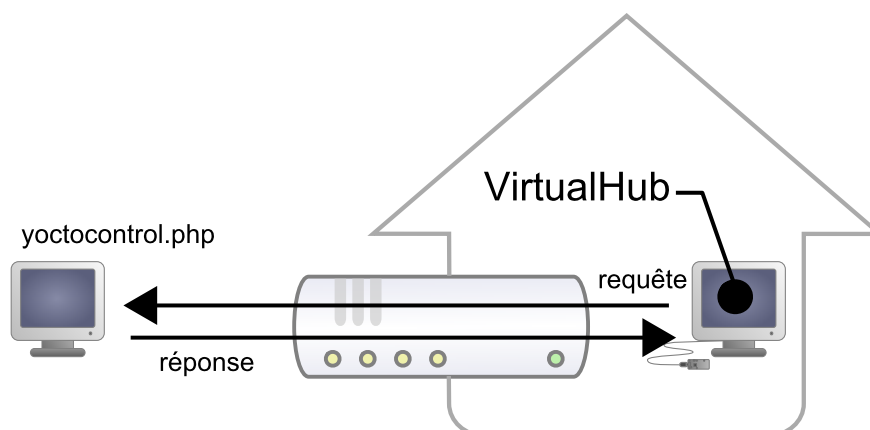
Les réponses aux requêtes venant des machines du LAN sont routées.



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 public 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 résout 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**

Serial	Logical Name	Description	Action
VIRTHUB0-7d1a86fb0		VirtualHub	configure view log file
RELAYHI1-00055		Yocto-PowerRelay	configure view log file beacon
TMPSENS1-05E7F		Yocto-Temperature	configure view log file beacon

Cliquer sur le bouton "configure" de la première ligne

VIRTHUB0-7d1a86fb09

Edit parameters for VIRTHUB0-7d1a86fb09, and click on the **Save** button.

Serial #: VIRTHUB0-7d1a86fb09
 Product name: VirtualHub
 Software version: 10789
 Logical name:

Incoming connections

Authentication to read information from the devices: NO [edit](#)
 Authentication to make changes to the devices: NO [edit](#)

Outgoing callbacks

Callback URL: octoHub [edit](#)
 Delay between callbacks: min: 3 [s] max: 600 [s]

[Save](#) [Cancel](#)

Cliquer sur le bouton "edit" de la section Outgoing callbacks.

Edit callback

This VirtualHub can post the advertised values of all devices on a specific URL on a regular basis. If you wish to use this feature, choose the callback type follow the steps below carefully.

1. Specify the Type of callback you want to use: **Yocto-API callback**

Yoctopuce devices can be controlled through remote PHP scripts. That Yocto-API callback protocol is designed so it can pass through NAT filters without opening ports. See your device user manual, *PHP programming* section for more details.

2. Specify the URL to use for reporting values. *HTTPS protocol is not yet supported.*
 Callback URL:

3. If your callback requires authentication, enter credentials here. Digest authentication is recommended, but Basic authentication works as well.
 Username:
 Password:

4. Setup the desired frequency of notifications:
 No less than seconds between two notification
 But notify after seconds in any case

5. Press on the **Test** button to check your parameters.
 6. When everything works, press on the **OK** button.

[Test](#) [OK](#) [Cancel](#)

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 exécutées 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.

9.5. 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 `Y_STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `Y_CURRENTVALUE_INVALID`, etc. Dans tous les cas, la valeur retournée sera du type attendu, et ne sera pas un pointeur nul qui risquerait de faire crasher votre programme. Au pire, si vous affichez la valeur sans la tester, elle sera hors du cadre attendu pour la valeur retournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI_SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

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

10. Utilisation du Yocto-Altimeter 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` permet de compiler les exemples en ligne de commande, et en pleine connaissance des arguments de compilation et link.

Les bibliothèques Yoctopuce² pour C++ vous sont fournies au format source dans leur intégralité. Une partie de la bibliothèque 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 bibliothèque 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++ 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.

Afin des les garder simples, tous les exemples fournis dans cette documentation sont des applications consoles. Il va de soit que que le fonctionnement des bibliothèques 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 lié avec les bibliothèques Yoctopuce.

10.1. Contrôle de la fonction Altitude

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

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

[...]
String errmsg;
YAltitude *altitude;
```

¹ <http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express>

² www.yoctopuce.com/FR/libraries.php

```
// On récupère l'objet représentant le module (ici connecté en local sur USB)
yRegisterHub("usb", errmsg);
altitude = yFindAltitude("YALTIMK1-123456.altitude");

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.h et yocto_altitude.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_altitude.h` est nécessaire pour gérer les modules contenant un altimètre, comme le Yocto-Altimeter.

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` une explication du problème.

yFindAltitude

La fonction `yFindAltitude`, permet de retrouver un altimètre 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-Altimeter avec le numéros de série `YALTIMK1-123456` que vous auriez appelé *"MonModule"* et dont vous auriez nommé la fonction *altitude* *"MaFonction"*, les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YAltitude *altitude = yFindAltitude("YALTIMK1-123456.altitude");
YAltitude *altitude = yFindAltitude("YALTIMK1-123456.MaFonction");
YAltitude *altitude = yFindAltitude("MonModule.altitude");
YAltitude *altitude = yFindAltitude("MonModule.MaFonction");
YAltitude *altitude = yFindAltitude("MaFonction");
```

`yFindAltitude` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'altimètre.

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `yFindAltitude` permet d'obtenir l'altitude actuelle mesurée par le capteur. La valeur de retour est un nombre flottant en mètres.

yFindTemperature et yFindPressure

Les fonctions `yFindTemperature` et `yFindPressure` permettent de travailler respectivement avec les mesures de température et de pression, et se manipulent exactement comme `yFindAltitude`.

Un exemple réel

Lancez votre environnement C++ et ouvrez le projet exemple correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-Altimeter** 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_altitude.h"
#include "yocto_temperature.h"
#include "yocto_pressure.h"
#include <iostream>
#include <stdlib.h>

using namespace std;

static void usage(void)
{
    cout << "usage: demo <serial_number> " << endl;
    cout << "      demo <logical_name>" << endl;
    cout << "      demo any" << endl;
    u64 now = yGetTickCount();
    while (yGetTickCount() - now < 3000) {
        // wait 3 sec to show the message
    }
    exit(1);
}

int main(int argc, const char * argv[])
{
    string errmsg, target;
    YAltitude *asensor;
    YTemperature *tsensor;
    YPressure *psensor;

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

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

    if(target == "any") {
        asensor = yFirstAltitude();
        tsensor = yFirstTemperature();
        psensor = yFirstPressure();
        if (asensor == NULL || tsensor == NULL || psensor == NULL) {
            cout << "No module connected (check USB cable)" << endl;
            return 1;
        }
    } else {
        asensor = yFindAltitude(target + ".altitude");
        tsensor = yFindTemperature(target + ".temperature");
        psensor = yFindPressure(target + ".pressure");
    }

    while(1) {
        if(!asensor->isOnline()) {
            cout << "Module not connected (check identification and USB cable)";
            break;
        }

        cout << "Current altitude: " << asensor->get_currentValue() << " m (QNH= " <<
            asensor->get_qnh() << " hPa)" << endl;
        cout << "Current temperature: " << tsensor->get_currentValue() << " Â°C" << endl;
        cout << "Current pressure: " << psensor->get_currentValue() << " hPa" << endl;

        cout << " (press Ctrl-C to exit)" << endl;
        ySleep(1000, errmsg);
    };

    yFreeAPI();
    return 0;
}
```

10.2. Contrôle de la partie module

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

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

#include "yocto_api.h"

using namespace std;

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

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

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

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

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

    if (module->isOnline()) {
        if (argc > 2) {
            if (string(argv[2]) == "ON")
                module->set_beacon(Y_BEACON_ON);
            else
                module->set_beacon(Y_BEACON_OFF);
        }
        cout << "serial:          " << module->get_serialNumber() << endl;
        cout << "logical name: " << module->get_logicalName() << endl;
        cout << "luminosity:   " << module->get_luminosity() << endl;
        cout << "beacon:      ";
        if (module->get_beacon() == Y_BEACON_ON)
            cout << "ON" << endl;
        else
            cout << "OFF" << endl;
        cout << "upTime:      " << module->get_upTime() / 1000 << " sec" << endl;
        cout << "USB current: " << module->get_usbCurrent() << " mA" << endl;
        cout << "Logs:" << endl << module->get_lastLogs() << endl;
    } else {
        cout << argv[1] << " not connected (check identification and USB cable)"
            << endl;
    }
    yFreeAPI();
    return 0;
}
```

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

Modifications des réglages du module

Lorsque que vous souhaitez modifier les réglages d'un module, il suffit d'appeler la fonction `set_xxx()` correspondante, cependant cette modification n'a lieu que dans la mémoire vive du module: si le module redémarre, les modifications seront perdues. Pour qu'elle soient mémorisées de manière persistante, il est nécessaire de demander au module de sauvegarder sa configuration courante dans sa mémoire non volatile. Pour cela il faut utiliser la méthode `saveToFlash()`.

Inversement il est possible de forcer le module à oublier ses réglages courants en utilisant la méthode `revertFromFlash()`. Ce petit exemple ci-dessous vous permet changer le nom logique d'un module.

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

#include "yocto_api.h"

using namespace std;

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

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

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

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

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

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

```

10.3. 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 `Y_STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `Y_CURRENTVALUE_INVALID`, etc. Dans tous les cas, la valeur retournée sera du type attendu, et ne sera pas un pointeur nul qui risquerait de faire crasher votre programme. Au pire, si vous affichez la valeur sans la tester, elle sera hors du cadre attendu pour la valeur retournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI_SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

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

10.4. Intégration de la librairie Yoctopuce en C++

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

Intégration au format source

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

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

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

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

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

Intégration en librairie statique

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

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

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

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

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

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

Intégration en librairie dynamique

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

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

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

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

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

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

11. Utilisation du Yocto-Altimeter 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 librairie 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 pouvoir 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 Yoctopuce¹ 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 détaillé² avec des séquences vidéo montrant comment intégrer les fichiers de la librairie à vos projets.

11.1. Contrôle de la fonction Altitude

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

```
#import <Foundation/Foundation.h>
#import "yocto_api.h"
#import "yocto_altitude.h"
#import "yocto_temperature.h"
#import "yocto_pressure.h"

static void usage(void)
{
    NSLog(@"usage: demo <serial_number> ");
    NSLog(@"      demo <logical_name>");
    NSLog(@"      demo any          (use any discovered device)");
}
```

¹ www.yoctopuce.com/FR/libraries.php

² www.yoctopuce.com/FR/article/nouvelle-librairie-objective-c-pour-mac-os-x

```

    exit(1);
}

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

    if (argc < 2) {
        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]];
        YAltitude *asensor;
        YTemperature *tsensor;
        YPressure *psensor;

        if([target isEqualToString:@"any"]) {
            asensor = [YAltitude FirstAltitude];
            tsensor = [YTemperature FirstTemperature];
            psensor = [YPressure FirstPressure];
            if (asensor == NULL || tsensor == NULL || psensor == NULL) {
                NSLog(@"No module connected (check USB cable)");
                return 1;
            }
        } else {
            asensor = [YAltitude FindAltitude:[target stringByAppendingString:@"altitude"]];
            tsensor = [YTemperature FindTemperature:[target
stringByAppendingString:@"temperature"]];
            psensor = [YPressure FindPressure:[target stringByAppendingString:@"pressure"]];
        }
        while(1) {
            if(![asensor isOnline]) {
                NSLog(@"Module not connected (check identification and USB cable)\n");
                break;
            }
            NSLog(@"Current altitude:      %f m (QNH=%f)\n", [asensor get_currentValue], [asensor
get_qnh]);
            NSLog(@"Current pressure:      %f hPa\n", [psensor get_currentValue]);
            NSLog(@"Current temperature: %f C\n", [tsensor get_currentValue]);
            NSLog(@" (press Ctrl-C to exit)\n");
            [YAPI Sleep:1000:NULL];
        }
        [YAPI FreeAPI];
    }

    return 0;
}

```

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

yocto_api.h et yocto_altitude.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_altitude.h` est nécessaire pour gérer les modules contenant un altimètre, comme le Yocto-Altimeter.

[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.

[Altitude FindAltitude]

La fonction [Altitude FindAltitude], permet de retrouver un altimetre 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-Altimeter avec le numéros de série *YALTIMK1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *altitude* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YAltitude *altitude = [YAltitude FindAltitude:@"YALTIMK1-123456.altitude"];
YAltitude *altitude = [YAltitude FindAltitude:@"YALTIMK1-123456.MaFonction"];
YAltitude *altitude = [YAltitude FindAltitude:@"MonModule.altitude"];
YAltitude *altitude = [YAltitude FindAltitude:@"MonModule.MaFonction"];
YAltitude *altitude = [YAltitude FindAltitude:@"MaFonction"];
```

[YAltitude FindAltitude] renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'altimètre.

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `YAltitude.FindAltitude` permet d'obtenir l'altitude actuelle mesurée par le capteur. La valeur de retour est un nombre flottant en mètres.

YTemperature.FindTemperature et YPressure.FindPressure

Les fonctions `YTemperature.FindTemperature` et `YPressure.FindPressure` permettent de travailler respectivement avec les mesures de température et de pression, et se manipulent exactement comme `YAltitude.FindAltitude`.

11.2. 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 au 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 alentours 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;
}

```

11.3. 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 `Y_STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `Y_CURRENTVALUE_INVALID`, etc. Dans tous les cas, la valeur retournée sera du type attendu, et ne sera pas un pointeur nul qui risquerait de faire crasher votre programme. Au pire, si vous affichez la valeur sans la tester, elle sera hors du cadre attendu pour la valeur retournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI_SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

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

12. Utilisation du Yocto-Altimeter en VisualBasic .NET

VisualBasic a longtemps été la porte d'entrée privilégiée vers le monde Microsoft. Nous nous devons 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.

12.1. Installation

Téléchargez la librairie Yoctopuce pour Visual Basic depuis le site web de Yoctopuce². Il n'y a pas de programme d'installation, copiez simplement le 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.

12.2. 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 modules³. 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 **Élément 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.

12.3. Contrôle de la fonction Altitude

Il suffit de quelques lignes de code pour piloter un Yocto-Altimeter. Voici le squelette d'un fragment de code VisualBasic .NET qui utilise la fonction Altitude.

```
[...]
Dim errmsg As String
Dim altitude As YAltitude

REM On récupère l'objet représentant le module (ici connecté en local sur USB)
yRegisterHub("usb", errmsg)
altitude = yFindAltitude("YALTIMK1-123456.altitude")

REM Pour gérer le hot-plug, on vérifie que le module est là
If (altitude.IsOnline()) Then
    REM Utiliser altitude.GetCurrentValue(), ...
End If
```

Voyons maintenant en détail ce que font ces quelques lignes.

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.

yFindAltitude

La fonction `yFindAltitude`, permet de retrouver un altimetre 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-Altimeter avec le numéros de série `YALTIMK1-123456` que vous auriez appelé `"MonModule"` et dont vous auriez nommé la fonction `altitude` `"MaFonction"`, les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
altitude = yFindAltitude("YALTIMK1-123456.altitude")
altitude = yFindAltitude("YALTIMK1-123456.MaFonction")
altitude = yFindAltitude("MonModule.altitude")
altitude = yFindAltitude("MonModule.MaFonction")
altitude = yFindAltitude("MaFonction")
```

`yFindAltitude` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'altimètre.

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

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `yFindAltitude` permet d'obtenir l'altitude actuelle mesurée par le capteur. La valeur de retour est un nombre flottant en mètres.

yFindTemperature et yFindPressure

Les fonctions `yFindTemperature` et `yFindPressure` permettent de travailler respectivement avec les mesures de température et de pression, et se manipulent exactement comme `yFindAltitude`.

Un exemple réel

Lancez Microsoft VisualBasic et ouvrez le projet exemple correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-Altimeter** 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>")
        Console.WriteLine(execname + " <logical_name>")
        Console.WriteLine(execname + " any ")
        System.Threading.Thread.Sleep(2500)
    End Sub

    Sub Main()
        Dim argv() As String = System.Environment.GetCommandLineArgs()
        Dim errmsg As String = ""
        Dim target As String
        Dim asensor As YAltitude
        Dim tsensor As YTemperature
        Dim psensor As YPressure

        If argv.Length < 2 Then Usage()

        target = argv(1)

        REM Setup the API to use local USB devices
        If (yRegisterHub("usb", errmsg) <> YAPI_SUCCESS) Then
            Console.WriteLine("RegisterHub error: " + errmsg)
        End If

        If target = "any" Then
            asensor = yFirstAltitude()
            tsensor = yFirstTemperature()
            psensor = yFirstPressure()

            If asensor Is Nothing Or tsensor Is Nothing Or psensor Is Nothing Then
                Console.WriteLine("No module connected (check USB cable) ")
            End If
        Else
            asensor = yFindAltitude(target + ".altitude")
            tsensor = yFindTemperature(target + ".temperature")
            psensor = yFindPressure(target + ".pressure")
        End If

        While (True)
            If Not (asensor.isOnline()) Then
                Console.WriteLine("Module not connected (check identification and USB cable)")
            End If
        End While
    End Sub
End Module
```

```

End
End If
Console.WriteLine("Current altitude:    " + Str(asensor.get_currentValue()) _
                  + " m " + "(QNH=" + Str(asensor.get_qnh()) + "hPa)")
Console.WriteLine("Current pressure:    " + Str(psensor.get_currentValue()) _
                  + " hPa")
Console.WriteLine("Current temperature: " + Str(tsensor.get_currentValue()) _
                  + " Â°C")
Console.WriteLine("  (press Ctrl-C to exit)")
ySleep(1000, errmsg)
End While
yFreeAPI()
End Sub

End Module

```

12.4. 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 (yRegisterHub("usb", errmsg) <> YAPI_SUCCESS) Then
        Console.WriteLine("RegisterHub error:" + errmsg)
    End
End If

If argv.Length < 2 Then usage()

m = yFindModule(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.WriteLine("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
yFreeAPI()
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 au 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 yRegisterHub("usb", errmsg) <> YAPI_SUCCESS Then
        Console.WriteLine("RegisterHub error: " + errmsg)
    End
    End If

    m = yFindModule(argv(1)) REM use serial or logical name
    If m.isOnline() Then
        newname = argv(2)
        If (Not yCheckLogicalName(newname)) Then
            Console.WriteLine("Invalid name (" + newname + ")")
        End
        End If
        m.set_logicalName(newname)
        m.saveToFlash() REM do not forget this
        Console.WriteLine("Module: serial= " + m.get_serialNumber())
        Console.WriteLine(" / name= " + m.get_logicalName())
    Else
        Console.WriteLine("not connected (check identification and USB cable)")
    End If
    yFreeAPI()

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 alentours 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 yRegisterHub("usb", errmsg) <> YAPI_SUCCESS Then
            Console.WriteLine("RegisterHub error: " + errmsg)
        End If

        Console.WriteLine("Device list")
        M = yFirstModule()
        While M IsNot Nothing
            Console.WriteLine(M.get_serialNumber() + " (" + M.get_productName() + ")")
            M = M.nextModule()
        End While
        yFreeAPI()
    End Sub

End Module
```

12.5. 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 `Y_STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `Y_CURRENTVALUE_INVALID`, etc. Dans tous les cas, la valeur retournée sera du type attendu, et

ne sera pas un pointeur nul qui risquerait de faire crasher votre programme. Au pire, si vous affichez la valeur sans la tester, elle sera hors du cadre attendu pour la valeur retournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI_SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

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

13. Utilisation du Yocto-Altimeter en C#

C# (prononcez C-Sharp) est un langage orienté objet promu par Microsoft qui n'est pas sans rappeler 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.

13.1. Installation

Téléchargez la librairie Yoctopuce pour Visual C# depuis le site web de Yoctopuce². Il n'y a pas de programme d'installation, copiez simplement le 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.

13.2. 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 modules³. 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 **Élément 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 fonctionnements des librairies est strictement identiques si vous les intégrez dans une application dotée d'une interface graphique.

13.3. Contrôle de la fonction Altitude

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

```
[...]
string errormsg = "";
YAltitude altitude;

// On récupère l'objet représentant le module (ici connecté en local sur USB)
YAPI.RegisterHub("usb", errormsg);
altitude = YAltitude.FindAltitude("YALTIMK1-123456.altitude");

// Pour gérer le hot-plug, on vérifie que le module est là
if (altitude.isOnline())
{ // Utiliser altitude.get_currentValue(): ...
}
```

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 `errormsg` une explication du problème.

YAltitude.FindAltitude

La fonction `YAltitude.FindAltitude`, permet de retrouver un altimètre 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-Altimeter avec le numéros de série `YALTIMK1-123456` que vous auriez appelé `"MonModule"` et dont vous auriez nommé la fonction `altitude` `"MaFonction"`, les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
altitude = YAltitude.FindAltitude("YALTIMK1-123456.altitude");
altitude = YAltitude.FindAltitude("YALTIMK1-123456.MaFonction");
altitude = YAltitude.FindAltitude("MonModule.altitude");
altitude = YAltitude.FindAltitude("MonModule.MaFonction");
altitude = YAltitude.FindAltitude("MaFonction");
```

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

`YAltitude.FindAltitude` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'altimètre.

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `YAltitude.FindAltitude` permet d'obtenir l'altitude actuelle mesurée par le capteur. La valeur de retour est un nombre flottant en mètres.

YTemperature.FindTemperature et YPressure.FindPressure

Les fonctions `YTemperature.FindTemperature` et `YPressure.FindPressure` permettent de travailler respectivement avec les mesures de température et de pression, et se manipulent exactement comme `YAltitude.FindAltitude`.

Un exemple réel

Lancez Visual C# et ouvrez le projet exemple correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-Altimeter** 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(execname + " <serial_number>");
            Console.WriteLine(execname + " <logical_name>");
            Console.WriteLine(execname + " any ");
            System.Threading.Thread.Sleep(2500);
            Environment.Exit(0);
        }

        static void Main(string[] args)
        {
            string errormsg = "";
            string target;

            YAltitude asensor;
            YTemperature tsensor;
            YPressure psensor;

            if (args.Length < 1) usage();
            target = args[0].ToUpper();

            // Setup the API to use local USB devices
            if (YAPI.RegisterHub("usb", ref errormsg) != YAPI.SUCCESS) {
                Console.WriteLine("RegisterHub error: " + errormsg);
                Environment.Exit(0);
            }

            if (target == "ANY") {
                asensor = YAltitude.FirstAltitude();
                tsensor = YTemperature.FirstTemperature();
                psensor = YPressure.FirstPressure();

                if ((asensor == null) || (tsensor == null) || (psensor == null)) {
                    Console.WriteLine("No module connected (check USB cable) ");
                }
            }
        }
    }
}
```

```

        Environment.Exit(0);
    }
} else {
    asensor = YAltitude.FindAltitude(target + ".altitude");
    tsensor = YTemperature.FindTemperature(target + ".temperature");
    psensor = YPressure.FindPressure(target + ".pressure");
}

if (!asensor.isOnline()) {
    Console.WriteLine("Module not connected");
    Console.WriteLine("check identification and USB cable");
    Environment.Exit(0);
}

while (asensor.isOnline()) {
    Console.WriteLine("Current altitude:    " + asensor.get_currentValue().ToString()
        + " m "
        + "(QNH=" + asensor.get_qnh().ToString() + " hPa)");
    Console.WriteLine("Current temperature: " + tsensor.get_currentValue().ToString()
        + " Â°C");
    Console.WriteLine("Current pressure:    " + psensor.get_currentValue().ToString()
        + " hPa");
    Console.WriteLine(" (press Ctrl-C to exit)");

    YAPI.Sleep(1000, ref errormsg);
}
YAPI.FreeAPI();
}
}
}

```

13.4. 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 errormsg = "";

            if (YAPI.RegisterHub("usb", ref errormsg) != YAPI.SUCCESS) {
                Console.WriteLine("RegisterHub error: " + errormsg);
                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:\x\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 alentours de 100000 cycles. Pour résumer vous ne pouvez employer la fonction `YModule.saveToFlash()` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

Enumeration des modules

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

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

```

13.5. 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 `Y_STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `Y_CURRENTVALUE_INVALID`, etc. Dans tous les cas, la valeur retournée sera du type attendu, et ne sera pas un pointeur nul qui risquerait de faire crasher votre programme. Au pire, si vous affichez la valeur sans la tester, elle sera hors du cadre attendu pour la valeur retournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI_SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

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

14. Utilisation du Yocto-Altimeter avec Universal Windows Platform

Universal Windows Platform, abrégé UWP, n'est pas un langage à proprement parler mais une plate-forme logicielle créée par Microsoft. Cette plateforme permet d'exécuter un nouveau type d'applications : les applications universelles Windows. Ces applications peuvent fonctionner sur toutes les machines qui fonctionnent sous Windows 10. Cela comprend les PCs, les tablettes, les smartphones, la Xbox One, mais aussi Windows IoT Core.

La bibliothèque Yoctopuce UWP permet d'utiliser les modules Yoctopuce dans une application universelle Windows et est entièrement écrite C#. Elle peut être ajoutée à un projet Visual Studio 2017¹.

14.1. Fonctions bloquantes et fonctions asynchrones

La bibliothèque 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 bibliothèque UWP peut être utilisée 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 bibliothèque 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-clés `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-clé `async`
- le mot-clé `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 utiliser 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 asynchrones**, 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.

14.2. Installation

Téléchargez la librairie Yoctopuce pour \$LANG\$ depuis le site web de Yoctopuce ². Il n'y a pas de programme d'installation, copiez simplement le 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.

14.3. 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 d'accéder 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 `uap:DeviceCapability` dans le `uap:Capabilities`. Ce `uap:DeviceCapability` doit avoir un attribut "Name" qui vaut "humaninterfacedevice".

A l'intérieur de ce `uap:DeviceCapability`, il faut déclarer tous les modules qui peuvent être utilisés. Concrètement, pour chaque module, il faut ajouter un `uap: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-Altimeter voici ce qu'il faut ajouter dans le n u d "Capabilities":

```
<DeviceCapability Name="humaninterfacedevice">
  <!-- Yocto-Altimeter -->
  <Device Id="vidpid:24e0 003F">
    <Function Type="usage:ff00 0001" />
  </Device>
</DeviceCapability>
```

Malheureusement, il n'est pas possible d'écrire une règle qui autorise tous les modules Yoctopuce, par conséquent il faut impérativement ajouter chaque module que l'on désire utiliser.

14.4. Contrôle de la fonction Altitude

Il suffit de quelques lignes de code pour piloter un Yocto-Altimeter. Voici le squelette d'un fragment de code c# qui utilise la fonction Altitude.

```
[...]

await YAPI.RegisterHub("usb");
altitude = YAltitude.FindAltitude("YALTIMK1-123456.altitude");

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

[...]
```

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.

YAltitude.FindAltitude

La fonction `YAltitude.FindAltitude` permet de retrouver un altimètre 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-Altimeter avec le numéro de série *YALTIMK1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *altitude* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
altitude = YAltitude.FindAltitude("YALTIMK1-123456.altitude");
altitude = YAltitude.FindAltitude("YALTIMK1-123456.MaFonction");
altitude = YAltitude.FindAltitude("MonModule.altitude");
altitude = YAltitude.FindAltitude("MonModule.MaFonction");
altitude = YAltitude.FindAltitude("MaFonction");
```

`YAltitude.FindAltitude` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'altimètre.

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `YAltitude.FindAltitude` permet d'obtenir l'altitude actuelle mesurée par le capteur. La valeur de retour est un nombre flottant en mètres.

YTemperature.FindTemperature et YPressure.FindPressure

Les fonctions `YTemperature.FindTemperature` et `YPressure.FindPressure` permettent de travailler respectivement avec les mesures de température et de pression, et se manipulent exactement comme `YAltitude.FindAltitude`.

14.5. Un exemple concret

Lancez Visual Studio et ouvrez le projet correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-Altimeter** de la librairie Yoctopuce.

Le projet 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 différents 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 override async Task<int> Run()
        {
            try {
                await YAPI.RegisterHub(HubURL);
                YAltitude asensor;
                YTemperature tsensor;
                YPressure psensor;

                if (Target.ToLower() == "any") {
                    asensor = YAltitude.FirstAltitude();
                    tsensor = YTemperature.FirstTemperature();
                    psensor = YPressure.FirstPressure();

                    if ((asensor == null) || (tsensor == null) || (psensor == null)) {
                        WriteLine("No module connected (check USB cable) ");
                        return -1;
                    }
                } else {
                    asensor = YAltitude.FindAltitude(Target + ".altitude");
                    tsensor = YTemperature.FindTemperature(Target + ".temperature");
                    psensor = YPressure.FindPressure(Target + ".pressure");
                }
                while (await asensor.isOnline()) {
                    WriteLine("Altitude: " + await asensor.get_currentValue() + " m " +
                        "(QNH=" +
                            await asensor.get_qnh() + " hPa)");
                    WriteLine("Temperature: " + await tsensor.get_currentValue() + " Â°C");
                    WriteLine("Pressure: " + await psensor.get_currentValue() + " hPa");
                    await YAPI.Sleep(1000);
                }
                WriteLine("Module not connected (check identification and USB cable)");
            } catch (YAPI_Exception ex) {
```

```

        WriteLine("error: " + ex.Message);
    }

    YAPI.FreeAPI();
    return 0;
}
}
}

```

14.6. 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 alentours de 100000 cycles. Pour résumer vous ne pouvez employer la fonction `YModule.saveToFlash()` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

Enumeration des modules

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

```

14.7. 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 dès 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);
}

```


15. Utilisation du Yocto-Altimeter 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 payant¹.

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 versions de Delphi².

Afin de 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 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.

15.1. Préparation

Connectez-vous sur le site de Yoctopuce et téléchargez la librairie Yoctopuce pour Delphi³. 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 Delphi⁴.

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.

15.2. Contrôle de la fonction Altitude

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.

```
program helloworld;  
{ $APPTYPE CONSOLE }  
uses
```

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'environnement**

```

SysUtils,
Windows,
yocto_api,
yocto_altitude,
yocto_temperature,
yocto_pressure;

Procedure Usage();
var
  exe : string;

begin
  exe:= ExtractFileName(paramstr(0));
  WriteLn(exe+' <serial_number>');
  WriteLn(exe+' <logical_name>');
  WriteLn(exe+' any');
  halt;
End;

var
  asensor      : TYAltitude;
  tsensor      : TYTemperature;
  psensor      : TYPressure;
  m            : TYModule;
  errmsg,serial : string;
  done         : boolean;

begin
  if (paramcount<1) then usage();

  // Setup the API to use local USB devices
  if yRegisterHub('usb', errmsg)<>YAPI_SUCCESS then
  begin
    Write('RegisterHub error: '+errmsg);
    exit;
  end;

  if paramstr(1)='any' then
  begin
    // lets try to find the first available altitude sensor
    asensor := yFirstAltitude();
    if asensor=nil then
    begin
      writeln('No module connected (check USB cable)');
      halt;
    end
  end
  else
    // or the one specified on command line
    asensor:= YFindAltitude(paramstr(1)+'.altitude');

  // make sure it is online
  if not asensor.isOnline() then
  begin
    writeln('No module connected (check USB cable)');
    halt;
  end;

  // lets find the parent module so we can get the other sensors
  m := asensor.get_module();
  serial := m.get_serialNumber();

  // retrieve all sensor present on the yocto-meteo
  asensor := yFindAltitude(serial+'.altitude');
  tsensor := yFindTemperature(serial+'.temperature');
  psensor := yFindPressure(serial+'.pressure');

  // let's poll
  done := false;
  repeat
    if (asensor.isOnline()) then
    begin
      Writeln('Curr altitude:      '+FloatToStr(asensor.get_currentValue())+' m ' +
        '(QNH='+FloatToStr(asensor.get_qnh())+' hPa)');
      Writeln('Curr temperature: '+FloatToStr(tsensor.get_currentValue())+' deg C');
      Writeln('Curr pressure:      '+FloatToStr(psensor.get_currentValue())+' hPa');
      Writeln('      (press Ctrl-C to exit)');
      Writeln('');
    end
  end

```

```

        Sleep(1000);
    end
else
begin
    Writeln('Module not connected (check identification and USB cable)');
    done := true;
end;
until done;
yFreeAPI();
end.

```

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

yocto_api et yocto_altitude

Ces deux unités permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api` doit toujours être utilisé, `yocto_altitude` est nécessaire pour gérer les modules contenant un altimètre, comme le Yocto-Altimeter.

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` une explication du problème.

yFindAltitude

La fonction `yFindAltitude`, permet de retrouver un altimètre 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-Altimeter avec le numéro de série `YALTIMK1-123456` que vous auriez appelé *"MonModule"* et dont vous auriez nommé la fonction *altitude* *"MaFonction"*, les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```

altitude := yFindAltitude("YALTIMK1-123456.altitude");
altitude := yFindAltitude("YALTIMK1-123456.MaFonction");
altitude := yFindAltitude("MonModule.altitude");
altitude := yFindAltitude("MonModule.MaFonction");
altitude := yFindAltitude("MaFonction");

```

`yFindAltitude` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'altimètre.

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `yFindAltitude` permet d'obtenir l'altitude actuelle mesurée par le capteur. La valeur de retour est un nombre flottant en mètres.

yFindTemperature et yFindPressure

Les fonctions `yFindTemperature` et `yFindPressure` permettent de travailler respectivement avec les mesures de température et de pression, et se manipulent exactement comme `yFindAltitude`.

15.3. 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 = 'YALTIMK1-123456'; // 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 = 'YALTIMK1-123456'; // 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 alentours 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.

```

15.4. 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 `Y_STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `Y_CURRENTVALUE_INVALID`, etc. Dans tous les cas, la valeur retournée sera du type attendu, et ne sera pas un pointeur nul qui risquerait de faire crasher votre programme. Au pire, si vous affichez la valeur sans la tester, elle sera hors du cadre attendu pour la valeur retournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI_SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

Quand vous travaillez sans les exceptions, il est possible d'obtenir un code d'erreur et un message expliquant l'origine de l'erreur en le demandant à l'objet qui a retourné une erreur à l'aide des

méthodes `errType()` et `errMessage()`. Ce sont les même informations qui auraient été associées à l'exception si elles avaient été actives.

16. Utilisation du Yocto-Altimeter 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 langage 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, Mac 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.

16.1. Fichiers sources

Les classes de la librairie Yoctopuce² 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 de le configurer de manière à ce qu'il retrouve automatiquement les fichiers sources de l'API.

16.2. 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 le fonctionnement des librairies est strictement identiques si vous les intégrez dans une application dotée d'une interface graphique.

¹ <http://www.python.org/download/>

² www.yoctopuce.com/FR/libraries.php

16.3. Contrôle de la fonction Altitude

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

```
[...]

errmsg=YRefParam()
#On récupère l'objet représentant le module (ici connecté en local sur USB)
YAPI.RegisterHub("usb",errmsg)
altitude = YAltitude.FindAltitude("YALTIMK1-123456.altitude")

#Pour gérer le hot-plug, on vérifie que le module est là
if altitude.isOnline():
    #Use altitude.get_currentValue()
    ...

[...]
```

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.

YAltitude.FindAltitude

La fonction `YAltitude.FindAltitude`, permet de retrouver un altimètre 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-Altimeter avec le numéros de série `YALTIMK1-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *altitude* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
altitude = YAltitude.FindAltitude("YALTIMK1-123456.altitude")
altitude = YAltitude.FindAltitude("YALTIMK1-123456.MaFonction")
altitude = YAltitude.FindAltitude("MonModule.altitude")
altitude = YAltitude.FindAltitude("MonModule.MaFonction")
altitude = YAltitude.FindAltitude("MaFonction")
```

`YAltitude.FindAltitude` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'altimètre.

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `YAltitude.FindAltitude` permet d'obtenir l'altitude actuelle mesurée par le capteur. La valeur de retour est un nombre flottant en mètres.

YTemperature.FindTemperature et YPressure.FindPressure

Les fonctions `YTemperature.FindTemperature` et `YPressure.FindPressure` permettent de travailler respectivement avec les mesures de température et de pression, et se manipulent exactement comme `YAltitude.FindAltitude`.

Un exemple réel

Lancez votre interpréteur Python et ouvrez le script correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-Altimeter** 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_altitude import *
from yocto_temperature import *
from yocto_pressure import *

def usage():
    scriptname = os.path.basename(sys.argv[0])
    print("Usage:")
    print(scriptname + ' <serial_number>')
    print(scriptname + ' <logical_name>')
    print(scriptname + ' any ')
    sys.exit()

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

errmsg = YRefParam()

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

target = sys.argv[1]

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

if target == 'any':
    # retrieve any altitude sensor
    sensor = YAltitude.FirstAltitude()
    if sensor is None:
        die('No module connected')
    m = sensor.get_module()
    target = m.get_serialNumber()

else:
    m = YModule.FindModule(target)

if not m.isOnline():
    die('device not connected')

altSensor = YAltitude.FindAltitude(target + '.altitude')
pressSensor = YPressure.FindPressure(target + '.pressure')
tempSensor = YTemperature.FindTemperature(target + '.temperature')

while altSensor.isOnline():
    print("%4.1f" % altSensor.get_currentValue() + "m (QNH=" \
        + "%4.1f" % altSensor.get_qnh() + "hPa) " \
        + "%4.1f" % pressSensor.get_currentValue() + "hPa " \
        + "%2.0f" % tempSensor.get_currentValue() + "deg C " \
        + "(Ctrl-c to stop) ")
    YAPI.Sleep(1000)
YAPI.FreeAPI()
```

16.4. 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 alentours de 100000 cycles. Pour résumer vous ne pouvez employer la fonction `YModule.saveToFlash()` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

Enumeration des modules

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

```

#!/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()

```

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 `Y_STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `Y_CURRENTVALUE_INVALID`, etc. Dans tous les cas, la valeur retournée sera du type attendu, et ne sera pas un pointeur nul qui risquerait de faire crasher votre programme. Au pire, si vous affichez la valeur sans la tester, elle sera hors du cadre attendu pour la valeur retournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI_SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

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

17. Utilisation du Yocto-Altimeter 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.

17.1. Préparation

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

- La librairie de programmation pour Java¹
- Le programme VirtualHub² 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.

17.2. Contrôle de la fonction Altitude

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

```
[...]

// On récupère l'objet représentant le module (ici connecté en local sur USB)
YAPI.RegisterHub("127.0.0.1");
altitude = YAltitude.FindAltitude("YALTIMK1-123456.altitude");

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

¹ www.yoctopuce.com/FR/libraries.php

² www.yoctopuce.com/FR/virtualhub.php

```
[...]
}
[...]
```

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.

YAltitude.FindAltitude

La fonction `YAltitude.FindAltitude`, permet de retrouver un altimètre 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-Altimeter avec le numéro de série `YALTIMK1-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *altitude* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
altitude = YAltitude.FindAltitude("YALTIMK1-123456.altitude")
altitude = YAltitude.FindAltitude("YALTIMK1-123456.MaFonction")
altitude = YAltitude.FindAltitude("MonModule.altitude")
altitude = YAltitude.FindAltitude("MonModule.MaFonction")
altitude = YAltitude.FindAltitude("MaFonction")
```

`YAltitude.FindAltitude` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'altimètre.

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `YAltitude.FindAltitude` permet d'obtenir l'altitude actuelle mesurée par le capteur. La valeur de retour est un nombre flottant en mètres.

YTemperature.FindTemperature et YPressure.FindPressure

Les fonctions `YTemperature.FindTemperature` et `YPressure.FindPressure` permettent de travailler respectivement avec les mesures de température et de pression, et se manipulent exactement comme `YAltitude.FindAltitude`.

Un exemple réel

Lancez votre environnement java et ouvrez le projet correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-Altimeter** 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);
    }

    YAltitude asensor;
    YTemperature tsensor;
    YPressure psensor;

    if (args.length == 0) {
        asensor = YAltitude.FirstAltitude();
        tsensor = YTemperature.FirstTemperature();
        psensor = YPressure.FirstPressure();
        if (asensor == null || tsensor == null || psensor == null) {
            System.out.println("No module connected (check USB cable)");
            System.exit(1);
        }
    } else {
        asensor = YAltitude.FindAltitude(args[0] + ".altitude");
        tsensor = YTemperature.FindTemperature(args[0] + ".temperature");
        psensor = YPressure.FindPressure(args[0] + ".pressure");
    }

    while (true) {
        try {
            System.out.println("Current altitude: " + asensor.get_currentValue() + " m
"
                                + "(QNH=" + asensor.get_qnh() + " hpa)");
            System.out.println("Current temperature: " + tsensor.get_currentValue() + "
Â°C");
            System.out.println("Current pressure: " + psensor.get_currentValue() + "
hPa");
            System.out.println("  (press Ctrl-C to exit)");
            YAPI.Sleep(1000);
        } catch (YAPI_Exception ex) {
            System.out.println("Module not connected (check identification and USB
cable)");
            break;
        }
        YAPI.FreeAPI();
    }
}

```

17.3. 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 alentours de 100000 cycles. Pour résumer vous ne pouvez employer la fonction `YModule.saveToFlash()` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

Enumeration des modules

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

```

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();
    }
}

```

17.4. 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 dès que vous débrancherez un module.

18. Utilisation du Yocto-Altimeter 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.

18.1. 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.

18.2. Préparation

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

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

18.3. Compatibilité

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

¹ www.yoctopuce.com/FR/libraries.php

Android 4.x

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

Support USB *host*

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

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

Matériel supporté

La librairie est testée et validée sur les machines suivantes:

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

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

18.4. Activer le port USB sous Android

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

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

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

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

```
</manifest>
```

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

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

Lancement automatique

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

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

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

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

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

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

18.5. Contrôle de la fonction Altitude

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

```
[...]

// On récupère l'objet représentant le module (ici connecté en local sur USB)
YAPI.EnableUSBHost(this);
YAPI.RegisterHub("usb");
altitude = YAltitude.FindAltitude("YALTIMK1-123456.altitude");

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

[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

YAPI.EnableUSBHost

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

YAPI.RegisterHub

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

YAltitude.FindAltitude

La fonction `YAltitude.FindAltitude` permet de retrouver un altimètre 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-Altimeter avec le numéro de série `YALTIMK1-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *altitude* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
altitude = YAltitude.FindAltitude("YALTIMK1-123456.altitude")
altitude = YAltitude.FindAltitude("YALTIMK1-123456.MaFonction")
altitude = YAltitude.FindAltitude("MonModule.altitude")
altitude = YAltitude.FindAltitude("MonModule.MaFonction")
altitude = YAltitude.FindAltitude("MaFonction")
```

`YAltitude.FindAltitude` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'altimètre.

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `YAltitude.FindAltitude` permet d'obtenir l'altitude actuelle mesurée par le capteur. La valeur de retour est un nombre flottant en mètres.

YTemperature.FindTemperature et YPressure.FindPressure

Les fonctions `YTemperature.FindTemperature` et `YPressure.FindPressure` permettent de travailler respectivement avec les mesures de température et de pression, et se manipulent exactement comme `YAltitude.FindAltitude`.

Un exemple réel

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

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

```
package com.yoctopuce.doc_exemples;

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

import com.yoctopuce.YoctoAPI.YAPI;
import com.yoctopuce.YoctoAPI.YAPI_Exception;
import com.yoctopuce.YoctoAPI.YAltitude;
import com.yoctopuce.YoctoAPI.YModule;
import com.yoctopuce.YoctoAPI.YPressure;
import com.yoctopuce.YoctoAPI.YTemperature;

public class GettingStarted_Yocto_Altimeter extends Activity implements
OnItemClickListener
{
    private ArrayAdapter<String> aa;
    private String serial = "";
    private Handler handler = null;

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

    @Override
    protected void onStart()
    {
        super.onStart();
        try {
            aa.clear();
            YAPI.EnableUSBHost(this);
            YAPI.RegisterHub("usb");
            YModule module = YModule.FirstModule();
            while (module != null) {
                if (module.get_productName().equals("Yocto-Altimeter")) {
                    String serial = module.get_serialNumber();
                    aa.add(serial);
                }
                module = module.nextModule();
            }
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
        aa.notifyDataSetChanged();
        handler.postDelayed(r, 500);
    }
}
```

```

    }

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

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int pos, long id)
    {
        serial = parent.getItemAtPosition(pos).toString();
    }

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

    final Runnable r = new Runnable()
    {
        public void run()
        {
            if (serial != null) {

                YAltitude alt_sensor = YAltitude.FindAltitude(serial);
                try {
                    TextView view = (TextView) findViewById(R.id.altfield);
                    view.setText(String.format("%.1f %s (QNH=%.1f)",
alt_sensor.getCurrentValue(), alt_sensor.getUnit(), alt_sensor.getQnh()));
                } catch (YAPI_Exception e) {
                    e.printStackTrace();
                }

                YPressure pres_sensor = YPressure.FindPressure(serial);
                try {
                    TextView view = (TextView) findViewById(R.id.presfield);
                    view.setText(String.format("%.1f %s", pres_sensor.getCurrentValue(),
pres_sensor.getUnit()));
                } catch (YAPI_Exception e) {
                    e.printStackTrace();
                }

                YTemperature temp_sensor = YTemperature.FindTemperature(serial);
                try {
                    TextView view = (TextView) findViewById(R.id.tempfield);
                    view.setText(String.format("%.1f %s", temp_sensor.getCurrentValue(),
temp_sensor.getUnit()));
                } catch (YAPI_Exception e) {
                    e.printStackTrace();
                }

            }
            handler.postDelayed(this, 1000);
        }
    };
}

```

18.6. 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.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Switch;
import android.widget.TextView;

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

public class ModuleControl extends Activity implements 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.setOnItemClickListener(this);
        aa = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item);
        aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        my_spin.setAdapter(aa);
    }

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

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

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

    private void DisplayModuleInfo()
    {
        TextView field;
        if (module == null)
            return;
        try {
            field = (TextView) findViewById(R.id.serialfield);
            field.setText(module.getSerialNumber());
            field = (TextView) findViewById(R.id.logicalnamefield);
            field.setText(module.getLogicalName());
            field = (TextView) findViewById(R.id.luminosityfield);
            field.setText(String.format("%d%", module.getLuminosity()));
            field = (TextView) findViewById(R.id.uptimefield);
            field.setText(module.getUpTime() / 1000 + " sec");
            field = (TextView) findViewById(R.id.usbcurrentfield);
            field.setText(module.getUsbCurrent() + " mA");
            Switch sw = (Switch) findViewById(R.id.beaconswitch);
            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 onItemClick(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.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

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

```

```

public class SaveSettings extends Activity implements 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 alentours de 100000 cycles. Pour résumer vous ne pouvez employer la fonction `YModule.saveToFlash()` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

Enumeration des modules

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

```

package com.yoctopuce.doc_examples;

import android.app.Activity;
import android.os.Bundle;
import android.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();
}
}

```

18.7. Gestion des erreurs

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

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

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

19. Mise à jour du firmware

Il existe plusieurs moyens de mettre à jour le firmware des modules Yoctopuce.

19.1. Le VirtualHub ou le YoctoHub

Il est possible de mettre à jour un module directement depuis l'interface web du VirtualHub 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 ne 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.

19.2. 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:\>
```

19.3. 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.

19.4. 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 `YFirmwareUpdate`.

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 a 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.**

19.5. 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 "mise 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).

```

YFirmwareUpdate fw_update;
fw_update = new YFirmwareUpdate(allBootLoader[0], newfirm, null);
int status = fw_update.startUpdate();
....

```

20. 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.

20.1. 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.

20.2. 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 œuvre, 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-Altimeter avec le numéro de de série *YALTIMK1-12345* et le nom logique *monModule*. L'URL suivante permettra de connaître l'état du module.

```
http://127.0.0.1:4444/bySerial/YALTIMK1-12345/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/YALTIMK1-12345/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/YALTIMK1-12345/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 altitude, il suffit de construire l'URL suivante.

```
http://127.0.0.1:4444/bySerial/YALTIMK1-12345/api/altitude.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/YALTIMK1-12345/api/altitude/logicalName
```

Assez logiquement, les attributs peuvent être modifiés de la même manière.

```
http://127.0.0.1:4444/bySerial/YALTIMK1-12345/api/altitude?logicalName=maFonction
```

Vous trouverez la liste des attributs disponibles pour votre Yocto-Altimeter 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/YALTIMK1-12345/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/YALTIMK1-12345/dataLogger.json?id=altitude&utc=1389801080
```

20.3. Utilisation des bibliothèques dynamiques

L'API Yoctopuce bas niveau est disponible sous différents formats de bibliothèque dynamiques écrites en C, dont les sources sont disponibles avec l'API C++. Utiliser une de ces bibliothèques 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
libyapi-armhf.so	Linux ARM HL
libyapi-i386.so	Linux Intel (32 bits)
yapi64.dll	Windows (64 bits)
yapi.dll	Windows (32 bits)

Ces bibliothèques dynamiques contiennent toutes les fonctionnalités nécessaires pour reconstruire entièrement toute l'API haut niveau dans n'importe quel langage capable d'intégrer ces bibliothèques. 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 connexion 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 appelée 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 = 'YALTIMK1-12345';
    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.
end.

```

VB6 et yapi.dll

Chaque point d'entrée de la DLL yapi.dll est disponible en deux versions, une classique C-decl, et une seconde compatible avec Visual Basic 6 préfixée avec *vb6_*.

20.4. 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.

21. Programmation avancée

Les chapitres précédents vous ont présenté dans chaque langage disponible les fonctions de programmation de base utilisables avec votre module Yocto-Altimeter. Ce chapitre présente de façon plus générale une utilisation plus avancée de votre module. Les exemples sont donnés dans le langage le plus populaire auprès des clients de Yoctopuce, à savoir C#. Néanmoins, vous trouverez dans les bibliothèques de programmation pour chaque langage 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.

21.1. 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 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 contrôler 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 *Exemples/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.

Détecter le changement de valeur d'un senseur

L'API Yoctopuce fournit aussi un système de callback permettant d'être prévenu automatiquement de la valeur d'un senseur, soit lorsqu'il a changé de manière significative, ou soit à intervalle fixe. Le code nécessaire à cet effet est assez similaire au code utilisé pour détecter l'arrivée d'un module.

Cette technique est très utile en particulier si vous voulez détecter des changements de valeur très rapides (de l'ordre de quelques millisecondes), car elle est beaucoup plus efficace (en terme de trafic sur USB) qu'une lecture répétée de la valeur et permet donc des meilleures performances.

L'appel des callbacks

Afin de permettre un meilleur contrôle du contexte d'appel, les callbacks de changement de valeurs et les callback périodiques ne sont appelés que pendant l'exécution des fonctions `YAPI.Sleep()` et `YAPI.HandleEvents()`. Vous devez donc appeler une de ces fonctions à intervalle régulier, soit depuis un timer soit depuis un thread parallèle.

```
while (true)
{
    // boucle d'attente permettant de déclencher les callbacks
    YAPI.Sleep(500, ref errmsg);
}
```

Dans les environnements de programmation où seul le thread d'interface a le droit d'interagir avec l'utilisateur, il est souvent approprié d'appeler `YAPI.HandleEvents()` depuis ce thread.

Le callback de changement de valeur

Ce type de callback est appelé lorsque un altimètre change de manière significative. Il prend en paramètre la fonction concernée et la nouvelle valeur, sous forme d'une chaîne de caractères¹.

```
static void valueChangeCallback(YAltitude fct, string value)
{
    Console.WriteLine(fct.get_hardwareId() + "=" + value);
}
```

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. Si vous avez besoin de garder la référence sur votre objet, vous pouvez la stocker directement dans l'objet YAltitude à l'aide de la fonction `set_userdata`. Il vous sera ainsi possible de la récupérer dans la procédure globale de callback en appelant `get_userdata`.

Mise en place du callback de changement de valeur

Le callback est mis en place pour une fonction Altitude donnée à l'aide de la méthode `registerValueCallback`. L'exemple suivant met en place un callback pour la première fonction Altitude disponible.

```
YAltitude f = YAltitude.FirstAltitude();
f.registerValueCallback(valueChangeCallback);
```

Vous remarquerez que chaque fonction d'un module peut ainsi avoir un callback différent. Par ailleurs, si vous prenez goût aux callback de changement de valeur, sachez qu'il ne sont pas limités aux senseurs, et que vous pouvez les utiliser avec tous les modules Yoctopuce (par exemple pour être notifié en cas de commutation d'un relais).

Le callback périodique

Ce type de callback est automatiquement appelé à intervalle réguliers. La fréquence d'appel peut être configurée individuellement pour chaque senseur, avec des fréquences pouvant aller de cent fois par seconde à une fois par heure. Le callback prend en paramètre la fonction concernée et la valeur mesurée, sous forme d'un objet YMeasure. Contrairement au callback de changement de valeur qui ne contient que la nouvelle valeur instantanée, l'objet YMeasure peut donner la valeur minimale, moyenne et maximale observée depuis le dernier appel du callback périodique. De plus, il contient aussi l'indication de l'heure exacte qui correspond à la mesure, de sorte à pouvoir l'interpréter correctement même en différé.

```
static void periodicCallback(YAltitude fct, YMeasure measure)
{
    Console.WriteLine(fct.get_hardwareId() + "=" +
        measure.get_averageValue());
}
```

Mise en place du callback périodique

Le callback est mis en place pour une fonction Altitude donnée à l'aide de la méthode `registerTimedReportCallback`. Pour que le callback périodique soit appelé, il faut aussi spécifier la fréquence d'appel à l'aide de la méthode `set_reportFrequency` (sinon le callback périodique est désactivé par défaut). La fréquence est spécifiée sous forme textuelle (comme pour l'enregistreur de données), en spécifiant le nombre d'occurrences par seconde (/s), par minute (/m) ou par heure (/h). La fréquence maximale est 100 fois par seconde (i.e. "100/s"), et fréquence minimale est 1 fois par heure (i.e. "1/h"). Lorsque la fréquence supérieure ou égale à 1/s, la mesure représente une valeur instantanée. Lorsque la fréquence est inférieure, la mesure comporte des valeurs minimale, moyenne et maximale distinctes sur la base d'un échantillonnage effectué automatiquement par le module.

¹ La valeur passée en paramètre est la même que celle rendue par la méthode `get_advertisedValue()`

L'exemple suivant met en place un callback périodique 4 fois par minute pour la première fonction Altitude disponible.

```
YAltitude f = YAltitude.FirstAltitude();
f.set_reportFrequency("4/m");
f.registerTimedReportCallback(periodicCallback);
```

Comme pour les callback de changement valeur, chaque fonction d'un module peut avoir un callback périodique différent.

Fonction callback générique

Parfois, il est souhaitable d'utiliser la même fonction de callback pour différents types de senseurs (par exemple pour une application de mesure générique). Ceci est possible en définissant le callback pour un objet de classe `YSensor` plutôt que `YAltitude`. Ainsi, la même fonction callback pourra être utilisée avec toutes les sous-classes de `YSensor` (et en particulier avec `YAltitude`). A l'intérieur du callback, on peut utiliser la méthode `get_unit()` pour obtenir l'unité physique du capteur si nécessaire pour l'affichage.

Un exemple concret

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 *Exemples/Prog-EventBased*.

21.2. L'enregistreur de données

Votre Yocto-Altimeter est équipé d'un enregistreur de données, aussi appelé datalogger, capable d'enregistrer en continu les mesures effectuées par le module. La fréquence d'enregistrement maximale est de cent fois par secondes (i.e. "100/s"), et la fréquence minimale est de une fois par heure (i.e. "1/h"). Lorsque la fréquence supérieure ou égale à 1/s, la mesure représente une valeur instantanée. Lorsque la fréquence est inférieure, l'enregistreur stocke non seulement une valeur moyenne, mais aussi les valeurs minimale et maximale observées durant la période, sur la base d'un échantillonnage effectué par le module.

La mémoire flash de l'enregistreur de données permet d'enregistrer environ 500'000 mesures instantanées, ou 125'000 mesures moyennées. Lorsque la mémoire du datalogger est saturée, les mesures les plus anciennes sont automatiquement effacées.

Prenez garde à ne pas laisser le datalogger fonctionner inutilement à haute vitesse: le nombre d'effacements possibles d'une mémoire flash est limité (typiquement 100'000 cycles d'écriture/effacement). A la vitesse maximale, l'enregistreur peut consommer plus de 100 cycles par jour ! Notez aussi qu'il se sert à rien d'enregistrer des valeurs plus rapidement que la fréquence de mesure du capteur lui-même.

Démarrage/arrêt du datalogger

Le datalogger peut être démarré à l'aide de la méthode `set_recording()`.

```
YDataLogger l = YDataLogger.FirstDataLogger();
l.set_recording(YDataLogger.RECORDING_ON);
```

Il est possible de faire démarrer automatiquement l'enregistrement des données dès la mise sous tension du module.

```
YDataLogger l = YDataLogger.FirstDataLogger();
l.set_autoStart(YDataLogger.AUTOSTART_ON);
l.get_module().saveToFlash(); // il faut sauver le réglage!
```

Remarque: les modules Yoctopuce n'ont pas besoin d'une connection USB active pour fonctionner: ils commencent à fonctionner dès qu'ils sont alimentés. Le Yocto-Altimeter peut enregistrer des

données sans être forcément raccordé à un ordinateur: il suffit d'activer le démarrage automatique du datalogger et d'alimenter le module avec un simple chargeur USB.

Effacement de la mémoire

La mémoire du datalogger peut être effacée à l'aide de la fonction `forgetAllDataStreams()`. Attention l'effacement est irréversible.

```
YDataLogger logger = YDataLogger.FirstDataLogger();
logger.forgetAllDataStreams();
```

Choix de la fréquence d'enregistrement

La fréquence d'enregistrement se configure individuellement pour chaque capteur, à l'aide de la méthode `set_logFrequency()`. La fréquence est spécifiée sous forme textuelle (comme pour les callback périodiques), en spécifiant le nombre d'occurrences par seconde (/s), par minute (/m) ou par heure (/h). La valeur par défaut est "1/s".

L'exemple suivant configure la fréquence d'enregistrement à 15 mesures par minute pour le premier capteur trouvé, quel que soit son type:

```
YSensor sensor = YSensor.FirstSensor();
sensor.set_logFrequency("15/m");
```

Pour économiser la mémoire flash, il est possible de désactiver l'enregistrement des mesures pour une fonction donnée. Pour ce faire, il suffit d'utiliser la valeur "OFF":

```
sensor.set_logFrequency("OFF");
```

Limitation: Le Yocto-Altimeter ne peut pas utiliser des fréquences différentes pour les notifications périodiques et pour l'enregistrement dans le datalogger. Il est possible de désactiver l'une ou l'autre de ces fonctionnalités indépendamment, mais si les deux sont activées, elles fonctionnent nécessairement à la même fréquence.

Récupération des données

Pour récupérer les données enregistrées dans la mémoire flash du Yocto-Altimeter, il faut appeler la méthode `get_recordedData()` de la fonction désirée, en spécifiant l'intervalle de temps qui nous intéresse. L'intervalle de temps est donnée à l'aide du timestamp UNIX de début et de fin. Il est aussi possible de spécifier 0 pour ne pas donner de limite de début ou de fin.

La fonction `get_recordedData()` ne retourne pas directement un tableau de valeurs mesurées, car selon la quantité de données, leur chargement pourrait potentiellement prendre trop de temps et entraver la réactivité de l'application. A la place, cette fonction retourne un objet `YDataSet` qui permet d'obtenir immédiatement une vue d'ensemble des données (résumé), puis d'en charger progressivement le détail lorsque c'est souhaitable.

Voici les principales méthodes pour accéder aux données enregistrées:

1. **dataset = sensor.get_recordedData(0,0):** on choisit l'intervalle de temps désiré
2. **dataset.loadMore():** pour charger les données progressivement
3. **dataset.get_summary():** retourne une mesure unique résumant tout l'intervalle de temps
4. **dataset.get_preview():** retourne un tableau de mesures représentant une version condensée de l'ensemble des mesures sur l'intervalle de temps choisi (réduction d'un facteur 200 environ)
5. **dataset.get_measures():** retourne un tableau contenant toutes les mesures de l'intervalle choisi (grandit au fur et à mesure de leur chargement avec `loadMore`)

Les mesures sont des objets `YMeasure`². On peut en y lire la valeur minimale, moyenne et maximale à l'aide des méthodes `get_minValue()`, `get_averageValue()` et `get_maxValue()` respectivement. Voici un petit exemple qui illustre ces fonctions:

```
// On veut récupérer toutes les données du datalogger
YDataSet dataset = sensor.get_recordedData(0, 0);

// Le 1er appel à loadMore() charge le résumé des données
dataset.loadMore();
YMeasure summary = dataset.get_summary();
string timeFmt = "dd MMM yyyy hh:mm:ss,fff";
string logFmt = "from {0} to {1} : average={2:0.00}{3}";
Console.WriteLine(String.Format(logFmt,
    summary.get_startTimeUTC_asDateTime().ToString(timeFmt),
    summary.get_endTimeUTC_asDateTime().ToString(timeFmt),
    summary.get_averageValue(), sensor.get_unit()));

// Les appels suivants à loadMore() chargent les mesures
Console.WriteLine("loading details");
int progress;
do {
    Console.Write(".");
    progress = dataset.loadMore();
} while(progress < 100);

// Ca y est, toutes les mesures sont là
List<YMeasure> details = dataset.get_measures();
foreach (YMeasure m in details) {
    Console.WriteLine(String.Format(logFmt,
        m.get_startTimeUTC_asDateTime().ToString(timeFmt),
        m.get_endTimeUTC_asDateTime().ToString(timeFmt),
        m.get_averageValue(), sensor.get_unit()));
}
```

Vous trouverez un exemple complet démontrant cette séquence dans la librairie de programmation Yoctopuce de chaque langage. L'exemple se trouve dans le répertoire *Examples/Prog-DataLogger*.

Horodatage

Le Yocto-Altimeter n'ayant pas de batterie, il n'est pas capable de deviner tout seul l'heure qu'il est au moment de sa mise sous tension. Néanmoins, le Yocto-Altimeter va automatiquement essayer de se mettre à l'heure de l'hôte auquel il est connecté afin de pouvoir correctement dater les mesures du datalogger:

- Lorsque le Yocto-Altimeter est branché à un ordinateur exécutant soit le VirtualHub, soit un programme quelconque utilisant la librairie Yoctopuce, il recevra l'heure de cet ordinateur.
- Lorsque le Yocto-Altimeter est branché à un YoctoHub-Ethernet, il recevra par ce biais l'heure que le YoctoHub a obtenu par le réseau (depuis un serveur du `pool.ntp.org`)
- Lorsque le Yocto-Altimeter est branché à un YoctoHub-Wireless, il recevra de celui-ci l'heure maintenue par son horloge RTC, précédemment obtenue par le réseau ou par un ordinateur.
- Lorsque le Yocto-Altimeter est branché à un appareil mobile Android, il recevra de celui-ci l'heure actuelle pour autant qu'une application utilisant la librairie Yoctopuce soit lancée.

Si aucune de ces conditions n'est remplie (par exemple si le module est simplement connecté à un chargeur USB), le Yocto-Altimeter fera de son mieux pour donner une date vraisemblable aux mesures, en repartant de l'heure des dernières mesures enregistrées. Ainsi, vous pouvez "mettre à l'heure" un Yocto-Altimeter "autonome" en le branchant sur un téléphone Android, lançant un enregistrement de données puis en le re-branchant tout seul sur un chargeur USB. Soyez toutefois conscients que, sans source de temps externe, l'horloge du Yocto-Altimeter peut dériver petit à petit (en principe pas plus de 0.3%).

² L'objet `YMeasure` du datalogger est exactement du même type que ceux qui sont passés aux fonctions de callback périodique.

21.3. Calibration des senseurs

Votre module Yocto-Altimeter est équipé d'un capteur numérique calibré en usine. Les valeurs qu'il renvoie sont censées être raisonnablement justes dans la majorité des cas. Il existe cependant des situations où des conditions extérieures peuvent avoir une influence sur les mesures.

L'API Yoctopuce offre le moyen de re-calibrer les valeurs mesurées par votre Yocto-Altimeter. Il ne s'agit pas de modifier les réglages hardware du module, mais plutôt d'effectuer une transformation a posteriori des mesures effectuées par le capteur. Cette transformation est pilotée par des paramètres qui seront stockés dans la mémoire flash du module, la rendant ainsi spécifique à chaque module. Cette re-calibration est donc entièrement software et reste parfaitement réversible.

Avant de décider de vous lancer dans la re-calibration de votre module Yocto-Altimeter, assurez-vous d'avoir bien compris les phénomènes qui influent sur les mesures de votre module, et que la différence en les valeurs vraies et les valeurs lues ne résultent pas d'une mauvaise utilisation ou d'un positionnement inadéquat.

Les modules Yoctopuce supportent deux types de calibration. D'une part une interpolation linéaire basée sur 1 à 5 points de référence, qui peut être effectuée directement à l'intérieur du Yocto-Altimeter. D'autre part l'API supporte une calibration arbitraire externe, implémentée à l'aide de callbacks.

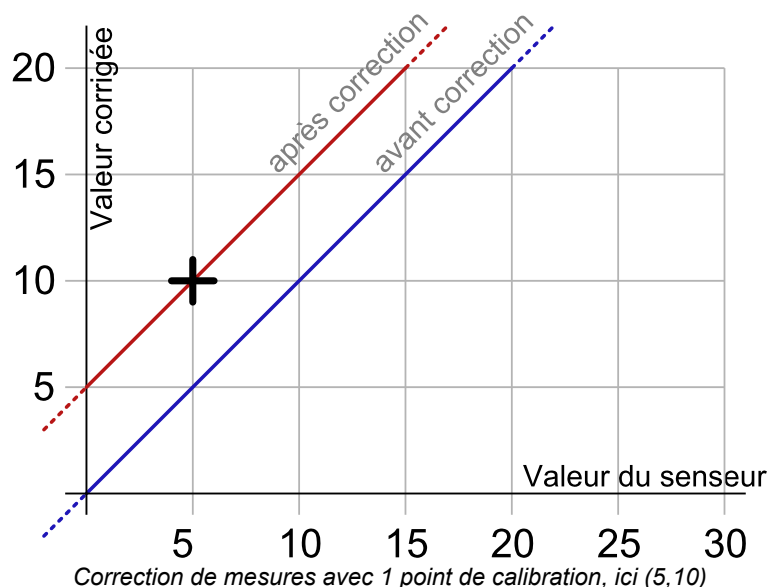
Interpolation linéaire 1 à 5 points

Ces transformations sont effectuées directement dans le Yocto-Altimeter ce qui signifie que vous n'avez qu'à enregistrer les points de calibration dans la mémoire flash du module, et tous les calculs de correction seront effectués de manière totalement transparente: La fonction `get_currentValue()` renverra la valeur corrigée, alors que la fonction `get_currentRawValue()` continuera de renvoyer la valeur avant correction.

Les points de calibration sont simplement des couples (*Valeur_lue*, *Valeur_corrigée*). Voyons l'influence du nombre de points de corrections sur les corrections.

Correction 1 point

La correction par 1 point ne fait qu'ajouter un décalage aux mesures. Par exemple, si vous fournissez le point de calibration (a, b), toutes les valeurs mesurées seront corrigées en leur ajoutant $b-a$, de sorte à ce que quand la valeur lue sur le capteur est a , la fonction altitude retournera b .



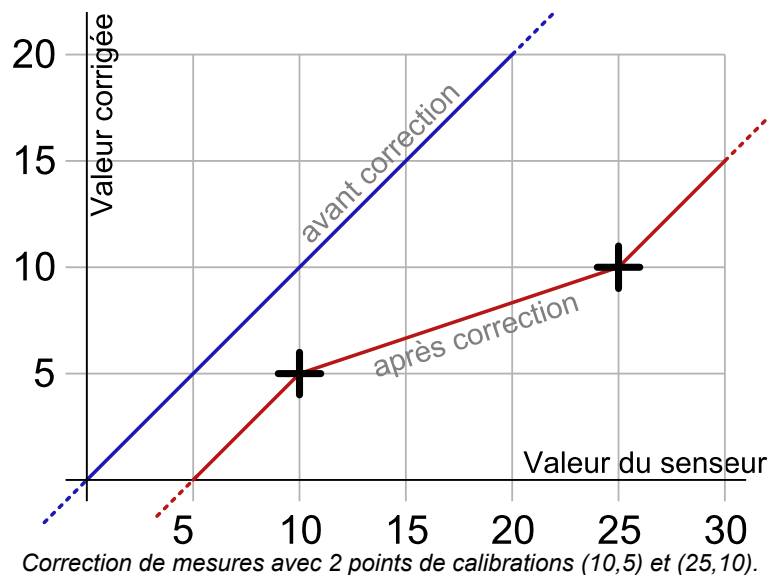
La mise en pratique est des plus simples: il suffit d'appeler la méthode `calibrateFromPoints()` de la fonction que l'on désire corriger. Le code suivant applique la correction illustrée sur le graphique ci-dessus à la première fonction altitude trouvée. Notez l'appel à la méthode `saveToFlash` du module

hébergeant la fonction, de manière à ce que le module n'oublie pas la calibration dès qu'il sera débranché.

```
Double[] ValuesBefore = {5};
Double[] ValuesAfter = {10};
YAltitude f = YAltitude.FirstAltitude();
f.calibrateFromPoints(ValuesBefore, ValuesAfter);
f.get_module().saveToFlash();
```

Correction 2 points

La correction 2 points permet d'effectuer à la fois un décalage et une multiplication par un facteur donné entre deux points. Si vous fournissez les deux points (a,b) et (c,d), le résultat de la fonction sera multiplié par $(d-b)/(c-a)$ dans l'intervalle [a,c] et décalé, de sorte à ce que quand la valeur lue par le senseur est a ou c, la fonction altitude retournera b ou respectivement d. A l'extérieur de l'intervalle [a,c], les valeurs seront simplement décalées de sorte à préserver la continuité des mesures: une augmentation de 1 sur la valeur lue par le senseur induira une augmentation de 1 sur la valeur retournée.



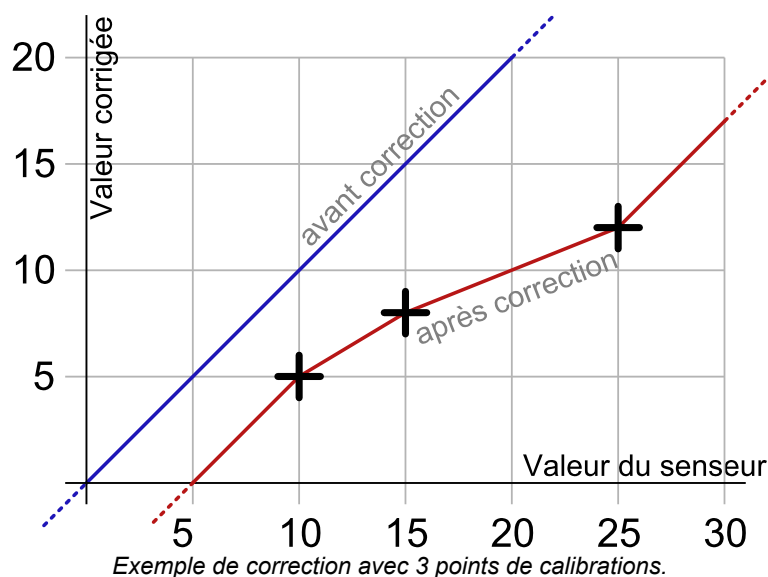
Le code permettant de programmer cette calibration est très similaire au code précédent

```
Double[] ValuesBefore = {10,25};
Double[] ValuesAfter = {5,10};
YAltitude f = YAltitude.FirstAltitude();
f.calibrateFromPoints(ValuesBefore, ValuesAfter);
f.get_module().saveToFlash();
```

Notez que les valeurs avant correction doivent être triées dans un ordre strictement croissant, sinon elles seront purement et simplement ignorées.

Correction de 3 à 5 points

Les corrections de 3 à 5 points ne sont qu'une généralisation de la méthode à deux points, permettant de ainsi de créer jusqu' 4 intervalles de correction pour plus de précision. Ces intervalles ne peuvent pas être disjoints.



Retour à la normale

Pour annuler les effets d'une calibration sur une fonction, il suffit d'appeler la méthode `calibrateFromPoints()` avec deux tableaux vides

```
Double[] ValuesBefore = {};
Double[] ValuesAfter = {};
YAltitude f = YAltitude.FirstAltitude();
f.calibrateFromPoints(ValuesBefore, ValuesAfter);
f.get_module().saveToFlash();
```

Vous trouverez dans le répertoire *Exemples\Prog-Calibration* des librairies Delphi, VB et C# une application permettant d'expérimenter les effets de la calibration 1 à 5 points.

Limitations

En raison des limitations de stockage et de traitement des valeurs flottantes dans le module Yoctopuce, les valeurs des valeurs lues et des valeur corrigées doivent respecter certaines contraintes numériques:

- Seules 3 décimales sont prises en compte (résolution de 0.001)
- La valeur minimale permise est -2'100'000
- La valeur maximale permise est +2'100'000

Interpolation arbitraire

Il est aussi possible de calculer l'interpolation à la place du module, pour calculer une interpolation par spline par exemple. Il suffit pour cela d'enregistrer un callback dans l'API. Ce callback devra préciser le nombre de points de correction auquel il s'attend.

```
public static double CustomInterpolation3Points(double rawValue, int calibType,
    int[] parameters, double[] beforeValues, double[] afterValues)
{
    double result;
    // la valeur à corriger est rawValue
    // les points de calibrations sont dans beforeValues et afterValues
    result = .... // interpolation de votre choix
    return result;
}
YAPI.RegisterCalibrationHandler(3, CustomInterpolation3Points);
```

Notez que ces callbacks d'interpolation sont globaux, et non pas spécifiques à chaque fonction. Ainsi à chaque fois que quelqu'un demandera une valeur à un module qui disposera dans sa mémoire flash du bon nombre de points de calibration, le callback correspondant sera appelé pour corriger la

valeur avant de la renvoyer, permettant ainsi de corriger les mesures de manière totalement transparente.

22. Référence de l'API de haut niveau

Ce chapitre résume les fonctions de l'API de haut niveau pour commander votre Yocto-Altimeter. La syntaxe et les types précis peuvent varier d'un langage à l'autre mais, sauf avis contraire toutes sont disponibles dans chaque langage. 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.

22.1. Fonctions générales

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_api.js'></script></code>
cpp	<code>#include "yocto_api.h"</code>
m	<code>#import "yocto_api.h"</code>
pas	<code>uses yocto_api;</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>
php	<code>require_once('yocto_api.php');</code>
es	in HTML: <code><script src='../lib/yocto_api.js'></script></code> in node.js: <code>require('yoctolib-es2017/yocto_api.js');</code>

Fonction globales

yCheckLogicalName(name)

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

yClearHTTPCallbackCacheDir(bool_removeFiles)

Désactive le cache de callback HTTP.

yDisableExceptions()

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

yEnableExceptions()

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

yEnableUSBHost(osContext)

Cette fonction est utilisée uniquement sous Android.

yFreeAPI()

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

yGetAPIVersion()

Retourne la version de la librairie Yoctopuce utilisée.

yGetCacheValidity()

Retourne la durée de validité des données chargée par la librairie.

yGetDeviceListValidity()

Retourne le délais entre chaque énumération forcée des YoctoHub utilisés.

yGetTickCount()

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

yHandleEvents(errmsg)

Maintient la communication de la librairie avec les modules Yoctopuce.

yInitAPI(mode, errmsg)

Initialise la librairie de programmation de Yoctopuce explicitement.

yPreregisterHub(url, errmsg)

Alternative plus tolérante à `RegisterHub()`.

`yRegisterDeviceArrivalCallback(arrivalCallback)`

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

`yRegisterDeviceRemovalCallback(removalCallback)`

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

`yRegisterHub(url, errmsg)`

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

`yRegisterHubDiscoveryCallback(hubDiscoveryCallback)`

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

`yRegisterHubWebsocketCallback(ws, errmsg, authpwd)`

Variante de la fonction `RegisterHub()` destinée à initialiser l'API Yoctopuce sur une session Websocket existante, dans le cadre d'un callback websocket entrant.

`yRegisterLogFunction(logfun)`

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

`ySelectArchitecture(arch)`

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

`ySetCacheValidity(cacheValidityMs)`

Change la durée de validité des données chargées par la librairie.

`ySetDelegate(object)`

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

`ySetDeviceListValidity(deviceListValidity)`

Change le délai entre chaque énumération forcée des YoctoHub utilisés.

`ySetHTTPCallbackCacheDir(str_directory)`

Active le cache du callback HTTP.

`ySetTimeout(callback, ms_timeout, args)`

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

`ySetUSBPacketAckMs(pktAckDelay)`

Active la quittance des paquets USB reçus par la librairie Yoctopuce.

`ySleep(ms_duration, errmsg)`

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

`yTestHub(url, mstimeout, errmsg)`

Test si un hub est joignable.

`yTriggerHubDiscovery(errmsg)`

Relance une détection des hubs réseau.

`yUnregisterHub(url)`

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

`yUpdateDeviceList(errmsg)`

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

`yUpdateDeviceList_async(callback, context)`

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

YAPI.CheckLogicalName() yCheckLogicalName()

YAPI

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

js	function yCheckLogicalName (name)
cpp	bool yCheckLogicalName (const string& name)
m	+(BOOL) CheckLogicalName :(NSString *) name
pas	function yCheckLogicalName (name : string): boolean
vb	function yCheckLogicalName (ByVal name As String) As Boolean
cs	bool CheckLogicalName (string name)
java	boolean CheckLogicalName (String name)
uwp	bool CheckLogicalName (string name)
py	def CheckLogicalName (name)
php	function yCheckLogicalName (\$name)
es	function 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() yClearHTTPCallbackCacheDir()

YAPI

Désactive le cache de callback HTTP.

```
php function yClearHTTPCallbackCacheDir( $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() yDisableExceptions()

YAPI

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

js	function yDisableExceptions ()
cpp	void yDisableExceptions ()
m	+(void) DisableExceptions
pas	procedure yDisableExceptions ()
vb	procedure yDisableExceptions ()
cs	void DisableExceptions ()
uwp	void DisableExceptions ()
py	def DisableExceptions ()
php	function yDisableExceptions ()
es	function 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() yEnableExceptions()

YAPI

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

js	function yEnableExceptions ()
cpp	void yEnableExceptions ()
m	+(void) EnableExceptions
pas	procedure yEnableExceptions ()
vb	procedure yEnableExceptions ()
cs	void EnableExceptions ()
uwp	void EnableExceptions ()
py	def EnableExceptions ()
php	function yEnableExceptions ()
es	function 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() yEnableUSBHost()

YAPI

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() yFreeAPI()

YAPI

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

js	function yFreeAPI ()
cpp	void yFreeAPI ()
m	+(void) FreeAPI
pas	procedure yFreeAPI ()
vb	procedure yFreeAPI ()
cs	void FreeAPI ()
java	void FreeAPI ()
uwp	void FreeAPI ()
py	def FreeAPI ()
php	function yFreeAPI ()
es	function FreeAPI ()

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

YAPI.GetAPIVersion() yGetAPIVersion()

YAPI

Retourne la version de la librairie Yoctopuce utilisée.

js	function yGetAPIVersion ()
cpp	string yGetAPIVersion ()
m	+(NSString*) GetAPIVersion
pas	function yGetAPIVersion (): string
vb	function yGetAPIVersion () As String
cs	String GetAPIVersion ()
java	String GetAPIVersion ()
uwp	string GetAPIVersion ()
py	def GetAPIVersion ()
php	function yGetAPIVersion ()
es	function 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() yGetCacheValidity()

YAPI

Retourne la durée de validité des données chargée par la librairie.

cpp	static u64 yGetCacheValidity ()
m	+(u64) GetCacheValidity
pas	function yGetCacheValidity (): u64
vb	function yGetCacheValidity () As Long
cs	ulong GetCacheValidity ()
java	long GetCacheValidity ()
uwp	Task<ulong> GetCacheValidity ()
py	def GetCacheValidity ()
php	function yGetCacheValidity ()
es	function 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() yGetDeviceListValidity()

YAPI

Retourne le délai entre chaque énumération forcée des YoctoHub utilisés.

cpp	static int yGetDeviceListValidity ()
m	+(int) GetDeviceListValidity
pas	function yGetDeviceListValidity (): LongInt
vb	function yGetDeviceListValidity () As Integer
cs	int GetDeviceListValidity ()
java	int GetDeviceListValidity ()
uwp	Task<int> GetDeviceListValidity ()
py	def GetDeviceListValidity ()
php	function yGetDeviceListValidity ()
es	function GetDeviceListValidity ()

Note: Cette fonction doit être appelée après yInitAPI.

Retourne :

le nombre de secondes entre chaque énumération.

YAPI.GetTickCount() yGetTickCount()

YAPI

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

js	function yGetTickCount ()
cpp	u64 yGetTickCount ()
m	+(u64) GetTickCount
pas	function yGetTickCount (): u64
vb	function yGetTickCount () As Long
cs	ulong GetTickCount ()
java	long GetTickCount ()
uwp	ulong GetTickCount ()
py	def GetTickCount ()
php	function yGetTickCount ()
es	function 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() yHandleEvents()

Maintient la communication de la librairie avec les modules Yoctopuce.

js	function yHandleEvents (errmsg)
cpp	YRETCODE yHandleEvents (string& errmsg)
m	+(YRETCODE) HandleEvents :(NSError**) errmsg
pas	function yHandleEvents (var errmsg : string): integer
vb	function yHandleEvents (ByRef errmsg As String) As YRETCODE
cs	YRETCODE HandleEvents (ref string errmsg)
java	int HandleEvents ()
uwp	Task<int> HandleEvents ()
py	def HandleEvents (errmsg =None)
php	function yHandleEvents (&\$ errmsg)
es	function 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() yInitAPI()

YAPI

Initialise la librairie de programmation de Yoctopuce explicitement.

js	function yInitAPI (mode , errmsg)
cpp	YRETCODE yInitAPI (int mode , string& errmsg)
m	+(YRETCODE) InitAPI :(int) mode :(NSError**) errmsg
pas	function yInitAPI (mode : integer, var errmsg : string): integer
vb	function yInitAPI (ByVal mode As Integer, ByRef errmsg As String) As Integer
cs	int InitAPI (int mode , ref string errmsg)
java	int InitAPI (int mode)
uwp	Task<int> InitAPI (int mode)
py	def InitAPI (mode , errmsg =None)
php	function yInitAPI (\$mode , & \$errmsg)
es	function 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 fonction est utilisée avec comme mode la valeur `Y_DETECT_NONE`, il faut explicitement appeler `yRegisterHub()` pour indiquer à la librairie sur quel VirtualHub les modules sont connectés, avant d'essayer d'y accéder.

Paramètres :

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

Retourne :

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

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

YAPI.PreregisterHub() yPreregisterHub()

Alternative plus tolérante à RegisterHub().

js	function yPreregisterHub (url , errmsg)
cpp	YRETCODE yPreregisterHub (const string& url , string& errmsg)
m	+(YRETCODE) PreregisterHub :(NSString *) url :(NSError**) errmsg
pas	function yPreregisterHub (url : string, var errmsg : string): integer
vb	function yPreregisterHub (ByVal url As String, ByRef errmsg As String) As Integer
cs	int PreregisterHub (string url , ref string errmsg)
java	int PreregisterHub (String url)
uwp	Task<int> PreregisterHub (string url)
py	def PreregisterHub (url , errmsg =None)
php	function yPreregisterHub (\$url, &\$errmsg)
es	function PreregisterHub (url , errmsg)

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

YAPI.RegisterDeviceArrivalCallback() yRegisterDeviceArrivalCallback()

YAPI

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

js	function yRegisterDeviceArrivalCallback (arrivalCallback)
cpp	void yRegisterDeviceArrivalCallback (yDeviceUpdateCallback arrivalCallback)
m	+(void) RegisterDeviceArrivalCallback :(yDeviceUpdateCallback) arrivalCallback
pas	procedure yRegisterDeviceArrivalCallback (arrivalCallback : yDeviceUpdateFunc)
vb	procedure yRegisterDeviceArrivalCallback (ByVal arrivalCallback As yDeviceUpdateFunc)
cs	void RegisterDeviceArrivalCallback (yDeviceUpdateFunc arrivalCallback)
java	void RegisterDeviceArrivalCallback (DeviceArrivalCallback arrivalCallback)
uwp	void RegisterDeviceArrivalCallback (DeviceUpdateHandler arrivalCallback)
py	def RegisterDeviceArrivalCallback (arrivalCallback)
php	function yRegisterDeviceArrivalCallback (\$arrivalCallback)
es	function RegisterDeviceArrivalCallback (arrivalCallback)

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

Paramètres :

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

YAPI.RegisterDeviceRemovalCallback() yRegisterDeviceRemovalCallback()

YAPI

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

js	function yRegisterDeviceRemovalCallback (removalCallback)
cpp	void yRegisterDeviceRemovalCallback (yDeviceUpdateCallback removalCallback)
m	+(void) RegisterDeviceRemovalCallback :(yDeviceUpdateCallback) removalCallback
pas	procedure yRegisterDeviceRemovalCallback (removalCallback : yDeviceUpdateFunc)
vb	procedure yRegisterDeviceRemovalCallback (ByVal removalCallback As yDeviceUpdateFunc)
cs	void RegisterDeviceRemovalCallback (yDeviceUpdateFunc removalCallback)
java	void RegisterDeviceRemovalCallback (DeviceRemovalCallback removalCallback)
uwp	void RegisterDeviceRemovalCallback (DeviceUpdateHandler removalCallback)
py	def RegisterDeviceRemovalCallback (removalCallback)
php	function yRegisterDeviceRemovalCallback (\$removalCallback)
es	function RegisterDeviceRemovalCallback (removalCallback)

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

Paramètres :

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

YAPI.RegisterHub() yRegisterHub()

YAPI

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

js	function yRegisterHub (url , errmsg)
cpp	YRETCODE yRegisterHub (const string& url , string& errmsg)
m	+(YRETCODE) RegisterHub :(NSString *) url :(NSError**) errmsg
pas	function yRegisterHub (url : string, var errmsg : string): integer
vb	function yRegisterHub (ByVal url As String, ByRef errmsg As String) As Integer
cs	int RegisterHub (string url , ref string errmsg)
java	int RegisterHub (String url)
uwp	Task<int> RegisterHub (string url)
py	def RegisterHub (url , errmsg =None)
php	function yRegisterHub (\$url, &\$errmsg)
es	function RegisterHub (url , errmsg)

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

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

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

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

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

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

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

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

YAPI.RegisterHubDiscoveryCallback() yRegisterHubDiscoveryCallback()

YAPI

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

cpp	void yRegisterHubDiscoveryCallback (YHubDiscoveryCallback hubDiscoveryCallback)
m	+(void) RegisterHubDiscoveryCallback : (YHubDiscoveryCallback) hubDiscoveryCallback
pas	procedure yRegisterHubDiscoveryCallback (hubDiscoveryCallback : YHubDiscoveryCallback)
vb	procedure yRegisterHubDiscoveryCallback (ByVal hubDiscoveryCallback As YHubDiscoveryCallback)
cs	void RegisterHubDiscoveryCallback (YHubDiscoveryCallback hubDiscoveryCallback)
java	void RegisterHubDiscoveryCallback (HubDiscoveryCallback hubDiscoveryCallback)
uwp	Task RegisterHubDiscoveryCallback (HubDiscoveryHandler hubDiscoveryCallback)
py	def RegisterHubDiscoveryCallback (hubDiscoveryCallback)

la fonction de callback reçoit deux chaînes de caractères en paramètre La première chaîne contient le numéro de série du hub réseau et la deuxième chaîne contient l'URL du hub. L'URL peut être passée directement en argument à la fonction yRegisterHub. Le callback sera appelé pendant l'exécution de la fonction 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() yRegisterHubWebsocketCallback()

YAPI

Variante de la fonction RegisterHub() 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() yRegisterLogFunction()

YAPI

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

cpp	void yRegisterLogFunction (yLogFunction logfun)
m	+(void) RegisterLogFunction :(yLogCallback) logfun
pas	procedure yRegisterLogFunction (logfun : yLogFunc)
vb	procedure yRegisterLogFunction (ByVal logfun As yLogFunc)
cs	void RegisterLogFunction (yLogFunc logfun)
java	void RegisterLogFunction (LogCallback logfun)
uwp	void RegisterLogFunction (LogHandler logfun)
py	def RegisterLogFunction (logfun)

Utile pour déboguer le fonctionnement de l'API.

Paramètres :

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

YAPI.SelectArchitecture() ySelectArchitecture()

YAPI

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

```
py def SelectArchitecture( arch)
```

Par défaut, la librairie Python détecte automatiquement la version de la librairie 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 recommandé 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", "i386", "x86_64", "32bit", "64bit"

Retourne :

rien.

En cas d'erreur, déclenche une exception.

YAPI.SetCacheValidity() ySetCacheValidity()

YAPI

Change la durée de validité des données chargées par la librairie.

cpp	static void ySetCacheValidity (u64 cacheValidityMs)
m	+(void) SetCacheValidity : (u64) cacheValidityMs
pas	procedure ySetCacheValidity (cacheValidityMs : u64)
vb	procedure ySetCacheValidity ()
cs	void SetCacheValidity (ulong cacheValidityMs)
java	void SetCacheValidity (long cacheValidityMs)
uwp	Task SetCacheValidity (ulong cacheValidityMs)
py	def SetCacheValidity (cacheValidityMs)
php	function ySetCacheValidity (\$cacheValidityMs)
es	function 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() ySetDelegate()

YAPI

(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() ySetDeviceListValidity()

YAPI

Change le délais entre chaque énumération forcée des YoctoHub utilisés.

cpp	static void ySetDeviceListValidity (int deviceListValidity)
m	+(void) SetDeviceListValidity : (int) deviceListValidity
pas	procedure ySetDeviceListValidity (deviceListValidity : LongInt)
vb	procedure ySetDeviceListValidity ()
cs	void SetDeviceListValidity (int deviceListValidity)
java	void SetDeviceListValidity (int deviceListValidity)
uwp	Task SetDeviceListValidity (int deviceListValidity)
py	def SetDeviceListValidity (deviceListValidity)
php	function ySetDeviceListValidity (\$ deviceListValidity)
es	function 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 délais. 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() ySetHTTPCallbackCacheDir()

YAPI

Active le cache du callback HTTP.

```
php function ySetHTTPCallbackCacheDir( $str_directory)
```

Le cache du callback HTTP permet de réduire de 50 à 70 % la quantité de données transmises 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.SetTimeout() ySetTimeout()

YAPI

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

```
js function ySetTimeout( callback, ms_timeout, args)
```

```
es function 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() ySetUSBPacketAckMs()

YAPI

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() ySleep()

YAPI

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

js	function ySleep (ms_duration , errmsg)
cpp	YRETCODE ySleep (unsigned ms_duration , string& errmsg)
m	+(YRETCODE) Sleep :(unsigned) ms_duration :(NSError **) errmsg
pas	function ySleep (ms_duration : integer, var errmsg : string): integer
vb	function ySleep (ByVal ms_duration As Integer, ByRef errmsg As String) As Integer
cs	int Sleep (int ms_duration , ref string errmsg)
java	int Sleep (long ms_duration)
uwp	Task<int> Sleep (ulong ms_duration)
py	def Sleep (ms_duration , errmsg =None)
php	function ySleep (\$ ms_duration , &\$ errmsg)
es	function 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() yTestHub()

YAPI

Test si un hub est joignable.

```

cpp YRETCODE yTestHub( const string& url, int mstimeout, string& errmsg)
m   +(YRETCODE) TestHub : (NSString*) url
                                : (int) mstimeout
                                : (NSError**) errmsg
pas function yTestHub( url: string,
                        mstimeout: integer,
                        var errmsg: string): integer
vb   function yTestHub( ByVal url As String,
                        ByVal mstimeout As Integer,
                        ByRef errmsg As String) As Integer
cs   int TestHub( string url, int mstimeout, ref string errmsg)
java int TestHub( String url, int mstimeout)
uwp  Task<int> TestHub( string url, uint mstimeout)
py   def TestHub( url, mstimeout, errmsg=None)
php  function yTestHub( $url, $mstimeout, &$errmsg)
es   function TestHub( url, mstimeout, errmsg)

```

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 RegisterHub. 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() yTriggerHubDiscovery()

YAPI

Relance une détection des hubs réseau.

cpp	<code>YRETCODE yTriggerHubDiscovery(string& errmsg)</code>
m	<code>+(YRETCODE) TriggerHubDiscovery : (NSError**) errmsg</code>
pas	<code>function yTriggerHubDiscovery(var errmsg: string): integer</code>
vb	<code>function yTriggerHubDiscovery(ByRef errmsg As String) As Integer</code>
cs	<code>int TriggerHubDiscovery(ref string errmsg)</code>
java	<code>int TriggerHubDiscovery()</code>
uwp	<code>Task<int> TriggerHubDiscovery()</code>
py	<code>def TriggerHubDiscovery(errmsg=None)</code>

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() yUnregisterHub()

YAPI

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

js	function yUnregisterHub (url)
cpp	void yUnregisterHub (const string& url)
m	+(void) UnregisterHub :(NSString *) url
pas	procedure yUnregisterHub (url : string)
vb	procedure yUnregisterHub (ByVal url As String)
cs	void UnregisterHub (string url)
java	void UnregisterHub (String url)
uwp	Task UnregisterHub (string url)
py	def UnregisterHub (url)
php	function yUnregisterHub (\$url)
es	function UnregisterHub (url)

Paramètres :

url une chaîne de caractères contenant "usb" ou

YAPI.UpdateDeviceList() yUpdateDeviceList()

YAPI

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

js	function yUpdateDeviceList (errmsg)
cpp	YRETCODE yUpdateDeviceList (string& errmsg)
m	+(YRETCODE) UpdateDeviceList :(NSError**) errmsg
pas	function yUpdateDeviceList (var errmsg : string): integer
vb	function yUpdateDeviceList (ByRef errmsg As String) As YRETCODE
cs	YRETCODE UpdateDeviceList (ref string errmsg)
java	int UpdateDeviceList ()
uwp	Task<int> UpdateDeviceList ()
py	def UpdateDeviceList (errmsg =None)
php	function yUpdateDeviceList (&\$ errmsg)
es	function 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 événements de hot-plug.

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

YAPI.UpdateDeviceList_async() yUpdateDeviceList_async()

YAPI

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.

22.2. Interface de contrôle du module

Cette interface est la même pour tous les modules USB de Yoctopuce. Elle permet de contrôler les paramètres généraux du module, et d'énumérer les fonctions fournies par chaque module.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_api.js'></script></code>
cpp	<code>#include "yocto_api.h"</code>
m	<code>#import "yocto_api.h"</code>
pas	<code>uses yocto_api;</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>
php	<code>require_once('yocto_api.php');</code>
es	in HTML: <code><script src=" ../lib/yocto_api.js"></script></code> in node.js: <code>require('yoctolib-es2017/yocto_api.js');</code>

Fonction globales

yFindModule(func)

Permet de retrouver un module d'après son numéro de série ou son nom logique.

yFindModuleInContext(yctx, func)

Permet de retrouver un module d'après un identifiant donné dans un Context YAPI.

yFirstModule()

Commence l'énumération des modules accessibles par la librairie.

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 *nième* 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 *nième* fonction du module.

module→functionName(functionIndex)

Retourne le nom logique de la *nième* fonction du module.

module→functionType(functionIndex)

Retourne le type de la *nième* fonction du module.

module→functionValue(functionIndex)

Retourne la valeur publiée par la *nième* 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 chaîne de caractère contenant les derniers logs du module.

module→get_logicalName()

Retourne le nom logique du module.

module→get_luminosity()

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

module→get_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 de version matériel du module, préprogrammé en usine.

module→get_rebootCountdown()

Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

module→get_serialNumber()

Retourne le numéro de série du module, préprogrammé en usine.

module→get_subDevices()

Retourne la liste des modules branchés au module courant.

module→get_upTime()

Retourne le nombre 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 déclenchement 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)

Prepares une mise à jour de firmware du module.

module→updateFirmwareEx(path, force)

Prepares 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() yFindModule()

YModule

Permet de retrouver un module d'après son numéro de série ou son nom logique.

js	function yFindModule (func)
cpp	YModule* yFindModule (string func)
m	+(YModule*) FindModule : (NSString*) func
pas	function yFindModule (func : string): TYModule
vb	function yFindModule (ByVal func As String) As YModule
cs	YModule FindModule (string func)
java	YModule FindModule (String func)
uwp	YModule FindModule (string func)
py	def FindModule (func)
php	function yFindModule (\$func)
es	function FindModule (func)

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

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 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() yFindModuleInContext()

YModule

Permet de retrouver un module d'après un identifiant donné dans un Context YAPI.

java	YModule FindModuleInContext(YAPIContext yctx , String func)
uwp	YModule FindModuleInContext(YAPIContext yctx , string func)
es	function 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 où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YModule.isOnline()` pour tester si le module est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence le module sans ambiguïté

Retourne :

un objet de classe `YModule` qui permet ensuite de contrôler le module.

YModule.FirstModule() yFirstModule()

YModule

Commence l'énumération des modules accessibles par la librairie.

js	function yFirstModule ()
cpp	YModule* yFirstModule ()
m	+(YModule*) FirstModule
pas	function yFirstModule (): TModule
vb	function yFirstModule () As YModule
cs	YModule FirstModule ()
java	YModule FirstModule ()
uwp	YModule FirstModule ()
py	def FirstModule ()
php	function yFirstModule ()
es	function FirstModule ()

Utiliser la fonction `YModule.nextModule()` pour itérer sur les autres modules.

Retourne :

un pointeur sur un objet `YModule`, correspondant au premier module accessible en ligne, ou `null` si aucun module n'a été trouvé.

module→checkFirmware()**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	function checkFirmware (path : string, onlynew : boolean): string
vb	function checkFirmware () As String
cs	string checkFirmware (string path , bool onlynew)
java	String checkFirmware (String path , boolean onlynew)
uwp	Task<string> checkFirmware (string path , bool onlynew)
py	def checkFirmware (path , onlynew)
php	function checkFirmware (\$ path , \$ onlynew)
es	function checkFirmware (path , 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	procedure clearCache ()
vb	procedure clearCache ()
cs	void clearCache ()
java	void clearCache ()
py	def clearCache ()
php	function clearCache ()
es	function clearCache ()

Invalide le cache des valeurs courantes du module. Force le prochain appel à une méthode `get_xxx()` ou `loadxxx()` pour charger 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	function describe (): string
vb	function describe () As String
cs	string describe ()
java	String describe ()
py	def describe ()
php	function describe ()
es	function 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	function download (pathname : string): TByteArray
vb	function download () As Byte
cs	byte[] download (string pathname)
java	byte[] download (String pathname)
uwp	Task<byte[]> download (string pathname)
py	def download (pathname)
php	function download (\$ pathname)
es	function download (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 *nième* fonction du module.

js	function functionBaseType (functionIndex)
cpp	string functionBaseType (int functionIndex)
pas	function functionBaseType (functionIndex : integer): string
vb	function functionBaseType (ByVal functionIndex As Integer) As String
cs	string functionBaseType (int functionIndex)
java	String functionBaseType (int functionIndex)
py	def functionBaseType (functionIndex)
php	function functionBaseType (\$functionIndex)
es	function 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	function functionCount (): integer
vb	function functionCount () As Integer
cs	int functionCount ()
java	int functionCount ()
py	def functionCount ()
php	function functionCount ()
es	function 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 *nième* fonction du module.

js	function functionId (functionIndex)
cpp	string functionId (int functionIndex)
m	-(NSString*) functionId : (int) functionIndex
pas	function functionId (functionIndex : integer): string
vb	function functionId (ByVal functionIndex As Integer) As String
cs	string functionId (int functionIndex)
java	String functionId (int functionIndex)
py	def functionId (functionIndex)
php	function functionId (\$functionIndex)
es	function 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 *nième* fonction du module.

js	function functionName (functionIndex)
cpp	string functionName (int functionIndex)
m	-(NSString*) functionName : (int) functionIndex
pas	function functionName (functionIndex : integer): string
vb	function functionName (ByVal functionIndex As Integer) As String
cs	string functionName (int functionIndex)
java	String functionName (int functionIndex)
py	def functionName (functionIndex)
php	function functionName (\$functionIndex)
es	function 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 *nième* fonction du module.

js	function functionType (functionIndex)
cpp	string functionType (int functionIndex)
pas	function functionType (functionIndex : integer): string
vb	function functionType (ByVal functionIndex As Integer) As String
cs	string functionType (int functionIndex)
java	String functionType (int functionIndex)
py	def functionType (functionIndex)
php	function functionType (\$functionIndex)
es	function 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 *nième* fonction du module.

js	function functionValue (functionIndex)
cpp	string functionValue (int functionIndex)
m	-(NSString*) functionValue : (int) functionIndex
pas	function functionValue (functionIndex : integer): string
vb	function functionValue (ByVal functionIndex As Integer) As String
cs	string functionValue (int functionIndex)
java	String functionValue (int functionIndex)
py	def functionValue (functionIndex)
php	function functionValue (\$functionIndex)
es	function 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	function get_allSettings (): TByteArray
vb	function get_allSettings () As Byte
cs	byte[] get_allSettings ()
java	byte[] get_allSettings ()
uwp	Task<byte[]> get_allSettings ()
py	def get_allSettings ()
php	function get_allSettings ()
es	function get_allSettings ()
cmd	YModule target get_allSettings

Utile pour sauvgarder les noms logiques, les calibrations et fichiers 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	function get_beacon (): Integer
vb	function get_beacon () As Integer
cs	int get_beacon ()
java	int get_beacon ()
uwp	Task<int> get_beacon ()
py	def get_beacon ()
php	function get_beacon ()
es	function get_beacon ()
cmd	YModule target get_beacon

Retourne :

soit Y_BEACON_OFF, soit Y_BEACON_ON, selon l'état de la balise de localisation

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

module→get_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	function get_errorMessage (): string
vb	function get_errorMessage () As String
cs	string get_errorMessage ()
java	String get_errorMessage ()
py	def get_errorMessage ()
php	function get_errorMessage ()
es	function 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	<code>function get_errorType()</code>
cpp	<code>YRETCODE get_errorType()</code>
pas	<code>function get_errorType(): YRETCODE</code>
vb	<code>function get_errorType() As YRETCODE</code>
cs	<code>YRETCODE get_errorType()</code>
java	<code>int get_errorType()</code>
py	<code>def get_errorType()</code>
php	<code>function get_errorType()</code>
es	<code>function get_errorType()</code>

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

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation 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	function get_firmwareRelease (): string
vb	function get_firmwareRelease () As String
cs	string get_firmwareRelease ()
java	String get_firmwareRelease ()
uwp	Task<string> get_firmwareRelease ()
py	def get_firmwareRelease ()
php	function get_firmwareRelease ()
es	function 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 Y_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	function get_functionIds (funType : string): TStringArray
vb	function get_functionIds () As List
cs	List<string> get_functionIds (string funType)
java	ArrayList<String> get_functionIds (String funType)
uwp	Task<List<string>> get_functionIds (string funType)
py	def get_functionIds (funType)
php	function get_functionIds (\$ funType)
es	function get_functionIds (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	def get_hardwareId ()
php	function get_hardwareId ()
es	function 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	-(NSData*) icon2d
pas	function get_icon2d (): TByteArray
vb	function get_icon2d () As Byte
cs	byte[] get_icon2d ()
java	byte[] get_icon2d ()
uwp	Task<byte[]> get_icon2d ()
py	def get_icon2d ()
php	function get_icon2d ()
es	function 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 chaîne de caractère contenant les derniers logs du module.

js	function get_lastLogs ()
cpp	string get_lastLogs ()
m	-(NSString*) lastLogs
pas	function get_lastLogs (): string
vb	function get_lastLogs () As String
cs	string get_lastLogs ()
java	String get_lastLogs ()
uwp	Task<string> get_lastLogs ()
py	def get_lastLogs ()
php	function get_lastLogs ()
es	function 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	function get_logicalName (): string
vb	function get_logicalName () As String
cs	string get_logicalName ()
java	String get_logicalName ()
uwp	Task<string> get_logicalName ()
py	def get_logicalName ()
php	function get_logicalName ()
es	function 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 Y_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	function get_luminosity (): LongInt
vb	function get_luminosity () As Integer
cs	int get_luminosity ()
java	int get_luminosity ()
uwp	Task<int> get_luminosity ()
py	def get_luminosity ()
php	function get_luminosity ()
es	function 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 Y_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	function get_parentHub (): string
vb	function get_parentHub () As String
cs	string get_parentHub ()
java	String get_parentHub ()
uwp	Task<string> get_parentHub ()
py	def get_parentHub ()
php	function get_parentHub ()
es	function 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	function get_persistentSettings (): Integer
vb	function get_persistentSettings () As Integer
cs	int get_persistentSettings ()
java	int get_persistentSettings ()
uwp	Task<int> get_persistentSettings ()
py	def get_persistentSettings ()
php	function get_persistentSettings ()
es	function get_persistentSettings ()
cmd	YModule target get_persistentSettings

Retourne :

une valeur parmi Y_PERSISTENTSETTINGS_LOADED, Y_PERSISTENTSETTINGS_SAVED et Y_PERSISTENTSETTINGS_MODIFIED représentant l'état courant des réglages persistents du module

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

module→**get_productId()****YModule****module**→**productId()**

Retourne l'identifiant USB du module, préprogrammé en usine.

js	function get_productId ()
cpp	int get_productId ()
m	-(int) productId
pas	function get_productId (): LongInt
vb	function get_productId () As Integer
cs	int get_productId ()
java	int get_productId ()
uwp	Task<int> get_productId ()
py	def get_productId ()
php	function get_productId ()
es	function 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 Y_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	function get_productName (): string
vb	function get_productName () As String
cs	string get_productName ()
java	String get_productName ()
uwp	Task<string> get_productName ()
py	def get_productName ()
php	function get_productName ()
es	function 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 Y_PRODUCTNAME_INVALID.

module→**get_productRelease()****YModule****module**→**productRelease()**

Retourne le numéro de version matériel du module, préprogrammé en usine.

js	function get_productRelease ()
cpp	int get_productRelease ()
m	-(int) productRelease
pas	function get_productRelease (): LongInt
vb	function get_productRelease () As Integer
cs	int get_productRelease ()
java	int get_productRelease ()
uwp	Task<int> get_productRelease ()
py	def get_productRelease ()
php	function get_productRelease ()
es	function get_productRelease ()
cmd	YModule target get_productRelease

Retourne :

un entier représentant le numéro de version matériel du module, préprogrammé en usine

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

module→get_rebootCountdown()**YModule****module→rebootCountdown()**

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	function get_rebootCountdown (): LongInt
vb	function get_rebootCountdown () As Integer
cs	int get_rebootCountdown ()
java	int get_rebootCountdown ()
uwp	Task<int> get_rebootCountdown ()
py	def get_rebootCountdown ()
php	function get_rebootCountdown ()
es	function 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 Y_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	function get_serialNumber (): string
vb	function get_serialNumber () As String
cs	string get_serialNumber ()
java	String get_serialNumber ()
uwp	Task<string> get_serialNumber ()
py	def get_serialNumber ()
php	function get_serialNumber ()
es	function 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 Y_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	function get_subDevices (): TStringArray
vb	function get_subDevices () As List
cs	List<string> get_subDevices ()
java	ArrayList<String> get_subDevices ()
uwp	Task<List<string>> get_subDevices ()
py	def get_subDevices ()
php	function get_subDevices ()
es	function 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 nombre de millisecondes écoulées depuis la mise sous tension du module

js	function get_upTime ()
cpp	s64 get_upTime ()
m	-(s64) upTime
pas	function get_upTime (): int64
vb	function get_upTime () As Long
cs	long get_upTime ()
java	long get_upTime ()
uwp	Task<long> get_upTime ()
py	def get_upTime ()
php	function get_upTime ()
es	function get_upTime ()
cmd	YModule target get_upTime

Retourne :

un entier représentant le nombre de millisecondes écoulées depuis la mise sous tension du module

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

module→**get_url()****YModule****module**→**url()**

Retourne l'URL utilisée pour accéder au module.

js	function get_url ()
cpp	string get_url ()
m	-(NSString*) url
pas	function get_url (): string
vb	function get_url () As String
cs	string get_url ()
java	String get_url ()
uwp	Task<string> get_url ()
py	def get_url ()
php	function get_url ()
es	function 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	function get_usbCurrent (): LongInt
vb	function get_usbCurrent () As Integer
cs	int get_usbCurrent ()
java	int get_usbCurrent ()
uwp	Task<int> get_usbCurrent ()
py	def get_usbCurrent ()
php	function get_usbCurrent ()
es	function 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 Y_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	function get_userdata (): Tobject
vb	function get_userdata () As Object
cs	object get_userdata ()
java	Object get_userdata ()
py	def get_userdata ()
php	function get_userdata ()
es	function get_userdata ()

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

Retourne :

l'objet stocké précédemment par l'appelant.

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	function get_userVar (): LongInt
vb	function get_userVar () As Integer
cs	int get_userVar ()
java	int get_userVar ()
uwp	Task<int> get_userVar ()
py	def get_userVar ()
php	function get_userVar ()
es	function 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 Y_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	function hasFunction (funcId : string): boolean
vb	function hasFunction () As Boolean
cs	bool hasFunction (string funcId)
java	boolean hasFunction (String funcId)
uwp	Task<bool> hasFunction (string funcId)
py	def hasFunction (funcId)
php	function hasFunction (\$funcId)
es	function hasFunction (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	function isOnline (): boolean
vb	function isOnline () As Boolean
cs	bool isOnline ()
java	boolean isOnline ()
py	def isOnline ()
php	function isOnline ()
es	function 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	function load (msValidity : u64): YRETCODE
vb	function load (ByVal msValidity As Long) As YRETCODE
cs	YRETCODE load (ulong msValidity)
java	int load (long msValidity)
py	def load (msValidity)
php	function load (\$msValidity)
es	function 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	function log (text : string): LongInt
vb	function log () As Integer
cs	int log (string text)
java	int log (String text)
uwp	Task<int> log (string text)
py	def log (text)
php	function log (\$ text)
es	function log (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	-(YModule*) nextModule
pas	function nextModule (): TYModule
vb	function nextModule () As YModule
cs	YModule nextModule ()
java	YModule nextModule ()
uwp	YModule nextModule ()
py	def nextModule ()
php	function nextModule ()
es	function nextModule ()

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	function reboot (secBeforeReboot : LongInt): LongInt
vb	function reboot () As Integer
cs	int reboot (int secBeforeReboot)
java	int reboot (int secBeforeReboot)
uwp	Task<int> reboot (int secBeforeReboot)
py	def reboot (secBeforeReboot)
php	function reboot (\$secBeforeReboot)
es	function reboot (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) callback
pas	function registerBeaconCallback (callback : TYModuleBeaconCallback): LongInt
vb	function registerBeaconCallback () As Integer
cs	int registerBeaconCallback (BeaconCallback callback)
java	int registerBeaconCallback (BeaconCallback callback)
uwp	Task<int> registerBeaconCallback (BeaconCallback callback)
py	def registerBeaconCallback (callback)
php	function registerBeaconCallback (\$callback)
es	function 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) callback
pas	function registerConfigChangeCallback (callback : TYModuleConfigChangeCallback): LongInt
vb	function registerConfigChangeCallback () As Integer
cs	int registerConfigChangeCallback (ConfigChangeCallback callback)
java	int registerConfigChangeCallback (ConfigChangeCallback callback)
uwp	Task<int> registerConfigChangeCallback (ConfigChangeCallback callback)
py	def registerConfigChangeCallback (callback)
php	function registerConfigChangeCallback (\$callback)
es	function 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.

```
cpp void registerLogCallback( YModuleLogCallback callback)
```

```
m -(void) registerLogCallback : (YModuleLogCallback) callback
```

```
vb function registerLogCallback( ByVal callback As YModuleLogCallback) As Integer
```

```
cs int registerLogCallback( LogCallback callback)
```

```
java void registerLogCallback( LogCallback callback)
```

```
py def registerLogCallback( callback)
```

Utile pour déboguer le fonctionnement d'un module Yoctopuce.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet module qui a produit un log, un chaîne de caractère qui 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	function revertFromFlash (): LongInt
vb	function revertFromFlash () As Integer
cs	int revertFromFlash ()
java	int revertFromFlash ()
uwp	Task<int> revertFromFlash ()
py	def revertFromFlash ()
php	function revertFromFlash ()
es	function 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	function saveToFlash (): LongInt
vb	function saveToFlash () As Integer
cs	int saveToFlash ()
java	int saveToFlash ()
uwp	Task<int> saveToFlash ()
py	def saveToFlash ()
php	function saveToFlash ()
es	function saveToFlash ()
cmd	YModule target saveToFlash

Attention le nombre total de sauvegardes possibles durant la vie du module est limité (environ 100000 cycles). N'appellez pas cette fonction dans une boucle.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→**set_allSettings()****YModule****module**→**setAllSettings()**

Rétablit tous les paramètres du module.

js	function set_allSettings (settings)
cpp	int set_allSettings (string settings)
m	-(int) setAllSettings : (NSData*) settings
pas	function set_allSettings (settings : TByteArray): LongInt
vb	procedure set_allSettings ()
cs	int set_allSettings ()
java	int set_allSettings (byte[] settings)
uwp	Task<int> set_allSettings ()
py	def set_allSettings (settings)
php	function set_allSettings (\$ settings)
es	function set_allSettings (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 tous les paramètres

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→set_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	function set_allSettingsAndFiles (settings : TByteArray): LongInt
vb	procedure set_allSettingsAndFiles ()
cs	int set_allSettingsAndFiles ()
java	int set_allSettingsAndFiles (byte[] settings)
uwp	Task<int> set_allSettingsAndFiles ()
py	def set_allSettingsAndFiles (settings)
php	function set_allSettingsAndFiles (\$settings)
es	function set_allSettingsAndFiles (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 tous les paramètres

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→**set_beacon()****YModule****module**→**setBeacon()**

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	function 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	Task<int> set_beacon (int newval)
py	def set_beacon (newval)
php	function set_beacon (\$ newval)
es	function set_beacon (newval)
cmd	YModule target set_beacon newval

Paramètres :

newval soit Y_BEACON_OFF, soit Y_BEACON_ON

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→**set_logicalName()****YModule****module**→**setLogicalName()**

Change le nom logique du module.

js	function set_logicalName (newval)
cpp	int set_logicalName (const string& newval)
m	-(int) setLogicalName : (NSString*) newval
pas	function set_logicalName (newval : string): integer
vb	function set_logicalName (ByVal newval As String) As Integer
cs	int set_logicalName (string newval)
java	int set_logicalName (String newval)
uwp	Task<int> set_logicalName (string newval)
py	def set_logicalName (newval)
php	function set_logicalName (\$newval)
es	function set_logicalName (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	function 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	Task<int> set_luminosity (int newval)
py	def set_luminosity (newval)
php	function set_luminosity (\$newval)
es	function set_luminosity (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	procedure set_userdata (data : Tobject)
vb	procedure set_userdata (ByVal data As Object)
cs	void set_userdata (object data)
java	void set_userdata (Object data)
py	def set_userdata (data)
php	function set_userdata (\$data)
es	function set_userdata (data)

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

Paramètres :

data objet quelconque à mémoriser

module→**set_userVar()****YModule****module**→**setUserVar()**

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	function 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	Task<int> set_userVar (int newval)
py	def set_userVar (newval)
php	function set_userVar (\$newval)
es	function set_userVar (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 déclenchement 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	function triggerConfigChangeCallback (): LongInt
vb	function triggerConfigChangeCallback () As Integer
cs	int triggerConfigChangeCallback ()
java	int triggerConfigChangeCallback ()
uwp	Task<int> triggerConfigChangeCallback ()
py	def triggerConfigChangeCallback ()
php	function triggerConfigChangeCallback ()
es	function 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	function triggerFirmwareUpdate (secBeforeReboot : LongInt): LongInt
vb	function triggerFirmwareUpdate () As Integer
cs	int triggerFirmwareUpdate (int secBeforeReboot)
java	int triggerFirmwareUpdate (int secBeforeReboot)
uwp	Task<int> triggerFirmwareUpdate (int secBeforeReboot)
py	def triggerFirmwareUpdate (secBeforeReboot)
php	function triggerFirmwareUpdate (\$ secBeforeReboot)
es	function triggerFirmwareUpdate (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	function updateFirmware (path : string): TYFirmwareUpdate
vb	function updateFirmware () As YFirmwareUpdate
cs	YFirmwareUpdate updateFirmware (string path)
java	YFirmwareUpdate updateFirmware (String path)
uwp	Task<YFirmwareUpdate> updateFirmware (string path)
py	def updateFirmware (path)
php	function updateFirmware (\$path)
es	function updateFirmware (path)
cmd	YModule target updateFirmware path

Cette méthode retourne un objet `YFirmwareUpdate` qui est utilisé pour mettre à jour le firmware du module.

Paramètres :

path le path du fichier `.byn` à utiliser

Retourne :

un objet `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	function updateFirmwareEx (path : string, force : boolean): TYFirmwareUpdate
vb	function updateFirmwareEx () As YFirmwareUpdate
cs	YFirmwareUpdate updateFirmwareEx (string path , bool force)
java	YFirmwareUpdate updateFirmwareEx (String path , boolean force)
uwp	Task<YFirmwareUpdate> updateFirmwareEx (string path , bool force)
py	def updateFirmwareEx (path , force)
php	function updateFirmwareEx (\$ path , \$ force)
es	function updateFirmwareEx (path , force)
cmd	YModule target updateFirmwareEx path force

Cette méthode retourne un objet YFirmwareUpdate qui est utilisé pour mettre à jour le firmware du module.

Paramètres :

path le path du fichier .byn à utiliser

force vrai pour forcer la mise à jour même si un prérequis ne semble pas satisfait

Retourne :

un objet 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)
```

```
es function 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.

22.3. Interface de la fonction Altitude

La classe YAltitude permet de lire et de configurer les capteurs d'altitude Yoctopuce. Elle hérite de la class YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, pour les capteurs barométriques, elle permet de faire la configuration de la pression de référence au niveau de la mer (QNH).

Pour utiliser les fonctions décrites ici, vous devez inclure:

es	in HTML: <code><script src="../../lib/yocto_altitude.js"></script></code> in node.js: <code>require('yoctolib-es2017/yocto_altitude.js');</code>
js	<code><script type='text/javascript' src='yocto_altitude.js'></script></code>
cpp	<code>#include "yocto_altitude.h"</code>
m	<code>#import "yocto_altitude.h"</code>
pas	<code>uses yocto_altitude;</code>
vb	<code>yocto_altitude.vb</code>
cs	<code>yocto_altitude.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YAltitude;</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YAltitude;</code>
py	<code>from yocto_altitude import *</code>
php	<code>require_once('yocto_altitude.php');</code>

Fonction globales

yFindAltitude(func)

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

yFindAltitudeInContext(yctx, func)

Permet de retrouver un altimetre d'après un identifiant donné dans un Context YAPI.

yFirstAltitude()

Commence l'énumération des altimètres accessibles par la librairie.

yFirstAltitudeInContext(yctx)

Commence l'énumération des altimètres accessibles par la librairie.

Méthodes des objets YAltitude

altitude→calibrateFromPoints(rawValues, refValues)

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

altitude→clearCache()

Invalide le cache.

altitude→describe()

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

altitude→get_advMode()

Retourne le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

altitude→get_advertisedValue()

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

altitude→get_currentRawValue()

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

altitude→get_currentValue()

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

altitude→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

altitude→get_errorMessage()

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

altitude→get_errorType()

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

altitude→get_friendlyName()

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

altitude→get_functionDescriptor()

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

altitude→get_functionId()

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

altitude→get_hardwareId()

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

altitude→get_highestValue()

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

altitude→get_logFrequency()

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

altitude→get_logicalName()

Retourne le nom logique de l'altimètre.

altitude→get_lowestValue()

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

altitude→get_module()

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

altitude→get_module_async(callback, context)

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

altitude→get_qnh()

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

altitude→get_recordedData(startTime, endTime)

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

altitude→get_reportFrequency()

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

altitude→get_resolution()

Retourne la résolution des valeurs mesurées.

altitude→get_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.

altitude→get_technology()

Renvoie la technologie employée par la fonction pour calculer l'altitude.

altitude→get_unit()

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

altitude→get_userData()

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

altitude→isOnline()

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

altitude→isOnline_async(callback, context)

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

altitude→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

altitude→load(msValidity)

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

altitude→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.

altitude→loadCalibrationPoints(rawValues, refValues)

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

altitude→load_async(msValidity, callback, context)

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

altitude→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

altitude→nextAltitude()

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

altitude→registerTimedReportCallback(callback)

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

altitude→registerValueCallback(callback)

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

altitude→set_advMode(newval)

Modifie le mode de calcul de la valeur publiée jusqu'au hub parent (`advertisedValue`).

altitude→set_currentValue(newval)

Modifie l'altitude actuelle supposée.

altitude→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

altitude→set_logFrequency(newval)

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

altitude→set_logicalName(newval)

Modifie le nom logique de l'altimètre.

altitude→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

altitude→set_qnh(newval)

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

altitude→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

altitude→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

altitude→set_userData(data)

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

altitude→startDataLogger()

Démarre l'enregistreur de données du module.

altitude→stopDataLogger()

Arrête l'enregistreur de données du module.

altitude→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

altitude→wait_async(callback, context)

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

YAltitude.FindAltitude() yFindAltitude()

YAltitude

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

js	function yFindAltitude (func)
cpp	YAltitude* yFindAltitude (string func)
m	+(YAltitude*) FindAltitude : (NSString*) func
pas	function yFindAltitude (func : string): TYAltitude
vb	function yFindAltitude (ByVal func As String) As YAltitude
cs	YAltitude FindAltitude (string func)
java	YAltitude FindAltitude (String func)
uwp	YAltitude FindAltitude (string func)
py	def FindAltitude (func)
php	function yFindAltitude (\$func)
es	function FindAltitude (func)

L'identifiant peut être spécifié sous plusieurs formes:

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

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

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 l'application.

Paramètres :

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

Retourne :

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

YAltitude.FindAltitudeInContext() yFindAltitudeInContext()

YAltitude

Permet de retrouver un altimetre d'après un identifiant donné dans un Context YAPI.

java	YAltitude FindAltitudeInContext (YAPIContext yctx , String func)
uwp	YAltitude FindAltitudeInContext (YAPIContext yctx , string func)
es	function FindAltitudeInContext (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 l'altimètre soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YAltitude.isOnline()` pour tester si l'altimètre est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

yctx un contexte YAPI

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

Retourne :

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

YAltitude.FirstAltitude() yFirstAltitude()

YAltitude

Commence l'énumération des altimètres accessibles par la librairie.

js	function yFirstAltitude ()
cpp	YAltitude* yFirstAltitude ()
m	+(YAltitude*) FirstAltitude
pas	function yFirstAltitude (): TYAltitude
vb	function yFirstAltitude () As YAltitude
cs	YAltitude FirstAltitude ()
java	YAltitude FirstAltitude ()
uwp	YAltitude FirstAltitude ()
py	def FirstAltitude ()
php	function yFirstAltitude ()
es	function FirstAltitude ()

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

Retourne :

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

YAltitude.FirstAltitudeInContext() yFirstAltitudeInContext()

YAltitude

Commence l'énumération des altimètres accessibles par la librairie.

java	<code>YAltitude FirstAltitudeInContext(YAPIContext yctx)</code>
uwp	<code>YAltitude FirstAltitudeInContext(YAPIContext yctx)</code>
es	<code>function FirstAltitudeInContext(yctx)</code>

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

Paramètres :

`yctx` un contexte YAPI.

Retourne :

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

altitude→calibrateFromPoints()**YAltitude**

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 function calibrateFromPoints( rawValues: TDoubleArray,
                                  refValues: TDoubleArray): LongInt
vb  procedure calibrateFromPoints( )
cs  int calibrateFromPoints( List<double> rawValues,
                             List<double> refValues)
java int calibrateFromPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)
uwp Task<int> calibrateFromPoints( List<double> rawValues,
                                   List<double> refValues)
py  def calibrateFromPoints( rawValues, refValues)
php function calibrateFromPoints( $rawValues, $refValues)
es  function calibrateFromPoints( rawValues, refValues)
cmd YAltitude target calibrateFromPoints rawValues refValues

```

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

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

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→clearCache()**YAltitude**

Invalide le cache.

js	function clearCache ()
cpp	void clearCache ()
m	-(void) clearCache
pas	procedure clearCache ()
vb	procedure clearCache ()
cs	void clearCache ()
java	void clearCache ()
py	def clearCache ()
php	function clearCache ()
es	function clearCache ()

Invalide le cache des valeurs courantes de l'altimètre. Force le prochain appel à une méthode `get_xxx()` ou `loadxxx()` pour charger les les données depuis le module.

altitude→describe()**YAltitude**

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

js	function describe ()
cpp	string describe ()
m	-(NSString*) describe
pas	function describe (): string
vb	function describe () As String
cs	string describe ()
java	String describe ()
py	def describe ()
php	function describe ()
es	function describe ()

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYL01-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant l'altimètre (ex:
`Relay(MyCustomName.relay1)=RELAYL01-123456.relay1`)

altitude→get_advMode()**YAltitude****altitude→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	function get_advMode (): Integer
vb	function get_advMode () As Integer
cs	int get_advMode ()
java	int get_advMode ()
uwp	Task<int> get_advMode ()
py	def get_advMode ()
php	function get_advMode ()
es	function get_advMode ()
cmd	YAltitude target get_advMode

Retourne :

une valeur parmi Y_ADVMODE_IMMEDIATE, Y_ADVMODE_PERIOD_AVG, Y_ADVMODE_PERIOD_MIN et Y_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 Y_ADVMODE_INVALID.

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

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

js	function get_advertisedValue ()
cpp	string get_advertisedValue ()
m	-(NSString*) advertisedValue
pas	function get_advertisedValue (): string
vb	function get_advertisedValue () As String
cs	string get_advertisedValue ()
java	String get_advertisedValue ()
uwp	Task<string> get_advertisedValue ()
py	def get_advertisedValue ()
php	function get_advertisedValue ()
es	function get_advertisedValue ()
cmd	YAltitude target get_advertisedValue

Retourne :

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

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

altitude→get_currentRawValue()**YAltitude****altitude→currentRawValue()**

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

js	function get_currentRawValue ()
cpp	double get_currentRawValue ()
m	-(double) currentRawValue
pas	function get_currentRawValue (): double
vb	function get_currentRawValue () As Double
cs	double get_currentRawValue ()
java	double get_currentRawValue ()
uwp	Task<double> get_currentRawValue ()
py	def get_currentRawValue ()
php	function get_currentRawValue ()
es	function get_currentRawValue ()
cmd	YAltitude target get_currentRawValue

Retourne :

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

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

altitude→get_currentValue()**YAltitude****altitude→currentValue()**

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

js	function get_currentValue ()
cpp	double get_currentValue ()
m	-(double) currentValue
pas	function get_currentValue (): double
vb	function get_currentValue () As Double
cs	double get_currentValue ()
java	double get_currentValue ()
uwp	Task<double> get_currentValue ()
py	def get_currentValue ()
php	function get_currentValue ()
es	function get_currentValue ()
cmd	YAltitude target get_currentValue

Retourne :

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

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

altitude→get_dataLogger()**YAltitude****altitude→dataLogger()**

Retourne l'objet YDataLogger du module qui héberge le senseur.

js	function get_dataLogger ()
cpp	YDataLogger* get_dataLogger ()
m	-(YDataLogger*) dataLogger
pas	function get_dataLogger (): TYDataLogger
vb	function get_dataLogger () As YDataLogger
cs	YDataLogger get_dataLogger ()
java	YDataLogger get_dataLogger ()
uwp	Task<YDataLogger> get_dataLogger ()
py	def get_dataLogger ()
php	function get_dataLogger ()
es	function get_dataLogger ()

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :

un objet de classe YDataLogger ou null en cas d'erreur.

altitude→get_errorMessage()**YAltitude****altitude→errorMessage()**

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

js	function get_errorMessage ()
cpp	string get_errorMessage ()
m	-(NSString*) errorMessage
pas	function get_errorMessage (): string
vb	function get_errorMessage () As String
cs	string get_errorMessage ()
java	String get_errorMessage ()
py	def get_errorMessage ()
php	function get_errorMessage ()
es	function get_errorMessage ()

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

Retourne :

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

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

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

js	function get_errorType ()
cpp	YRETCODE get_errorType ()
pas	function get_errorType (): YRETCODE
vb	function get_errorType () As YRETCODE
cs	YRETCODE get_errorType ()
java	int get_errorType ()
py	def get_errorType ()
php	function get_errorType ()
es	function get_errorType ()

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

Retourne :

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

altitude→get_friendlyName()**YAltitude****altitude→friendlyName()**

Retourne un identifiant global de l'altimètre 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	def get_friendlyName ()
php	function get_friendlyName ()
es	function get_friendlyName ()

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

Retourne :

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

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

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

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	function get_functionDescriptor (): YFUN_DESCR
vb	function get_functionDescriptor () As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor ()
java	String get_functionDescriptor ()
py	def get_functionDescriptor ()
php	function get_functionDescriptor ()
es	function get_functionDescriptor ()

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

Retourne :

un identifiant de type YFUN_DESCR.

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

altitude→get_functionId()**YAltitude****altitude→functionId()**

Retourne l'identifiant matériel de l'altimètre, 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	def get_functionId ()
php	function get_functionId ()
es	function get_functionId ()

Par exemple `re_lay1`.

Retourne :

une chaîne de caractères identifiant l'altimètre (ex: `re_lay1`)

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

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

Retourne l'identifiant matériel unique de l'altimètre 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	def get_hardwareId ()
php	function get_hardwareId ()
es	function get_hardwareId ()

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'altimètre (par exemple RELAYL01-123456 . r e l a y 1).

Retourne :

une chaîne de caractères identifiant l'altimètre (ex: RELAYL01-123456 . r e l a y 1)

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

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

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

js	function get_highestValue ()
cpp	double get_highestValue ()
m	-(double) highestValue
pas	function get_highestValue (): double
vb	function get_highestValue () As Double
cs	double get_highestValue ()
java	double get_highestValue ()
uwp	Task<double> get_highestValue ()
py	def get_highestValue ()
php	function get_highestValue ()
es	function get_highestValue ()
cmd	YAltitude target get_highestValue

Peut être réinitialisé à une valeur arbitraire grâce à `set_highestValue()`.

Retourne :

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

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

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

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	function get_logFrequency (): string
vb	function get_logFrequency () As String
cs	string get_logFrequency ()
java	String get_logFrequency ()
uwp	Task<string> get_logFrequency ()
py	def get_logFrequency ()
php	function get_logFrequency ()
es	function get_logFrequency ()
cmd	YAltitude target get_logFrequency

Retourne :

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

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

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

Retourne le nom logique de l'altimètre.

js	function get_logicalName ()
cpp	string get_logicalName ()
m	-(NSString*) logicalName
pas	function get_logicalName (): string
vb	function get_logicalName () As String
cs	string get_logicalName ()
java	String get_logicalName ()
uwp	Task<string> get_logicalName ()
py	def get_logicalName ()
php	function get_logicalName ()
es	function get_logicalName ()
cmd	YAltitude target get_logicalName

Retourne :

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

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

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

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

js	function get_lowestValue ()
cpp	double get_lowestValue ()
m	-(double) lowestValue
pas	function get_lowestValue (): double
vb	function get_lowestValue () As Double
cs	double get_lowestValue ()
java	double get_lowestValue ()
uwp	Task<double> get_lowestValue ()
py	def get_lowestValue ()
php	function get_lowestValue ()
es	function get_lowestValue ()
cmd	YAltitude target get_lowestValue

Peut être réinitialisé à une valeur arbitraire grâce à `set_lowestValue()`.

Retourne :

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

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

altitude→get_module()**YAltitude****altitude→module()**

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

js	function get_module ()
cpp	YModule * get_module ()
m	-(YModule*) module
pas	function get_module (): TYModule
vb	function get_module () As YModule
cs	YModule get_module ()
java	YModule get_module ()
py	def get_module ()
php	function get_module ()
es	function get_module ()

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

Retourne :

une instance de YModule

altitude→get_module_async()**YAltitude****altitude→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.

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

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

js	function get_qnh ()
cpp	double get_qnh ()
m	-(double) qnh
pas	function get_qnh (): double
vb	function get_qnh () As Double
cs	double get_qnh ()
java	double get_qnh ()
uwp	Task<double> get_qnh ()
py	def get_qnh ()
php	function get_qnh ()
es	function get_qnh ()
cmd	YAltitude target get_qnh

Retourne :

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

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

altitude→get_recordedData()**YAltitude****altitude→recordedData()**

Retourne un objet DataSet 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 (s64 startTime , s64 endTime)
m	-(YDataSet*) recordedData : (s64) startTime : (s64) endTime
pas	function get_recordedData (startTime : int64, endTime : int64): TYDataSet
vb	function get_recordedData () As YDataSet
cs	YDataSet get_recordedData (long startTime , long endTime)
java	YDataSet get_recordedData (long startTime , long endTime)
uwp	Task<YDataSet> get_recordedData (long startTime , long endTime)
py	def get_recordedData (startTime , endTime)
php	function get_recordedData (\$ startTime , \$ endTime)
es	function get_recordedData (startTime , endTime)
cmd	YAltitude target get_recordedData startTime endTime

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

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

Paramètres :

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

Retourne :

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

altitude→get_reportFrequency()**YAltitude****altitude→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	function get_reportFrequency (): string
vb	function get_reportFrequency () As String
cs	string get_reportFrequency ()
java	String get_reportFrequency ()
uwp	Task<string> get_reportFrequency ()
py	def get_reportFrequency ()
php	function get_reportFrequency ()
es	function get_reportFrequency ()
cmd	YAltitude target get_reportFrequency

Retourne :

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

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

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

Retourne la résolution des valeurs mesurées.

js	function get_resolution ()
cpp	double get_resolution ()
m	-(double) resolution
pas	function get_resolution (): double
vb	function get_resolution () As Double
cs	double get_resolution ()
java	double get_resolution ()
uwp	Task<double> get_resolution ()
py	def get_resolution ()
php	function get_resolution ()
es	function get_resolution ()
cmd	YAltitude target get_resolution

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

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

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

altitude→get_sensorState()**YAltitude****altitude→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	<code>function get_sensorState()</code>
cpp	<code>int get_sensorState()</code>
m	<code>-(int) sensorState</code>
pas	<code>function get_sensorState(): LongInt</code>
vb	<code>function get_sensorState() As Integer</code>
cs	<code>int get_sensorState()</code>
java	<code>int get_sensorState()</code>
uwp	<code>Task<int> get_sensorState()</code>
py	<code>def get_sensorState()</code>
php	<code>function get_sensorState()</code>
es	<code>function get_sensorState()</code>
cmd	<code>YAltitude target get_sensorState</code>

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 `Y_SENSORSTATE_INVALID`.

altitude→get_technology()**YAltitude****altitude→technology()**

Renvoie la technologie employée par la fonction pour calculer l'altitude.

js	function get_technology ()
cpp	string get_technology ()
m	-(NSString*) technology
pas	function get_technology (): string
vb	function get_technology () As String
cs	string get_technology ()
java	String get_technology ()
uwp	Task<string> get_technology ()
py	def get_technology ()
php	function get_technology ()
es	function get_technology ()
cmd	YAltitude target get_technology

les valeur possible sont "barometric" et "gps"

Retourne :

une chaîne de caractères

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

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

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

js	function get_unit ()
cpp	string get_unit ()
m	-(NSString*) unit
pas	function get_unit (): string
vb	function get_unit () As String
cs	string get_unit ()
java	String get_unit ()
uwp	Task<string> get_unit ()
py	def get_unit ()
php	function get_unit ()
es	function get_unit ()
cmd	YAltitude target get_unit

Retourne :

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

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

altitude→get_userdata()**YAltitude****altitude→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) <code>userData</code>
pas	function get_userdata (): Tobject
vb	function get_userdata () As Object
cs	object get_userdata ()
java	Object get_userdata ()
py	def get_userdata ()
php	function get_userdata ()
es	function get_userdata ()

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

Retourne :

l'objet stocké précédemment par l'appelant.

altitude→isOnline()**YAltitude**

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

js	function isOnline ()
cpp	bool isOnline ()
m	-(BOOL) isOnline
pas	function isOnline (): boolean
vb	function isOnline () As Boolean
cs	bool isOnline ()
java	boolean isOnline ()
py	def isOnline ()
php	function isOnline ()
es	function isOnline ()

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

Retourne :

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

altitude→isOnline_async()**YAltitude**

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

```
js function isOnline_async( callback, context)
```

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

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.

altitude→isSensorReady()**YAltitude**

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

```
cmd YAltitude 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

altitude→load()**YAltitude**

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

js	function load (msValidity)
cpp	YRETCODE load (int msValidity)
m	-(YRETCODE) load : (u64) msValidity
pas	function load (msValidity : u64): YRETCODE
vb	function load (ByVal msValidity As Long) As YRETCODE
cs	YRETCODE load (ulong msValidity)
java	int load (long msValidity)
py	def load (msValidity)
php	function load (\$msValidity)
es	function 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.

altitude→loadAttribute()**YAltitude**

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	function loadAttribute(attrName: string): string
vb	function loadAttribute() As String
cs	string loadAttribute(string attrName)
java	String loadAttribute(String attrName)
uwp	Task<string> loadAttribute(string attrName)
py	def loadAttribute(attrName)
php	function loadAttribute(\$attrName)
es	function loadAttribute(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.

altitude→loadCalibrationPoints()**YAltitude**

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 function loadCalibrationPoints( var rawValues: TDoubleArray,
                                   var refValues: TDoubleArray): LongInt
vb  procedure loadCalibrationPoints( )
cs  int loadCalibrationPoints( List<double> rawValues,
                             List<double> refValues)
java int loadCalibrationPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)
uwp Task<int> loadCalibrationPoints( List<double> rawValues,
                                   List<double> refValues)
py  def loadCalibrationPoints( rawValues, refValues)
php function loadCalibrationPoints( &$rawValues, &$refValues)
es  function loadCalibrationPoints( rawValues, refValues)
cmd YAltitude target loadCalibrationPoints rawValues refValues

```

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→load_async()**YAltitude**

Met en cache les valeurs courantes de l'altimètre, 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 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.

altitude→muteValueCallbacks()**YAltitude**

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

js	function muteValueCallbacks ()
cpp	int muteValueCallbacks ()
m	-(int) muteValueCallbacks
pas	function muteValueCallbacks (): LongInt
vb	function muteValueCallbacks () As Integer
cs	int muteValueCallbacks ()
java	int muteValueCallbacks ()
uwp	Task<int> muteValueCallbacks ()
py	def muteValueCallbacks ()
php	function muteValueCallbacks ()
es	function muteValueCallbacks ()
cmd	YAltitude 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 déclenchement 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.

altitude→nextAltitude()**YAltitude**

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

js	function nextAltitude ()
cpp	YAltitude * nextAltitude ()
m	-(YAltitude *) nextAltitude
pas	function nextAltitude (): TYAltitude
vb	function nextAltitude () As YAltitude
cs	YAltitude nextAltitude ()
java	YAltitude nextAltitude ()
uwp	YAltitude nextAltitude ()
py	def nextAltitude ()
php	function nextAltitude ()
es	function nextAltitude ()

Retourne :

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

altitude→registerTimedReportCallback()**YAltitude**

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

js	function registerTimedReportCallback (callback)
cpp	int registerTimedReportCallback (YAltitudeTimedReportCallback callback)
m	-(int) registerTimedReportCallback : (YAltitudeTimedReportCallback) callback
pas	function registerTimedReportCallback (callback : TYAltitudeTimedReportCallback): LongInt
vb	function registerTimedReportCallback () As Integer
cs	int registerTimedReportCallback (TimedReportCallback callback)
java	int registerTimedReportCallback (TimedReportCallback callback)
uwp	Task<int> registerTimedReportCallback (TimedReportCallback callback)
py	def registerTimedReportCallback (callback)
php	function registerTimedReportCallback (\$callback)
es	function 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'objet fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

altitude→registerValueCallback()**YAltitude**

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

js	function registerValueCallback (callback)
cpp	int registerValueCallback (YAltitudeValueCallback callback)
m	-(int) registerValueCallback : (YAltitudeValueCallback) callback
pas	function registerValueCallback (callback : TYAltitudeValueCallback): LongInt
vb	function registerValueCallback () As Integer
cs	int registerValueCallback (ValueCallback callback)
java	int registerValueCallback (UpdateCallback callback)
uwp	Task<int> registerValueCallback (ValueCallback callback)
py	def registerValueCallback (callback)
php	function registerValueCallback (\$callback)
es	function 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.

altitude→set_advMode()**YAltitude****altitude→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	function 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	Task<int> set_advMode (int newval)
py	def set_advMode (newval)
php	function set_advMode (\$newval)
es	function set_advMode (newval)
cmd	YAltitude target set_advMode newval

Paramètres :

newval une valeur parmi Y_ADVMODE_IMMEDIATE, Y_ADVMODE_PERIOD_AVG, Y_ADVMODE_PERIOD_MIN et Y_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.

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

Modifie l'altitude actuelle supposée.

js	function set_currentValue (newval)
cpp	int set_currentValue (double newval)
m	-(int) setCurrentValue : (double) newval
pas	function set_currentValue (newval : double): integer
vb	function set_currentValue (ByVal newval As Double) As Integer
cs	int set_currentValue (double newval)
java	int set_currentValue (double newval)
uwp	Task<int> set_currentValue (double newval)
py	def set_currentValue (newval)
php	function set_currentValue (\$newval)
es	function set_currentValue (newval)
cmd	YAltitude target set_currentValue newval

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

Paramètres :

newval une valeur numérique représentant l'altitude actuelle supposée

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→set_highestValue()**YAltitude****altitude→setHighestValue()**

Modifie la mémoire de valeur maximale observée.

js	function set_highestValue (newval)
cpp	int set_highestValue (double newval)
m	-(int) setHighestValue : (double) newval
pas	function 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	Task<int> set_highestValue (double newval)
py	def set_highestValue (newval)
php	function set_highestValue (\$newval)
es	function set_highestValue (newval)
cmd	YAltitude 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.

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

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

js	function set_logFrequency (newval)
cpp	int set_logFrequency (const string& newval)
m	-(int) setLogFrequency : (NSString*) newval
pas	function 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	Task<int> set_logFrequency (string newval)
py	def set_logFrequency (newval)
php	function set_logFrequency (\$ newval)
es	function set_logFrequency (newval)
cmd	YAltitude target set_logFrequency newval

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

Modifie le nom logique de l'altimètre.

js	function set_logicalName (newval)
cpp	int set_logicalName (const string& newval)
m	-(int) setLogicalName : (NSString*) newval
pas	function set_logicalName (newval : string): integer
vb	function set_logicalName (ByVal newval As String) As Integer
cs	int set_logicalName (string newval)
java	int set_logicalName (String newval)
uwp	Task<int> set_logicalName (string newval)
py	def set_logicalName (newval)
php	function set_logicalName (\$ newval)
es	function set_logicalName (newval)
cmd	YAltitude target set_logicalName newval

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

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

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

Modifie la mémoire de valeur minimale observée.

js	function set_lowestValue (newval)
cpp	int set_lowestValue (double newval)
m	-(int) setLowestValue : (double) newval
pas	function 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	Task<int> set_lowestValue (double newval)
py	def set_lowestValue (newval)
php	function set_lowestValue (\$newval)
es	function set_lowestValue (newval)
cmd	YAltitude 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.

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

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

js	function set_qnh (newval)
cpp	int set_qnh (double newval)
m	-(int) setQnh : (double) newval
pas	function set_qnh (newval : double): integer
vb	function set_qnh (ByVal newval As Double) As Integer
cs	int set_qnh (double newval)
java	int set_qnh (double newval)
uwp	Task<int> set_qnh (double newval)
py	def set_qnh (newval)
php	function set_qnh (\$newval)
es	function set_qnh (newval)
cmd	YAltitude target set_qnh newval

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→set_reportFrequency()**YAltitude****altitude→setReportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

js	function set_reportFrequency (newval)
cpp	int set_reportFrequency (const string& newval)
m	-(int) setReportFrequency : (NSString*) newval
pas	function 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	Task<int> set_reportFrequency (string newval)
py	def set_reportFrequency (newval)
php	function set_reportFrequency (\$ newval)
es	function set_reportFrequency (newval)
cmd	YAltitude target set_reportFrequency newval

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

Modifie la résolution des valeurs physique mesurées.

js	function set_resolution (newval)
cpp	int set_resolution (double newval)
m	-(int) setResolution : (double) newval
pas	function 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	Task<int> set_resolution (double newval)
py	def set_resolution (newval)
php	function set_resolution (\$newval)
es	function set_resolution (newval)
cmd	YAltitude target set_resolution newval

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

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

altitude→set_userdata()**YAltitude****altitude→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	procedure set_userdata (data : Tobject)
vb	procedure set_userdata (ByVal data As Object)
cs	void set_userdata (object data)
java	void set_userdata (Object data)
py	def set_userdata (data)
php	function set_userdata (\$data)
es	function 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

altitude→startDataLogger()**YAltitude**

Démarre l'enregistreur de données du module.

js	function startDataLogger ()
cpp	int startDataLogger ()
m	-(int) startDataLogger
pas	function startDataLogger (): LongInt
vb	function startDataLogger () As Integer
cs	int startDataLogger ()
java	int startDataLogger ()
uwp	Task<int> startDataLogger ()
py	def startDataLogger ()
php	function startDataLogger ()
es	function startDataLogger ()
cmd	YAltitude 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.

altitude→stopDataLogger()**YAltitude**

Arrête l'enregistreur de données du module.

js	function stopDataLogger ()
cpp	int stopDataLogger ()
m	-(int) stopDataLogger
pas	function stopDataLogger (): LongInt
vb	function stopDataLogger () As Integer
cs	int stopDataLogger ()
java	int stopDataLogger ()
uwp	Task<int> stopDataLogger ()
py	def stopDataLogger ()
php	function stopDataLogger ()
es	function stopDataLogger ()
cmd	YAltitude target stopDataLogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

altitude→unmuteValueCallbacks()**YAltitude**

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

js	function unmuteValueCallbacks ()
cpp	int unmuteValueCallbacks ()
m	-(int) unmuteValueCallbacks
pas	function unmuteValueCallbacks (): LongInt
vb	function unmuteValueCallbacks () As Integer
cs	int unmuteValueCallbacks ()
java	int unmuteValueCallbacks ()
uwp	Task<int> unmuteValueCallbacks ()
py	def unmuteValueCallbacks ()
php	function unmuteValueCallbacks ()
es	function unmuteValueCallbacks ()
cmd	YAltitude 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.

altitude→wait_async()**YAltitude**

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)
```

```
es function 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.

22.4. Interface de la fonction Pressure

La classe YPressure permet de lire et de configurer les capteurs de pression Yoctopuce. Elle hérite de la class 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_pressure.js"></script> in node.js: require('yoctolib-es2017/yocto_pressure.js');
js	<script type='text/javascript' src='yocto_pressure.js'></script>
cpp	#include "yocto_pressure.h"
m	#import "yocto_pressure.h"
pas	uses yocto_pressure;
vb	yocto_pressure.vb
cs	yocto_pressure.cs
java	import com.yoctopuce.YoctoAPI.YPressure;
uwp	import com.yoctopuce.YoctoAPI.YPressure;
py	from yocto_pressure import *
php	require_once('yocto_pressure.php');

Fonction globales

yFindPressure(func)

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

yFindPressureInContext(yctx, func)

Permet de retrouver un capteur de pression d'après un identifiant donné dans un Context YAPI.

yFirstPressure()

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

yFirstPressureInContext(yctx)

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

Méthodes des objets YPressure

pressure→calibrateFromPoints(rawValues, refValues)

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

pressure→clearCache()

Invalide le cache.

pressure→describe()

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

pressure→get_advMode()

Retourne le mode de calcul de la valeur publiée jusqu'au hub parent (advertisedValue).

pressure→get_advertisedValue()

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

pressure→get_currentRawValue()

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

pressure→get_currentValue()

Retourne la valeur actuelle de la pression, en millibar (hPa), sous forme de nombre à virgule.

pressure→get_dataLogger()

Retourne l'objet YDataLogger du module qui héberge le senseur.

pressure→get_errorMessage()

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

pressure→get_errorType()

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

pressure→get_friendlyName()

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

pressure→get_functionDescriptor()

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

pressure→get_functionId()

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

pressure→get_hardwareId()

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

pressure→get_highestValue()

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

pressure→get_logFrequency()

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

pressure→get_logicalName()

Retourne le nom logique du capteur de pression.

pressure→get_lowestValue()

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

pressure→get_module()

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

pressure→get_module_async(callback, context)

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

pressure→get_recordedData(startTime, endTime)

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

pressure→get_reportFrequency()

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

pressure→get_resolution()

Retourne la résolution des valeurs mesurées.

pressure→get_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.

pressure→get_unit()

Retourne l'unité dans laquelle la pression est exprimée.

pressure→get_userData()

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

pressure→isOnline()

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

pressure→isOnline_async(callback, context)

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

pressure→isSensorReady()

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

pressure→load(msValidity)

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

pressure→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.

pressure→loadCalibrationPoints(rawValues, refValues)

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

pressure→load_async(msValidity, callback, context)

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

pressure→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

pressure→nextPressure()

Continue l'énumération des capteurs de pression commencée à l'aide de `yFirstPressure()`.

pressure→registerTimedReportCallback(callback)

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

pressure→registerValueCallback(callback)

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

pressure→set_advMode(newval)

Modifie le mode de calcul de la valeur publiée jusqu'au hub parent (`advertisedValue`).

pressure→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

pressure→set_logFrequency(newval)

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

pressure→set_logicalName(newval)

Modifie le nom logique du capteur de pression.

pressure→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

pressure→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

pressure→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

pressure→set_userData(data)

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

pressure→startDataLogger()

Démarre l'enregistreur de données du module.

pressure→stopDataLogger()

Arrête l'enregistreur de données du module.

pressure→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

pressure→wait_async(callback, context)

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

YPressure.FindPressure() yFindPressure()

YPressure

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

js	function yFindPressure (func)
cpp	YPressure* yFindPressure (string func)
m	+(YPressure*) FindPressure : (NSString*) func
pas	function yFindPressure (func : string): TYPressure
vb	function yFindPressure (ByVal func As String) As YPressure
cs	YPressure FindPressure (string func)
java	YPressure FindPressure (String func)
uwp	YPressure FindPressure (string func)
py	def FindPressure (func)
php	function yFindPressure (\$func)
es	function FindPressure (func)

L'identifiant peut être spécifié sous plusieurs formes:

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

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

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 l'application.

Paramètres :

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

Retourne :

un objet de classe `YPressure` qui permet ensuite de contrôler le capteur de pression.

YPressure.FindPressureInContext() yFindPressureInContext()

YPressure

Permet de retrouver un capteur de pression d'après un identifiant donné dans un Context YAPI.

java	YPressure FindPressureInContext (YAPIContext yctx , String func)
uwp	YPressure FindPressureInContext (YAPIContext yctx , string func)
es	function FindPressureInContext (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 pression soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YPressure.isOnline()` pour tester si le capteur de pression est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

yctx un contexte YAPI

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

Retourne :

un objet de classe `YPressure` qui permet ensuite de contrôler le capteur de pression.

YPressure.FirstPressure() yFirstPressure()

YPressure

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

js	function yFirstPressure ()
cpp	YPressure* yFirstPressure ()
m	+(YPressure*) FirstPressure
pas	function yFirstPressure (): TYPpressure
vb	function yFirstPressure () As YPressure
cs	YPressure FirstPressure ()
java	YPressure FirstPressure ()
uwp	YPressure FirstPressure ()
py	def FirstPressure ()
php	function yFirstPressure ()
es	function FirstPressure ()

Utiliser la fonction `YPressure.nextPressure( )` pour itérer sur les autres capteurs de pression.

Retourne :

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

YPressure.FirstPressureInContext() yFirstPressureInContext()

YPressure

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

java	YPressure FirstPressureInContext (YAPIContext yctx)
uwp	YPressure FirstPressureInContext (YAPIContext yctx)
es	function FirstPressureInContext (yctx)

Utiliser la fonction YPressure.nextPressure() pour itérer sur les autres capteurs de pression.

Paramètres :

yctx un contexte YAPI.

Retourne :

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

pressure→calibrateFromPoints()**YPressure**

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 function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt
vb procedure calibrateFromPoints( )
cs int calibrateFromPoints( List<double> rawValues,
                             List<double> refValues)
java int calibrateFromPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)
uwp Task<int> calibrateFromPoints( List<double> rawValues,
                                   List<double> refValues)
py def calibrateFromPoints( rawValues, refValues)
php function calibrateFromPoints( $rawValues, $refValues)
es function calibrateFromPoints( rawValues, refValues)
cmd YPressure target calibrateFromPoints rawValues refValues

```

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

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

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

pressure→clearCache()**YPressure**

Invalide le cache.

js	function clearCache ()
cpp	void clearCache ()
m	-(void) clearCache
pas	procedure clearCache ()
vb	procedure clearCache ()
cs	void clearCache ()
java	void clearCache ()
py	def clearCache ()
php	function clearCache ()
es	function clearCache ()

Invalide le cache des valeurs courantes du capteur de pression. Force le prochain appel à une méthode `get_xxx()` ou `loadxxx()` pour charger les les données depuis le module.

pressure→describe()**YPressure**

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

js	function describe ()
cpp	string describe ()
m	-(NSString*) describe
pas	function describe (): string
vb	function describe () As String
cs	string describe ()
java	String describe ()
py	def describe ()
php	function describe ()
es	function describe ()

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner Relay(MyCustomName.relay1)=RELAYL01-123456.relay1 si le module est déjà connecté ou Relay(BadCustomName.relay1)=unresolved si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le capteur de pression (ex:
Relay(MyCustomName.relay1)=RELAYL01-123456.relay1)

pressure→**get_advMode()****YPressure****pressure**→**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	function get_advMode (): Integer
vb	function get_advMode () As Integer
cs	int get_advMode ()
java	int get_advMode ()
uwp	Task<int> get_advMode ()
py	def get_advMode ()
php	function get_advMode ()
es	function get_advMode ()
cmd	YPressure target get_advMode

Retourne :

une valeur parmi Y_ADVMODE_IMMEDIATE, Y_ADVMODE_PERIOD_AVG, Y_ADVMODE_PERIOD_MIN et Y_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 Y_ADVMODE_INVALID.

pressure→get_advertisedValue()**YPressure****pressure→advertisedValue()**

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

js	function get_advertisedValue ()
cpp	string get_advertisedValue ()
m	-(NSString*) advertisedValue
pas	function get_advertisedValue (): string
vb	function get_advertisedValue () As String
cs	string get_advertisedValue ()
java	String get_advertisedValue ()
uwp	Task<string> get_advertisedValue ()
py	def get_advertisedValue ()
php	function get_advertisedValue ()
es	function get_advertisedValue ()
cmd	YPressure target get_advertisedValue

Retourne :

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

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

pressure→**get_currentRawValue()****YPressure****pressure**→**currentRawValue()**

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

js	function get_currentRawValue ()
cpp	double get_currentRawValue ()
m	-(double) currentRawValue
pas	function get_currentRawValue (): double
vb	function get_currentRawValue () As Double
cs	double get_currentRawValue ()
java	double get_currentRawValue ()
uwp	Task<double> get_currentRawValue ()
py	def get_currentRawValue ()
php	function get_currentRawValue ()
es	function get_currentRawValue ()
cmd	YPressure target get_currentRawValue

Retourne :

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

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

pressure→get_currentValue()**YPressure****pressure→currentValue()**

Retourne la valeur actuelle de la pression, en millibar (hPa), sous forme de nombre à virgule.

js	function get_currentValue ()
cpp	double get_currentValue ()
m	-(double) currentValue
pas	function get_currentValue (): double
vb	function get_currentValue () As Double
cs	double get_currentValue ()
java	double get_currentValue ()
uwp	Task<double> get_currentValue ()
py	def get_currentValue ()
php	function get_currentValue ()
es	function get_currentValue ()
cmd	YPressure target get_currentValue

Retourne :

une valeur numérique représentant la valeur actuelle de la pression, en millibar (hPa), sous forme de nombre à virgule

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

pressure→**get_dataLogger()****YPressure****pressure**→**dataLogger()**

Retourne l'objet YDataLogger du module qui héberge le senseur.

js	function get_dataLogger ()
cpp	YDataLogger* get_dataLogger ()
m	-(YDataLogger*) dataLogger
pas	function get_dataLogger (): TYDataLogger
vb	function get_dataLogger () As YDataLogger
cs	YDataLogger get_dataLogger ()
java	YDataLogger get_dataLogger ()
uwp	Task<YDataLogger> get_dataLogger ()
py	def get_dataLogger ()
php	function get_dataLogger ()
es	function get_dataLogger ()

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :

un objet de classe YDataLogger ou null en cas d'erreur.

pressure→get_errorMessage()**YPressure****pressure→errorMessage()**

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

js	function get_errorMessage ()
cpp	string get_errorMessage ()
m	-(NSString*) errorMessage
pas	function get_errorMessage (): string
vb	function get_errorMessage () As String
cs	string get_errorMessage ()
java	String get_errorMessage ()
py	def get_errorMessage ()
php	function get_errorMessage ()
es	function get_errorMessage ()

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

Retourne :

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

pressure→get_errorType()**YPressure****pressure→errorType()**

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

js	function get_errorType ()
cpp	YRETCODE get_errorType ()
pas	function get_errorType (): YRETCODE
vb	function get_errorType () As YRETCODE
cs	YRETCODE get_errorType ()
java	int get_errorType ()
py	def get_errorType ()
php	function get_errorType ()
es	function get_errorType ()

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

Retourne :

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

pressure→get_friendlyName()**YPressure****pressure→friendlyName()**

Retourne un identifiant global du capteur de pression 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	def get_friendlyName ()
php	function get_friendlyName ()
es	function get_friendlyName ()

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

Retourne :

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

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

pressure→**get_functionDescriptor()****YPressure****pressure**→**functionDescriptor()**

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	function get_functionDescriptor (): YFUN_DESCR
vb	function get_functionDescriptor () As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor ()
java	String get_functionDescriptor ()
py	def get_functionDescriptor ()
php	function get_functionDescriptor ()
es	function get_functionDescriptor ()

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

Retourne :

un identifiant de type YFUN_DESCR.

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

pressure→get_functionId()**YPressure****pressure→functionId()**

Retourne l'identifiant matériel du capteur de pression, 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	def get_functionId ()
php	function get_functionId ()
es	function get_functionId ()

Par exemple `re_lay1`.

Retourne :

une chaîne de caractères identifiant le capteur de pression (ex: `re_lay1`)

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

pressure→**get_hardwareId()****YPressure****pressure**→**hardwareId()**

Retourne l'identifiant matériel unique du capteur de pression 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	def get_hardwareId ()
php	function get_hardwareId ()
es	function get_hardwareId ()

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de pression (par exemple RELAYL01-123456 . r e l a y 1).

Retourne :

une chaîne de caractères identifiant le capteur de pression (ex: RELAYL01-123456 . r e l a y 1)

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

pressure→get_highestValue()**YPressure****pressure→highestValue()**

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

js	function get_highestValue ()
cpp	double get_highestValue ()
m	-(double) highestValue
pas	function get_highestValue (): double
vb	function get_highestValue () As Double
cs	double get_highestValue ()
java	double get_highestValue ()
uwp	Task<double> get_highestValue ()
py	def get_highestValue ()
php	function get_highestValue ()
es	function get_highestValue ()
cmd	YPressure 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 pression depuis le démarrage du module

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

pressure→get_logFrequency()**YPressure****pressure→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	function get_logFrequency (): string
vb	function get_logFrequency () As String
cs	string get_logFrequency ()
java	String get_logFrequency ()
uwp	Task<string> get_logFrequency ()
py	def get_logFrequency ()
php	function get_logFrequency ()
es	function get_logFrequency ()
cmd	YPressure target get_logFrequency

Retourne :

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

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

pressure→get_logicalName()**YPressure****pressure→logicalName()**

Retourne le nom logique du capteur de pression.

js	function get_logicalName ()
cpp	string get_logicalName ()
m	-(NSString*) logicalName
pas	function get_logicalName (): string
vb	function get_logicalName () As String
cs	string get_logicalName ()
java	String get_logicalName ()
uwp	Task<string> get_logicalName ()
py	def get_logicalName ()
php	function get_logicalName ()
es	function get_logicalName ()
cmd	YPressure target get_logicalName

Retourne :

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

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

pressure→**get_lowestValue()****YPressure****pressure**→**lowestValue()**

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

js	function get_lowestValue ()
cpp	double get_lowestValue ()
m	-(double) lowestValue
pas	function get_lowestValue (): double
vb	function get_lowestValue () As Double
cs	double get_lowestValue ()
java	double get_lowestValue ()
uwp	Task<double> get_lowestValue ()
py	def get_lowestValue ()
php	function get_lowestValue ()
es	function get_lowestValue ()
cmd	YPressure 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 pression depuis le démarrage du module

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

pressure→get_module()**YPressure****pressure→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	function get_module (): TYModule
vb	function get_module () As YModule
cs	YModule get_module ()
java	YModule get_module ()
py	def get_module ()
php	function get_module ()
es	function get_module ()

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

Retourne :

une instance de YModule

pressure→get_module_async()**YPressure****pressure→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.

pressure→get_recordedData()**YPressure****pressure→recordedData()**

Retourne un objet DataSet 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 (s64 startTime , s64 endTime)
m	-(YDataSet*) recordedData : (s64) startTime : (s64) endTime
pas	function get_recordedData (startTime : int64, endTime : int64): TYDataSet
vb	function get_recordedData () As YDataSet
cs	YDataSet get_recordedData (long startTime , long endTime)
java	YDataSet get_recordedData (long startTime , long endTime)
uwp	Task<YDataSet> get_recordedData (long startTime , long endTime)
py	def get_recordedData (startTime , endTime)
php	function get_recordedData (\$ startTime , \$ endTime)
es	function get_recordedData (startTime , endTime)
cmd	YPressure target get_recordedData startTime endTime

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

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

Paramètres :

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

Retourne :

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

pressure→get_reportFrequency()**YPressure****pressure→reportFrequency()**

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	function get_reportFrequency (): string
vb	function get_reportFrequency () As String
cs	string get_reportFrequency ()
java	String get_reportFrequency ()
uwp	Task<string> get_reportFrequency ()
py	def get_reportFrequency ()
php	function get_reportFrequency ()
es	function get_reportFrequency ()
cmd	YPressure target get_reportFrequency

Retourne :

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

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

pressure→get_resolution()**YPressure****pressure→resolution()**

Retourne la résolution des valeurs mesurées.

js	function get_resolution ()
cpp	double get_resolution ()
m	-(double) resolution
pas	function get_resolution (): double
vb	function get_resolution () As Double
cs	double get_resolution ()
java	double get_resolution ()
uwp	Task<double> get_resolution ()
py	def get_resolution ()
php	function get_resolution ()
es	function get_resolution ()
cmd	YPressure target get_resolution

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

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

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

pressure→get_sensorState()**YPressure****pressure→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	function get_sensorState (): LongInt
vb	function get_sensorState () As Integer
cs	int get_sensorState ()
java	int get_sensorState ()
uwp	Task<int> get_sensorState ()
py	def get_sensorState ()
php	function get_sensorState ()
es	function get_sensorState ()
cmd	YPressure 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 Y_SENSORSTATE_INVALID.

pressure→get_unit()**YPressure****pressure→unit()**

Retourne l'unité dans laquelle la pression est exprimée.

js	function get_unit ()
cpp	string get_unit ()
m	-(NSString*) unit
pas	function get_unit (): string
vb	function get_unit () As String
cs	string get_unit ()
java	String get_unit ()
uwp	Task<string> get_unit ()
py	def get_unit ()
php	function get_unit ()
es	function get_unit ()
cmd	YPressure target get_unit

Retourne :

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

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

pressure→**get_userData()****YPressure****pressure**→**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	function get_userData (): Tobject
vb	function get_userData () As Object
cs	object get_userData ()
java	Object get_userData ()
py	def get_userData ()
php	function get_userData ()
es	function get_userData ()

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

Retourne :

l'objet stocké précédemment par l'appelant.

pressure→isOnline()**YPressure**

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

js	function isOnline ()
cpp	bool isOnline ()
m	-(BOOL) isOnline
pas	function isOnline (): boolean
vb	function isOnline () As Boolean
cs	bool isOnline ()
java	boolean isOnline ()
py	def isOnline ()
php	function isOnline ()
es	function isOnline ()

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

Retourne :

`true` si le capteur de pression est joignable, `false` sinon

pressure→isOnline_async()**YPressure**

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

```
js function isOnline_async( callback, context)
```

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

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.

pressure→isSensorReady()**YPressure**

Vérifie si le capteur est actuellement en état de transmettre une mesure valide.

```
cmd YPressure 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

pressure→load()**YPressure**

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

js	function load (msValidity)
cpp	YRETCODE load (int msValidity)
m	-(YRETCODE) load : (u64) msValidity
pas	function load (msValidity : u64): YRETCODE
vb	function load (ByVal msValidity As Long) As YRETCODE
cs	YRETCODE load (ulong msValidity)
java	int load (long msValidity)
py	def load (msValidity)
php	function load (\$msValidity)
es	function 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.

pressure→loadAttribute()**YPressure**

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	function loadAttribute(attrName: string): string
vb	function loadAttribute() As String
cs	string loadAttribute(string attrName)
java	String loadAttribute(String attrName)
uwp	Task<string> loadAttribute(string attrName)
py	def loadAttribute(attrName)
php	function loadAttribute(\$attrName)
es	function loadAttribute(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.

pressure→loadCalibrationPoints()**YPressure**

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 function loadCalibrationPoints( var rawValues: TDoubleArray,
                                   var refValues: TDoubleArray): LongInt
vb  procedure loadCalibrationPoints( )
cs  int loadCalibrationPoints( List<double> rawValues,
                             List<double> refValues)
java int loadCalibrationPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)
uwp Task<int> loadCalibrationPoints( List<double> rawValues,
                                   List<double> refValues)
py  def loadCalibrationPoints( rawValues, refValues)
php function loadCalibrationPoints( &$rawValues, &$refValues)
es  function loadCalibrationPoints( rawValues, refValues)
cmd YPressure target loadCalibrationPoints rawValues refValues

```

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

pressure→load_async()**YPressure**

Met en cache les valeurs courantes du capteur de pression, 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 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.

pressure→muteValueCallbacks()**YPressure**

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

js	function muteValueCallbacks ()
cpp	int muteValueCallbacks ()
m	-(int) muteValueCallbacks
pas	function muteValueCallbacks (): LongInt
vb	function muteValueCallbacks () As Integer
cs	int muteValueCallbacks ()
java	int muteValueCallbacks ()
uwp	Task<int> muteValueCallbacks ()
py	def muteValueCallbacks ()
php	function muteValueCallbacks ()
es	function muteValueCallbacks ()
cmd	YPressure 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 déclenchement 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.

pressure→nextPressure()**YPressure**

Continue l'énumération des capteurs de pression commencée à l'aide de `yFirstPressure()`.

js	<code>function nextPressure()</code>
cpp	<code>YPressure * nextPressure()</code>
m	<code>-(YPressure*) nextPressure</code>
pas	<code>function nextPressure(): TYPressure</code>
vb	<code>function nextPressure() As YPressure</code>
cs	<code>YPressure nextPressure()</code>
java	<code>YPressure nextPressure()</code>
uwp	<code>YPressure nextPressure()</code>
py	<code>def nextPressure()</code>
php	<code>function nextPressure()</code>
es	<code>function nextPressure()</code>

Retourne :

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

pressure→registerTimedReportCallback()**YPressure**

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

js	function registerTimedReportCallback (callback)
cpp	int registerTimedReportCallback (YPressureTimedReportCallback callback)
m	-(int) registerTimedReportCallback : (YPressureTimedReportCallback) callback
pas	function registerTimedReportCallback (callback : TYPressureTimedReportCallback): LongInt
vb	function registerTimedReportCallback () As Integer
cs	int registerTimedReportCallback (TimedReportCallback callback)
java	int registerTimedReportCallback (TimedReportCallback callback)
uwp	Task<int> registerTimedReportCallback (TimedReportCallback callback)
py	def registerTimedReportCallback (callback)
php	function registerTimedReportCallback (\$callback)
es	function 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'objet fonction dont la valeur a changé, et un objet YMeasure décrivant la nouvelle valeur publiée.

pressure→registerValueCallback()**YPressure**

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

js	function registerValueCallback (callback)
cpp	int registerValueCallback (YPressureValueCallback callback)
m	-(int) registerValueCallback : (YPressureValueCallback) callback
pas	function registerValueCallback (callback : TYPPressureValueCallback): LongInt
vb	function registerValueCallback () As Integer
cs	int registerValueCallback (ValueCallback callback)
java	int registerValueCallback (UpdateCallback callback)
uwp	Task<int> registerValueCallback (ValueCallback callback)
py	def registerValueCallback (callback)
php	function registerValueCallback (\$callback)
es	function 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.

pressure→set_advMode()**YPressure****pressure→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	function 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	Task<int> set_advMode (int newval)
py	def set_advMode (newval)
php	function set_advMode (\$newval)
es	function set_advMode (newval)
cmd	YPressure target set_advMode newval

Paramètres :

newval une valeur parmi Y_ADVMODE_IMMEDIATE, Y_ADVMODE_PERIOD_AVG, Y_ADVMODE_PERIOD_MIN et Y_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.

pressure→set_highestValue()**YPressure****pressure→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	function 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	Task<int> set_highestValue (double newval)
py	def set_highestValue (newval)
php	function set_highestValue (\$newval)
es	function set_highestValue (newval)
cmd	YPressure 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.

pressure→set_logFrequency()**YPressure****pressure→setLogFrequency()**

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

js	function set_logFrequency (newval)
cpp	int set_logFrequency (const string& newval)
m	-(int) setLogFrequency : (NSString*) newval
pas	function 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	Task<int> set_logFrequency (string newval)
py	def set_logFrequency (newval)
php	function set_logFrequency (\$ newval)
es	function set_logFrequency (newval)
cmd	YPressure target set_logFrequency newval

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

pressure→set_logicalName()

YPressure

pressure→setLogicalName()

Modifie le nom logique du capteur de pression.

js	function set_logicalName(newval)
cpp	int set_logicalName(const string& newval)
m	-(int) setLogicalName : (NSString*) newval
pas	function set_logicalName(newval: string): integer
vb	function set_logicalName(ByVal newval As String) As Integer
cs	int set_logicalName(string newval)
java	int set_logicalName(String newval)
uwp	Task<int> set_logicalName(string newval)
py	def set_logicalName(newval)
php	function set_logicalName(\$newval)
es	function set_logicalName(newval)
cmd	YPressure target set_logicalName newval

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

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

pressure→set_lowestValue()**YPressure****pressure→setLowestValue()**

Modifie la mémoire de valeur minimale observée.

js	function set_lowestValue (newval)
cpp	int set_lowestValue (double newval)
m	-(int) setLowestValue : (double) newval
pas	function 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	Task<int> set_lowestValue (double newval)
py	def set_lowestValue (newval)
php	function set_lowestValue (\$newval)
es	function set_lowestValue (newval)
cmd	YPressure 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.

pressure→set_reportFrequency()**YPressure****pressure→setReportFrequency()**

Modifie la fréquence de notification périodique des valeurs mesurées.

js	function set_reportFrequency (newval)
cpp	int set_reportFrequency (const string& newval)
m	-(int) setReportFrequency : (NSString*) newval
pas	function 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	Task<int> set_reportFrequency (string newval)
py	def set_reportFrequency (newval)
php	function set_reportFrequency (\$ newval)
es	function set_reportFrequency (newval)
cmd	YPressure target set_reportFrequency newval

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

pressure→set_resolution()

YPressure

pressure→setResolution()

Modifie la résolution des valeurs physique mesurées.

js	function set_resolution (newval)
cpp	int set_resolution (double newval)
m	-(int) setResolution : (double) newval
pas	function 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	Task<int> set_resolution (double newval)
py	def set_resolution (newval)
php	function set_resolution (\$newval)
es	function set_resolution (newval)
cmd	YPressure target set_resolution newval

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

Paramètres :

newval une valeur numérique représentant la résolution des valeurs physique mesurées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

pressure→set_userdata()**YPressure****pressure→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	procedure set_userdata (data : Tobject)
vb	procedure set_userdata (ByVal data As Object)
cs	void set_userdata (object data)
java	void set_userdata (Object data)
py	def set_userdata (data)
php	function set_userdata (\$data)
es	function 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

pressure→startDataLogger()**YPressure**

Démarre l'enregistreur de données du module.

js	function startDataLogger ()
cpp	int startDataLogger ()
m	-(int) startDataLogger
pas	function startDataLogger (): LongInt
vb	function startDataLogger () As Integer
cs	int startDataLogger ()
java	int startDataLogger ()
uwp	Task<int> startDataLogger ()
py	def startDataLogger ()
php	function startDataLogger ()
es	function startDataLogger ()
cmd	YPressure 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.

pressure→stopDataLogger()**YPressure**

Arrête l'enregistreur de données du module.

js	function stopDataLogger ()
cpp	int stopDataLogger ()
m	-(int) stopDataLogger
pas	function stopDataLogger (): LongInt
vb	function stopDataLogger () As Integer
cs	int stopDataLogger ()
java	int stopDataLogger ()
uwp	Task<int> stopDataLogger ()
py	def stopDataLogger ()
php	function stopDataLogger ()
es	function stopDataLogger ()
cmd	YPressure target stopDataLogger

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

pressure→unmuteValueCallbacks()**YPressure**

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

js	function unmuteValueCallbacks ()
cpp	int unmuteValueCallbacks ()
m	-(int) unmuteValueCallbacks
pas	function unmuteValueCallbacks (): LongInt
vb	function unmuteValueCallbacks () As Integer
cs	int unmuteValueCallbacks ()
java	int unmuteValueCallbacks ()
uwp	Task<int> unmuteValueCallbacks ()
py	def unmuteValueCallbacks ()
php	function unmuteValueCallbacks ()
es	function unmuteValueCallbacks ()
cmd	YPressure 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.

pressure→wait_async()**YPressure**

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)
```

```
es function 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.

22.5. Interface de la fonction Temperature

La classe YTemperature permet de lire et de configurer les capteurs de température Yoctopuce. Elle hérite de la class 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:

es	in HTML: <script src="../../lib/yocto_temperature.js"></script> in node.js: require('yoctolib-es2017/yocto_temperature.js');
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');

Fonction globales

yFindTemperature(func)

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

yFindTemperatureInContext(yctx, func)

Permet de retrouver un capteur de température d'après un identifiant donné dans un Context YAPI.

yFirstTemperature()

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

yFirstTemperatureInContext(yctx)

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

Méthodes des objets YTemperature

temperature→calibrateFromPoints(rawValues, refValues)

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

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

temperature→get_reportFrequency()

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

temperature→get_resolution()

Retourne la résolution des valeurs mesurées.

temperature→get_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_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→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().

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)

Change 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() yFindTemperature()

YTemperature

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

js	function yFindTemperature (func)
cpp	YTemperature* yFindTemperature (string func)
m	+(YTemperature*) FindTemperature : (NSString*) func
pas	function yFindTemperature (func : string): TYTemperature
vb	function yFindTemperature (ByVal func As String) As YTemperature
cs	YTemperature FindTemperature (string func)
java	YTemperature FindTemperature (String func)
uwp	YTemperature FindTemperature (string func)
py	def FindTemperature (func)
php	function yFindTemperature (\$func)
es	function FindTemperature (func)

L'identifiant peut être spécifié sous plusieurs formes:

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

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

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 l'application.

Paramètres :

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

Retourne :

un objet de classe `YTemperature` qui permet ensuite de contrôler le capteur de température.

YTemperature.FindTemperatureInContext() yFindTemperatureInContext()

YTemperature

Permet de retrouver un capteur de température d'après un identifiant donné dans un Context YAPI.

```

java YTemperature FindTemperatureInContext( YAPIContext yctx,
                                           String func)
uwp  YTemperature FindTemperatureInContext( YAPIContext yctx,
                                           string func)
es   function 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 où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YTemperature.isOnline()` pour tester si le capteur de température est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

yctx un contexte YAPI

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

Retourne :

un objet de classe `YTemperature` qui permet ensuite de contrôler le capteur de température.

YTemperature.FirstTemperature() yFirstTemperature()

YTemperature

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

js	function yFirstTemperature ()
cpp	YTemperature* yFirstTemperature ()
m	+(YTemperature*) FirstTemperature
pas	function yFirstTemperature (): TYTemperature
vb	function yFirstTemperature () As YTemperature
cs	YTemperature FirstTemperature ()
java	YTemperature FirstTemperature ()
uwp	YTemperature FirstTemperature ()
py	def FirstTemperature ()
php	function yFirstTemperature ()
es	function 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() yFirstTemperatureInContext()

YTemperature

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

java	YTemperature FirstTemperatureInContext (YAPIContext yctx)
uwp	YTemperature FirstTemperatureInContext (YAPIContext yctx)
es	function 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.

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 function calibrateFromPoints( rawValues: TDoubleArray,
                                   refValues: TDoubleArray): LongInt
vb procedure calibrateFromPoints( )
cs int calibrateFromPoints( List<double> rawValues,
                             List<double> refValues)
java int calibrateFromPoints( ArrayList<Double> rawValues,
                              ArrayList<Double> refValues)
uwp Task<int> calibrateFromPoints( List<double> rawValues,
                                   List<double> refValues)
py def calibrateFromPoints( rawValues, refValues)
php function calibrateFromPoints( $rawValues, $refValues)
es function calibrateFromPoints( rawValues, 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	procedure clearCache ()
vb	procedure clearCache ()
cs	void clearCache ()
java	void clearCache ()
py	def clearCache ()
php	function clearCache ()
es	function 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 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	function describe (): string
vb	function describe () As String
cs	string describe ()
java	String describe ()
py	def describe ()
php	function describe ()
es	function describe ()

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYL01-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le capteur de température (ex:
`Relay(MyCustomName.relay1)=RELAYL01-123456.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	function get_advMode (): Integer
vb	function get_advMode () As Integer
cs	int get_advMode ()
java	int get_advMode ()
uwp	Task<int> get_advMode ()
py	def get_advMode ()
php	function get_advMode ()
es	function get_advMode ()
cmd	YTemperature target get_advMode

Retourne :

une valeur parmi Y_ADVMODE_IMMEDIATE, Y_ADVMODE_PERIOD_AVG, Y_ADVMODE_PERIOD_MIN et Y_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 Y_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	function get_advertisedValue (): string
vb	function get_advertisedValue () As String
cs	string get_advertisedValue ()
java	String get_advertisedValue ()
uwp	Task<string> get_advertisedValue ()
py	def get_advertisedValue ()
php	function get_advertisedValue ()
es	function 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 Y_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	function get_currentRawValue (): double
vb	function get_currentRawValue () As Double
cs	double get_currentRawValue ()
java	double get_currentRawValue ()
uwp	Task<double> get_currentRawValue ()
py	def get_currentRawValue ()
php	function get_currentRawValue ()
es	function 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 Y_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	function get_currentValue (): double
vb	function get_currentValue () As Double
cs	double get_currentValue ()
java	double get_currentValue ()
uwp	Task<double> get_uwpCurrentValue ()
py	def get_currentValue ()
php	function get_currentValue ()
es	function get_currentValue ()
cmd	YTemperature target get_currentValue

Retourne :

une valeur numérique représentant la valeur actuelle de la température, en degrés Celsius, sous forme de nombre à virgule

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

temperature→get_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	function get_dataLogger (): TYDataLogger
vb	function get_dataLogger () As YDataLogger
cs	YDataLogger get_dataLogger ()
java	YDataLogger get_dataLogger ()
uwp	Task<YDataLogger> get_dataLogger ()
py	def get_dataLogger ()
php	function get_dataLogger ()
es	function get_dataLogger ()

Cette méthode retourne un objet de la classe YDataLogger qui permet de contrôler les paramètres globaux de l'enregistreur de données. L'objet retourné ne doit pas être libéré.

Retourne :

un objet de classe 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	function get_errorMessage (): string
vb	function get_errorMessage () As String
cs	string get_errorMessage ()
java	String get_errorMessage ()
py	def get_errorMessage ()
php	function get_errorMessage ()
es	function 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	<code>function get_errorType()</code>
cpp	<code>YRETCODE get_errorType()</code>
pas	<code>function get_errorType(): YRETCODE</code>
vb	<code>function get_errorType() As YRETCODE</code>
cs	<code>YRETCODE get_errorType()</code>
java	<code>int get_errorType()</code>
py	<code>def get_errorType()</code>
php	<code>function get_errorType()</code>
es	<code>function get_errorType()</code>

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

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation 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	def get_friendlyName ()
php	function get_friendlyName ()
es	function get_friendlyName ()

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

Retourne :

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

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

temperature→**get_functionDescriptor()****YTemperature****temperature**→**functionDescriptor()**

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	function get_functionDescriptor (): YFUN_DESCR
vb	function get_functionDescriptor () As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor ()
java	String get_functionDescriptor ()
py	def get_functionDescriptor ()
php	function get_functionDescriptor ()
es	function get_functionDescriptor ()

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

Retourne :

un identifiant de type YFUN_DESCR.

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

temperature→**get_functionId()****YTemperature****temperature**→**functionId()**

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	def get_functionId ()
php	function get_functionId ()
es	function get_functionId ()

Par exemple `re lay1`.

Retourne :

une chaîne de caractères identifiant le capteur de température (ex: `re lay1`)

En cas d'erreur, déclenche une exception ou retourne `Y_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	def get_hardwareId ()
php	function get_hardwareId ()
es	function get_hardwareId ()

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du capteur de température (par exemple RELAYL01-123456 . relay1).

Retourne :

une chaîne de caractères identifiant le capteur de température (ex: RELAYL01-123456 . relay1)

En cas d'erreur, déclenche une exception ou retourne Y_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	function get_highestValue (): double
vb	function get_highestValue () As Double
cs	double get_highestValue ()
java	double get_highestValue ()
uwp	Task<double> get_highestValue ()
py	def get_highestValue ()
php	function get_highestValue ()
es	function 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 **Y_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	function get_logFrequency (): string
vb	function get_logFrequency () As String
cs	string get_logFrequency ()
java	String get_logFrequency ()
uwp	Task<string> get_logFrequency ()
py	def get_logFrequency ()
php	function get_logFrequency ()
es	function 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 Y_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	function get_logicalName (): string
vb	function get_logicalName () As String
cs	string get_logicalName ()
java	String get_logicalName ()
uwp	Task<string> get_logicalName ()
py	def get_logicalName ()
php	function get_logicalName ()
es	function 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 Y_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	function get_lowestValue (): double
vb	function get_lowestValue () As Double
cs	double get_lowestValue ()
java	double get_lowestValue ()
uwp	Task<double> get_lowestValue ()
py	def get_lowestValue ()
php	function get_lowestValue ()
es	function 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 **Y_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	function get_module (): TYModule
vb	function get_module () As YModule
cs	YModule get_module ()
java	YModule get_module ()
py	def get_module ()
php	function get_module ()
es	function 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 DataSet 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 (s64 startTime , s64 endTime)
m	-(YDataSet*) recordedData : (s64) startTime : (s64) endTime
pas	function get_recordedData (startTime : int64, endTime : int64): TYDataSet
vb	function get_recordedData () As YDataSet
cs	YDataSet get_recordedData (long startTime , long endTime)
java	YDataSet get_recordedData (long startTime , long endTime)
uwp	Task<YDataSet> get_recordedData (long startTime , long endTime)
py	def get_recordedData (startTime , endTime)
php	function get_recordedData (\$ startTime , \$ endTime)
es	function get_recordedData (startTime , endTime)
cmd	YTemperature target get_recordedData startTime endTime

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

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

Paramètres :

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

Retourne :

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

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	function get_reportFrequency (): string
vb	function get_reportFrequency () As String
cs	string get_reportFrequency ()
java	String get_reportFrequency ()
uwp	Task<string> get_reportFrequency ()
py	def get_reportFrequency ()
php	function get_reportFrequency ()
es	function 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 Y_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	function get_resolution (): double
vb	function get_resolution () As Double
cs	double get_resolution ()
java	double get_resolution ()
uwp	Task<double> get_resolution ()
py	def get_resolution ()
php	function get_resolution ()
es	function 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.

Retourne :

une valeur numérique représentant la résolution des valeurs mesurées

En cas d'erreur, déclenche une exception ou retourne Y_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	function get_sensorState (): LongInt
vb	function get_sensorState () As Integer
cs	int get_sensorState ()
java	int get_sensorState ()
uwp	Task<int> get_sensorState ()
py	def get_sensorState ()
php	function get_sensorState ()
es	function 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 Y_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	function get_sensorType (): Integer
vb	function get_sensorType () As Integer
cs	int get_sensorType ()
java	int get_sensorType ()
uwp	Task<int> get_sensorType ()
py	def get_sensorType ()
php	function get_sensorType ()
es	function get_sensorType ()
cmd	YTemperature target get_sensorType

Retourne :

une valeur parmi Y_SENSORTYPE_DIGITAL, Y_SENSORTYPE_TYPE_K, Y_SENSORTYPE_TYPE_E, Y_SENSORTYPE_TYPE_J, Y_SENSORTYPE_TYPE_N, Y_SENSORTYPE_TYPE_R, Y_SENSORTYPE_TYPE_S, Y_SENSORTYPE_TYPE_T, Y_SENSORTYPE_PT100_4WIRES, Y_SENSORTYPE_PT100_3WIRES, Y_SENSORTYPE_PT100_2WIRES, Y_SENSORTYPE_RES_OHM, Y_SENSORTYPE_RES_NTC, Y_SENSORTYPE_RES_LINEAR et Y_SENSORTYPE_RES_INTERNAL représentant le type de capteur de température utilisé par le module

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

temperature→get_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	function get_signalUnit (): string
vb	function get_signalUnit () As String
cs	string get_signalUnit ()
java	String get_signalUnit ()
uwp	Task<string> get_signalUnit ()
py	def get_signalUnit ()
php	function get_signalUnit ()
es	function 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 Y_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	function get_signalValue (): double
vb	function get_signalValue () As Double
cs	double get_signalValue ()
java	double get_signalValue ()
uwp	Task<double> get_signalValue ()
py	def get_signalValue ()
php	function get_signalValue ()
es	function 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 Y_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	function get_unit (): string
vb	function get_unit () As String
cs	string get_unit ()
java	String get_unit ()
uwp	Task<string> get_unit ()
py	def get_unit ()
php	function get_unit ()
es	function 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 Y_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	function get_userdata (): Tobject
vb	function get_userdata () As Object
cs	object get_userdata ()
java	Object get_userdata ()
py	def get_userdata ()
php	function get_userdata ()
es	function get_userdata ()

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

Retourne :

l'objet stocké précédemment par l'appelant.

temperature→isOnline()**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	function isOnline (): boolean
vb	function isOnline () As Boolean
cs	bool isOnline ()
java	boolean isOnline ()
py	def isOnline ()
php	function isOnline ()
es	function 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→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	function load (msValidity : u64): YRETCODE
vb	function load (ByVal msValidity As Long) As YRETCODE
cs	YRETCODE load (ulong msValidity)
java	int load (long msValidity)
py	def load (msValidity)
php	function load (\$msValidity)
es	function 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.

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	function loadAttribute(attrName: string): string
vb	function loadAttribute() As String
cs	string loadAttribute(string attrName)
java	String loadAttribute(String attrName)
uwp	Task<string> loadAttribute(string attrName)
py	def loadAttribute(attrName)
php	function loadAttribute(\$attrName)
es	function loadAttribute(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 function loadCalibrationPoints( var rawValues: TDoubleArray,
                                   var refValues: TDoubleArray): LongInt
vb   procedure loadCalibrationPoints( )
cs   int loadCalibrationPoints( List<double> rawValues,
                               List<double> refValues)
java int loadCalibrationPoints( ArrayList<Double> rawValues,
                               ArrayList<Double> refValues)
uwp Task<int> loadCalibrationPoints( List<double> rawValues,
                                   List<double> refValues)
py   def loadCalibrationPoints( rawValues, refValues)
php  function loadCalibrationPoints( &$rawValues, &$refValues)
es   function 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 function loadThermistorResponseTable( var tempValues: TDoubleArray,
                                     var resValues: TDoubleArray): LongInt
vb   procedure loadThermistorResponseTable( )
cs   int loadThermistorResponseTable( List<double> tempValues,
                                     List<double> resValues)
java int loadThermistorResponseTable( ArrayList<Double> tempValues,
                                     ArrayList<Double> resValues)
uwp Task<int> loadThermistorResponseTable( List<double> tempValues,
                                     List<double> resValues)
py   def loadThermistorResponseTable( tempValues, resValues)
php   function loadThermistorResponseTable( &$amp;tempValues, &$amp;resValues)
es   function 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 Celcius) 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 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 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	function muteValueCallbacks (): LongInt
vb	function muteValueCallbacks () As Integer
cs	int muteValueCallbacks ()
java	int muteValueCallbacks ()
uwp	Task<int> muteValueCallbacks ()
py	def muteValueCallbacks ()
php	function muteValueCallbacks ()
es	function 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 déclenchement 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()`.

js	function nextTemperature ()
cpp	YTemperature * nextTemperature ()
m	-(YTemperature*) nextTemperature
pas	function nextTemperature (): TYTemperature
vb	function nextTemperature () As YTemperature
cs	YTemperature nextTemperature ()
java	YTemperature nextTemperature ()
uwp	YTemperature nextTemperature ()
py	def nextTemperature ()
php	function nextTemperature ()
es	function nextTemperature ()

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) callback
pas	function registerTimedReportCallback (callback : TYTemperatureTimedReportCallback): LongInt
vb	function registerTimedReportCallback () As Integer
cs	int registerTimedReportCallback (TimedReportCallback callback)
java	int registerTimedReportCallback (TimedReportCallback callback)
uwp	Task<int> registerTimedReportCallback (TimedReportCallback callback)
py	def registerTimedReportCallback (callback)
php	function registerTimedReportCallback (\$ callback)
es	function 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'objet 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) callback
pas	function registerValueCallback (callback : TYTemperatureValueCallback): LongInt
vb	function registerValueCallback () As Integer
cs	int registerValueCallback (ValueCallback callback)
java	int registerValueCallback (UpdateCallback callback)
uwp	Task<int> registerValueCallback (ValueCallback callback)
py	def registerValueCallback (callback)
php	function registerValueCallback (\$callback)
es	function 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	function 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	Task<int> set_advMode (int newval)
py	def set_advMode (newval)
php	function set_advMode (\$newval)
es	function set_advMode (newval)
cmd	YTemperature target set_advMode newval

Paramètres :

newval une valeur parmi Y_ADVMODE_IMMEDIATE, Y_ADVMODE_PERIOD_AVG, Y_ADVMODE_PERIOD_MIN et Y_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	function 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	Task<int> set_highestValue(double newval)
py	def set_highestValue(newval)
php	function set_highestValue(\$newval)
es	function set_highestValue(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 (const string& newval)
m	-(int) setLogFrequency : (NSString*) newval
pas	function 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	Task<int> set_logFrequency (string newval)
py	def set_logFrequency (newval)
php	function set_logFrequency (\$ newval)
es	function set_logFrequency (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".

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 (const string& newval)
m	-(int) setLogicalName : (NSString*) newval
pas	function set_logicalName (newval : string): integer
vb	function set_logicalName (ByVal newval As String) As Integer
cs	int set_logicalName (string newval)
java	int set_logicalName (String newval)
uwp	Task<int> set_logicalName (string newval)
py	def set_logicalName (newval)
php	function set_logicalName (\$ newval)
es	function set_logicalName (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	function 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	Task<int> set_lowestValue (double newval)
py	def set_lowestValue (newval)
php	function set_lowestValue (\$newval)
es	function set_lowestValue (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	function set_ntcParameters (res25 : double, beta : double): LongInt
vb	function set_ntcParameters () As Integer
cs	int set_ntcParameters (double res25 , double beta)
java	int set_ntcParameters (double res25 , double beta)
uwp	Task<int> set_ntcParameters (double res25 , double beta)
py	def set_ntcParameters (res25 , beta)
php	function set_ntcParameters (\$res25 , \$beta)
es	function set_ntcParameters (res25 , 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 (const string& newval)
m	-(int) setReportFrequency : (NSString*) newval
pas	function 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	Task<int> set_reportFrequency (string newval)
py	def set_reportFrequency (newval)
php	function set_reportFrequency (\$ newval)
es	function set_reportFrequency (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".

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	function 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	Task<int> set_resolution (double newval)
py	def set_resolution (newval)
php	function set_resolution (\$newval)
es	function set_resolution (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.

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	function 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	Task<int> set_sensorType (int newval)
py	def set_sensorType (newval)
php	function set_sensorType (\$newval)
es	function set_sensorType (newval)
cmd	YTemperature target set_sensorType newval

Cette fonction sert à spécifier le type de thermocouple (K,E, etc..) raccordé au module. Cette fonction n'aura pas d'effet si le module utilise un capteur digital ou un thermistor. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur parmi Y_SENSORTYPE_DIGITAL, Y_SENSORTYPE_TYPE_K, Y_SENSORTYPE_TYPE_E, Y_SENSORTYPE_TYPE_J, Y_SENSORTYPE_TYPE_N, Y_SENSORTYPE_TYPE_R, Y_SENSORTYPE_TYPE_S, Y_SENSORTYPE_TYPE_T, Y_SENSORTYPE_PT100_4WIRES, Y_SENSORTYPE_PT100_3WIRES, Y_SENSORTYPE_PT100_2WIRES, Y_SENSORTYPE_RES_OHM, Y_SENSORTYPE_RES_NTC, Y_SENSORTYPE_RES_LINEAR et Y_SENSORTYPE_RES_INTERNAL 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 function set_thermistorResponseTable( tempValues: TDoubleArray,
                                     resValues: TDoubleArray): LongInt
vb  procedure set_thermistorResponseTable( )
cs  int set_thermistorResponseTable( List<double> tempValues,
                                     List<double> resValues)
java int set_thermistorResponseTable( ArrayList<Double> tempValues,
                                     ArrayList<Double> resValues)
uwp Task<int> set_thermistorResponseTable( List<double> tempValues,
                                     List<double> resValues)
py  def set_thermistorResponseTable( tempValues, resValues)
php function set_thermistorResponseTable( $tempValues, $resValues)
es  function set_thermistorResponseTable( tempValues, 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 Celcius) 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()**

Change l'unité dans laquelle la température mesurée est exprimée.

js	function set_unit (newval)
cpp	int set_unit (const string& newval)
m	-(int) setUnit : (NSString*) newval
pas	function 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	Task<int> set_unit (string newval)
py	def set_unit (newval)
php	function set_unit (\$ newval)
es	function set_unit (newval)
cmd	YTemperature target set_unit newval

Cette unité est une chaîne de caractère. 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 se termine 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

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	procedure set_userData (data : TObject)
vb	procedure set_userData (ByVal data As Object)
cs	void set_userData (object data)
java	void set_userData (Object data)
py	def set_userData (data)
php	function set_userData (\$data)
es	function set_userData (data)

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

Paramètres :

data objet quelconque à mémoriser

temperature→startDataLogger()**YTemperature**

Démarre l'enregistreur de données du module.

js	function startDataLogger ()
cpp	int startDataLogger ()
m	-(int) startDataLogger
pas	function startDataLogger (): LongInt
vb	function startDataLogger () As Integer
cs	int startDataLogger ()
java	int startDataLogger ()
uwp	Task<int> startDataLogger ()
py	def startDataLogger ()
php	function startDataLogger ()
es	function 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	function stopDataLogger (): LongInt
vb	function stopDataLogger () As Integer
cs	int stopDataLogger ()
java	int stopDataLogger ()
uwp	Task<int> stopDataLogger ()
py	def stopDataLogger ()
php	function stopDataLogger ()
es	function 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	function unmuteValueCallbacks (): LongInt
vb	function unmuteValueCallbacks () As Integer
cs	int unmuteValueCallbacks ()
java	int unmuteValueCallbacks ()
uwp	Task<int> unmuteValueCallbacks ()
py	def unmuteValueCallbacks ()
php	function unmuteValueCallbacks ()
es	function 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)
```

```
es function 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.

22.6. Interface de la fonction DataLogger

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_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');
es	in HTML: <script src=" ../lib/yocto_api.js"></script> in node.js: require('yoctolib-es2017/yocto_api.js');

Fonction globales

yFindDataLogger(func)

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

yFindDataLoggerInContext(yctx, func)

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

yFirstDataLogger()

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

yFirstDataLoggerInContext(yctx)

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

Méthodes des objets YDataLogger

datalogger→clearCache()

Invalide le cache.

datalogger→describe()

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

datalogger→forgetAllDataStreams()

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

datalogger→get_advertisedValue()

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

datalogger→get_autoStart()

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

datalogger→get_beaconDriven()

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

datalogger→get_currentRunIndex()

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

datalogger→get_dataSets()

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

datalogger→get_dataStreams(v)

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur (ancienne méthode).

datalogger→get_errorMessage()

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

datalogger→get_errorType()

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

datalogger→get_friendlyName()

Retourne un identifiant global de l'enregistreur de données au format NOM_MODULE . NOM_FONCTION.

datalogger→get_functionDescriptor()

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

datalogger→get_functionId()

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

datalogger→get_hardwareId()

Retourne l'identifiant matériel unique de l'enregistreur de données au format SERIAL . FUNCTIONID.

datalogger→get_logicalName()

Retourne le nom logique de l'enregistreur de données.

datalogger→get_module()

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

datalogger→get_module_async(callback, context)

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

datalogger→get_recording()

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

datalogger→get_timeUTC()

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

datalogger→get_userData()

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

datalogger→isOnline()

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

datalogger→isOnline_async(callback, context)

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

datalogger→load(msValidity)

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

datalogger→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.

datalogger→load_async(msValidity, callback, context)

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

datalogger→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

datalogger→nextDataLogger()

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

datalogger→registerValueCallback(callback)

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

datalogger→**set_autoStart(newval)**

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

datalogger→**set_beaconDriven(newval)**

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

datalogger→**set_logicalName(newval)**

Modifie le nom logique de l'enregistreur de données.

datalogger→**set_recording(newval)**

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

datalogger→**set_timeUTC(newval)**

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

datalogger→**set_userData(data)**

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

datalogger→**unmuteValueCallbacks()**

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

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

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

YDataLogger.FindDataLogger() yFindDataLogger()

YDataLogger

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

js	function yFindDataLogger (func)
cpp	YDataLogger* yFindDataLogger (string func)
m	+(YDataLogger*) FindDataLogger : (NSString*) func
pas	function yFindDataLogger (func : string): TYDataLogger
vb	function yFindDataLogger (ByVal func As String) As YDataLogger
cs	YDataLogger FindDataLogger (string func)
java	YDataLogger FindDataLogger (String func)
uwp	YDataLogger FindDataLogger (string func)
py	def FindDataLogger (func)
php	function yFindDataLogger (\$func)
es	function FindDataLogger (func)

L'identifiant peut être spécifié sous plusieurs formes:

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

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

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 l'application.

Paramètres :

func une chaîne de caractères qui référence l'enregistreur de données sans ambiguïté

Retourne :

un objet de classe `YDataLogger` qui permet ensuite de contrôler l'enregistreur de données.

YDataLogger.FindDataLoggerInContext() yFindDataLoggerInContext()

YDataLogger

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

```
java YDataLogger FindDataLoggerInContext( YAPIContext yctx,
                                           String func)
uwp  YDataLogger FindDataLoggerInContext( YAPIContext yctx,
                                           string func)
es   function FindDataLoggerInContext( 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 l'enregistreur de données soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YDataLogger.isOnline()` pour tester si l'enregistreur de données est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'enregistreur de données sans ambiguïté

Retourne :

un objet de classe `YDataLogger` qui permet ensuite de contrôler l'enregistreur de données.

YDataLogger.FirstDataLogger() yFirstDataLogger()

YDataLogger

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

js	function yFirstDataLogger ()
cpp	YDataLogger* yFirstDataLogger ()
m	+(YDataLogger*) FirstDataLogger
pas	function yFirstDataLogger (): TYDataLogger
vb	function yFirstDataLogger () As YDataLogger
cs	YDataLogger FirstDataLogger ()
java	YDataLogger FirstDataLogger ()
uwp	YDataLogger FirstDataLogger ()
py	def FirstDataLogger ()
php	function yFirstDataLogger ()
es	function FirstDataLogger ()

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

Retourne :

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

YDataLogger.FirstDataLoggerInContext() yFirstDataLoggerInContext()

YDataLogger

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

java	<code>YDataLogger FirstDataLoggerInContext(YAPIContext yctx)</code>
uwp	<code>YDataLogger FirstDataLoggerInContext(YAPIContext yctx)</code>
es	<code>function FirstDataLoggerInContext(yctx)</code>

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

Paramètres :

`yctx` un contexte YAPI.

Retourne :

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

datalogger→clearCache()**YDataLogger**

Invalide le cache.

js	function clearCache ()
cpp	void clearCache ()
m	-(void) clearCache
pas	procedure clearCache ()
vb	procedure clearCache ()
cs	void clearCache ()
java	void clearCache ()
py	def clearCache ()
php	function clearCache ()
es	function clearCache ()

Invalide le cache des valeurs courantes de l'enregistreur de données. Force le prochain appel à une méthode `get_xxx()` ou `loadxxx()` pour charger les données depuis le module.

datalogger→describe()**YDataLogger**

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

js	function describe ()
cpp	string describe ()
m	-(NSString*) describe
pas	function describe (): string
vb	function describe () As String
cs	string describe ()
java	String describe ()
py	def describe ()
php	function describe ()
es	function describe ()

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilisé lors du premier accès à la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYL01-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant l'enregistreur de données (ex:
`Relay(MyCustomName.relay1)=RELAYL01-123456.relay1`)

datalogger→forgetAllDataStreams()**YDataLogger**

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

js	function forgetAllDataStreams ()
cpp	int forgetAllDataStreams ()
m	-(int) forgetAllDataStreams
pas	function forgetAllDataStreams (): LongInt
vb	function forgetAllDataStreams () As Integer
cs	int forgetAllDataStreams ()
java	int forgetAllDataStreams ()
uwp	Task<int> forgetAllDataStreams ()
py	def forgetAllDataStreams ()
php	function forgetAllDataStreams ()
es	function forgetAllDataStreams ()
cmd	YDataLogger target forgetAllDataStreams

Cette méthode remet aussi à zéro le compteur de Runs.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

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

js	function get_advertisedValue ()
cpp	string get_advertisedValue ()
m	-(NSString*) advertisedValue
pas	function get_advertisedValue (): string
vb	function get_advertisedValue () As String
cs	string get_advertisedValue ()
java	String get_advertisedValue ()
uwp	Task<string> get_advertisedValue ()
py	def get_advertisedValue ()
php	function get_advertisedValue ()
es	function get_advertisedValue ()
cmd	YDataLogger target get_advertisedValue

Retourne :

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

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

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

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

js	function get_autoStart ()
cpp	Y_AUTOSTART_enum get_autoStart ()
m	-(Y_AUTOSTART_enum) autoStart
pas	function get_autoStart (): Integer
vb	function get_autoStart () As Integer
cs	int get_autoStart ()
java	int get_autoStart ()
uwp	Task<int> get_autoStart ()
py	def get_autoStart ()
php	function get_autoStart ()
es	function get_autoStart ()
cmd	YDataLogger target get_autoStart

Retourne :

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

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

datalogger→get_beaconDriven()**YDataLogger****datalogger→beaconDriven()**

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

js	function get_beaconDriven ()
cpp	Y_BEACONDRIVEN_enum get_beaconDriven ()
m	-(Y_BEACONDRIVEN_enum) beaconDriven
pas	function get_beaconDriven (): Integer
vb	function get_beaconDriven () As Integer
cs	int get_beaconDriven ()
java	int get_beaconDriven ()
uwp	Task<int> get_beaconDriven ()
py	def get_beaconDriven ()
php	function get_beaconDriven ()
es	function get_beaconDriven ()
cmd	YDataLogger target get_beaconDriven

Retourne :

soit Y_BEACONDRIVEN_OFF, soit Y_BEACONDRIVEN_ON, selon vrais si l'enregistreur de données est synchronisé avec la balise de localisation

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

datalogger→get_currentRunIndex()**YDataLogger****datalogger→currentRunIndex()**

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

js	function get_currentRunIndex ()
cpp	int get_currentRunIndex ()
m	-(int) currentRunIndex
pas	function get_currentRunIndex (): LongInt
vb	function get_currentRunIndex () As Integer
cs	int get_currentRunIndex ()
java	int get_currentRunIndex ()
uwp	Task<int> get_currentRunIndex ()
py	def get_currentRunIndex ()
php	function get_currentRunIndex ()
es	function get_currentRunIndex ()
cmd	YDataLogger target get_currentRunIndex

Retourne :

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

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

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

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

js	function get_dataSets ()
cpp	vector<YDataSet> get_dataSets ()
m	-(NSMutableArray*) dataSets
pas	function get_dataSets (): TYDataSetArray
vb	function get_dataSets () As List
cs	List<YDataSet> get_dataSets ()
java	ArrayList<YDataSet> get_dataSets ()
uwp	Task<List<YDataSet>> get_dataSets ()
py	def get_dataSets ()
php	function get_dataSets ()
es	function get_dataSets ()
cmd	YDataLogger target get_dataSets

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

Retourne :

une liste d'objets YDataSet

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

datalogger→get_dataStreams()**YDataLogger****datalogger→dataStreams()**

Construit une liste de toutes les séquences de mesures mémorisées par l'enregistreur (ancienne méthode).

js	function get_dataStreams (v)
cpp	int get_dataStreams ()
m	-(int) dataStreams : (NSArray**) v
pas	function get_dataStreams (v : Tlist): integer
vb	procedure get_dataStreams (ByVal v As List)
cs	int get_dataStreams (List<YDataStream> v)
java	int get_dataStreams (ArrayList<YDataStream> v)
py	def get_dataStreams (v)
php	function get_dataStreams (&\$ v)
es	function get_dataStreams (v)

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

Cette méthode est préservée pour maintenir la compatibilité avec les applications existantes. Pour les nouvelles applications, il est préférable d'utiliser la méthode `get_dataSets()` ou d'appeler directement la méthode `get_recordedData()` sur l'objet représentant le capteur désiré.

Paramètres :

v un tableau de YDataStreams qui sera rempli avec les séquences trouvées

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

datalogger→get_errorMessage()**YDataLogger****datalogger→errorMessage()**

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

js	function get_errorMessage ()
cpp	string get_errorMessage ()
m	-(NSString*) errorMessage
pas	function get_errorMessage (): string
vb	function get_errorMessage () As String
cs	string get_errorMessage ()
java	String get_errorMessage ()
py	def get_errorMessage ()
php	function get_errorMessage ()
es	function get_errorMessage ()

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

Retourne :

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

datalogger→get_errorType()**YDataLogger****datalogger→errorType()**

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

js	function get_errorType ()
cpp	YRETCODE get_errorType ()
pas	function get_errorType (): YRETCODE
vb	function get_errorType () As YRETCODE
cs	YRETCODE get_errorType ()
java	int get_errorType ()
py	def get_errorType ()
php	function get_errorType ()
es	function get_errorType ()

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

Retourne :

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

datalogger→get_friendlyName()**YDataLogger****datalogger→friendlyName()**

Retourne un identifiant global de l'enregistreur de données 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	def get_friendlyName ()
php	function get_friendlyName ()
es	function get_friendlyName ()

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

Retourne :

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

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

datalogger→**get_functionDescriptor()****YDataLogger****datalogger**→**functionDescriptor()**

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	function get_functionDescriptor (): YFUN_DESCR
vb	function get_functionDescriptor () As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor ()
java	String get_functionDescriptor ()
py	def get_functionDescriptor ()
php	function get_functionDescriptor ()
es	function get_functionDescriptor ()

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

Retourne :

un identifiant de type YFUN_DESCR.

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

datalogger→**get_functionId()****YDataLogger****datalogger**→**functionId()**

Retourne l'identifiant matériel de l'enregistreur de données, 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	def get_functionId ()
php	function get_functionId ()
es	function get_functionId ()

Par exemple `relay1`.

Retourne :

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

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

datalogger→get_hardwareId()**YDataLogger****datalogger→hardwareId()**

Retourne l'identifiant matériel unique de l'enregistreur de données 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	def get_hardwareId ()
php	function get_hardwareId ()
es	function get_hardwareId ()

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'enregistreur de données (par exemple `RELAYL01-123456.relay1`).

Retourne :

une chaîne de caractères identifiant l'enregistreur de données (ex: `RELAYL01-123456.relay1`)

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

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

Retourne le nom logique de l'enregistreur de données.

js	function get_logicalName ()
cpp	string get_logicalName ()
m	-(NSString*) logicalName
pas	function get_logicalName (): string
vb	function get_logicalName () As String
cs	string get_logicalName ()
java	String get_logicalName ()
uwp	Task<string> get_logicalName ()
py	def get_logicalName ()
php	function get_logicalName ()
es	function get_logicalName ()
cmd	YDataLogger target get_logicalName

Retourne :

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

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

datalogger→get_module()**datalogger→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	function get_module (): TYModule
vb	function get_module () As YModule
cs	YModule get_module ()
java	YModule get_module ()
py	def get_module ()
php	function get_module ()
es	function get_module ()

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

Retourne :

une instance de YModule

datalogger→**get_module_async()****YDataLogger****datalogger**→**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.

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

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

js	function get_recording ()
cpp	Y_RECORDING_enum get_recording ()
m	-(Y_RECORDING_enum) recording
pas	function get_recording (): Integer
vb	function get_recording () As Integer
cs	int get_recording ()
java	int get_recording ()
uwp	Task<int> get_recording ()
py	def get_recording ()
php	function get_recording ()
es	function get_recording ()
cmd	YDataLogger target get_recording

Retourne :

une valeur parmi Y_RECORDING_OFF, Y_RECORDING_ON et Y_RECORDING_PENDING représentant l'état d'activation de l'enregistreur de données

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

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

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

js	function get_timeUTC ()
cpp	s64 get_timeUTC ()
m	-(s64) timeUTC
pas	function get_timeUTC (): int64
vb	function get_timeUTC () As Long
cs	long get_timeUTC ()
java	long get_timeUTC ()
uwp	Task<long> get_timeUTC ()
py	def get_timeUTC ()
php	function get_timeUTC ()
es	function get_timeUTC ()
cmd	YDataLogger target get_timeUTC

Retourne :

un entier représentant le timestamp Unix de l'heure UTC actuelle, lorsqu'elle est connue

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

datalogger→get_userdata()**YDataLogger****datalogger→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	function get_userdata (): Tobject
vb	function get_userdata () As Object
cs	object get_userdata ()
java	Object get_userdata ()
py	def get_userdata ()
php	function get_userdata ()
es	function get_userdata ()

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

Retourne :

l'objet stocké précédemment par l'appelant.

datalogger→isOnline()**YDataLogger**

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

js	function isOnline ()
cpp	bool isOnline ()
m	-(BOOL) isOnline
pas	function isOnline (): boolean
vb	function isOnline () As Boolean
cs	bool isOnline ()
java	boolean isOnline ()
py	def isOnline ()
php	function isOnline ()
es	function isOnline ()

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

Retourne :

`true` si l'enregistreur de données est joignable, `false` sinon

datalogger→isOnline_async()**YDataLogger**

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
```

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

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.

datalogger→load()**YDataLogger**

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

js	function load (msValidity)
cpp	YRETCODE load (int msValidity)
m	-(YRETCODE) load : (u64) msValidity
pas	function load (msValidity : u64): YRETCODE
vb	function load (ByVal msValidity As Long) As YRETCODE
cs	YRETCODE load (ulong msValidity)
java	int load (long msValidity)
py	def load (msValidity)
php	function load (\$msValidity)
es	function 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.

datalogger→loadAttribute()**YDataLogger**

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	function loadAttribute (attrName : string): string
vb	function loadAttribute () As String
cs	string loadAttribute (string attrName)
java	String loadAttribute (String attrName)
uwp	Task<string> loadAttribute (string attrName)
py	def loadAttribute (attrName)
php	function loadAttribute (\$attrName)
es	function loadAttribute (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.

datalogger→load_async()**YDataLogger**

Met en cache les valeurs courantes de l'enregistreur de données, 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 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.

datalogger→muteValueCallbacks()**YDataLogger**

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

js	function muteValueCallbacks ()
cpp	int muteValueCallbacks ()
m	-(int) muteValueCallbacks
pas	function muteValueCallbacks (): LongInt
vb	function muteValueCallbacks () As Integer
cs	int muteValueCallbacks ()
java	int muteValueCallbacks ()
uwp	Task<int> muteValueCallbacks ()
py	def muteValueCallbacks ()
php	function muteValueCallbacks ()
es	function muteValueCallbacks ()
cmd	YDataLogger 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 déclenchement 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.

datalogger→nextDataLogger()**YDataLogger**

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

js	<code>function nextDataLogger()</code>
cpp	<code>YDataLogger * nextDataLogger()</code>
m	<code>-(YDataLogger*) nextDataLogger</code>
pas	<code>function nextDataLogger(): TYDataLogger</code>
vb	<code>function nextDataLogger() As YDataLogger</code>
cs	<code>YDataLogger nextDataLogger()</code>
java	<code>YDataLogger nextDataLogger()</code>
uwp	<code>YDataLogger nextDataLogger()</code>
py	<code>def nextDataLogger()</code>
php	<code>function nextDataLogger()</code>
es	<code>function nextDataLogger()</code>

Retourne :

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

datalogger→registerValueCallback()**YDataLogger**

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

js	function registerValueCallback (callback)
cpp	int registerValueCallback (YDataLoggerValueCallback callback)
m	-(int) registerValueCallback : (YDataLoggerValueCallback) callback
pas	function registerValueCallback (callback : TYDataLoggerValueCallback): LongInt
vb	function registerValueCallback () As Integer
cs	int registerValueCallback (ValueCallback callback)
java	int registerValueCallback (UpdateCallback callback)
uwp	Task<int> registerValueCallback (ValueCallback callback)
py	def registerValueCallback (callback)
php	function registerValueCallback (\$callback)
es	function 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.

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

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

js	function set_autoStart (newval)
cpp	int set_autoStart (Y_AUTOSTART_enum newval)
m	-(int) setAutoStart : (Y_AUTOSTART_enum) newval
pas	function set_autoStart (newval : Integer): integer
vb	function set_autoStart (ByVal newval As Integer) As Integer
cs	int set_autoStart (int newval)
java	int set_autoStart (int newval)
uwp	Task<int> set_autoStart (int newval)
py	def set_autoStart (newval)
php	function set_autoStart (\$newval)
es	function set_autoStart (newval)
cmd	YDataLogger target set_autoStart newval

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

datalogger→set_beaconDriven()**YDataLogger****datalogger→setBeaconDriven()**

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

js	function set_beaconDriven (newval)
cpp	int set_beaconDriven (Y_BEACONDRIVEN_enum newval)
m	-(int) setBeaconDriven : (Y_BEACONDRIVEN_enum) newval
pas	function set_beaconDriven (newval : Integer): integer
vb	function set_beaconDriven (ByVal newval As Integer) As Integer
cs	int set_beaconDriven (int newval)
java	int set_beaconDriven (int newval)
uwp	Task<int> set_beaconDriven (int newval)
py	def set_beaconDriven (newval)
php	function set_beaconDriven (\$ newval)
es	function set_beaconDriven (newval)
cmd	YDataLogger target set_beaconDriven newval

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

Paramètres :

newval soit Y_BEACONDRIVEN_OFF, soit Y_BEACONDRIVEN_ON, selon le mode de synchronisation de l'enregistreur de données

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

datalogger→set_logicalName()**YDataLogger****datalogger→setLogicalName()**

Modifie le nom logique de l'enregistreur de données.

js	function set_logicalName (newval)
cpp	int set_logicalName (const string& newval)
m	-(int) setLogicalName : (NSString*) newval
pas	function set_logicalName (newval : string): integer
vb	function set_logicalName (ByVal newval As String) As Integer
cs	int set_logicalName (string newval)
java	int set_logicalName (String newval)
uwp	Task<int> set_logicalName (string newval)
py	def set_logicalName (newval)
php	function set_logicalName (\$ newval)
es	function set_logicalName (newval)
cmd	YDataLogger target set_logicalName newval

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

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

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

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

js	function set_recording (newval)
cpp	int set_recording (Y_RECORDING_enum newval)
m	-(int) setRecording : (Y_RECORDING_enum) newval
pas	function set_recording (newval : Integer): integer
vb	function set_recording (ByVal newval As Integer) As Integer
cs	int set_recording (int newval)
java	int set_recording (int newval)
uwp	Task<int> set_recording (int newval)
py	def set_recording (newval)
php	function set_recording (\$newval)
es	function set_recording (newval)
cmd	YDataLogger target set_recording newval

Paramètres :

newval une valeur parmi Y_RECORDING_OFF, Y_RECORDING_ON et Y_RECORDING_PENDING représentant l'état d'activation de l'enregistreur de données

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

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

js	function set_timeUTC (newval)
cpp	int set_timeUTC (s64 newval)
m	-(int) setTimeUTC : (s64) newval
pas	function set_timeUTC (newval : int64): integer
vb	function set_timeUTC (ByVal newval As Long) As Integer
cs	int set_timeUTC (long newval)
java	int set_timeUTC (long newval)
uwp	Task<int> set_timeUTC (long newval)
py	def set_timeUTC (newval)
php	function set_timeUTC (\$newval)
es	function set_timeUTC (newval)
cmd	YDataLogger target set_timeUTC newval

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

datalogger→set_userdata()**YDataLogger****datalogger→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	procedure set_userdata (data : TObject)
vb	procedure set_userdata (ByVal data As Object)
cs	void set_userdata (object data)
java	void set_userdata (Object data)
py	def set_userdata (data)
php	function set_userdata (\$data)
es	function set_userdata (data)

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

Paramètres :

data objet quelconque à mémoriser

datalogger→unmuteValueCallbacks()**YDataLogger**

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

js	function unmuteValueCallbacks ()
cpp	int unmuteValueCallbacks ()
m	-(int) unmuteValueCallbacks
pas	function unmuteValueCallbacks (): LongInt
vb	function unmuteValueCallbacks () As Integer
cs	int unmuteValueCallbacks ()
java	int unmuteValueCallbacks ()
uwp	Task<int> unmuteValueCallbacks ()
py	def unmuteValueCallbacks ()
php	function unmuteValueCallbacks ()
es	function unmuteValueCallbacks ()
cmd	YDataLogger 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.

datalogger→wait_async()**YDataLogger**

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)
```

```
es function 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.

22.7. Séquence de données enregistrées

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

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

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_api.js'></script></code>
cpp	<code>#include "yocto_api.h"</code>
m	<code>#import "yocto_api.h"</code>
pas	<code>uses yocto_api;</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>
php	<code>require_once('yocto_api.php');</code>
es	in HTML: <code><script src='../lib/yocto_api.js'></script></code> in node.js: <code>require('yoctolib-es2017/yocto_api.js');</code>

Méthodes des objets YDataSet

dataset→get_endTimeUTC()

Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

dataset→get_functionId()

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

dataset→get_hardwareId()

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format `SERIAL.FUNCTIONID`.

dataset→get_measures()

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

dataset→get_measuresAt(measure)

Retourne les mesures détaillées pour une mesure résumée précédemment retournée par `get_preview()`.

dataset→get_preview()

Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.

dataset→get_progress()

Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.

dataset→get_startTimeUTC()

Retourne l'heure absolue du début des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

dataset→get_summary()

Retourne un objet YMeasure résumant tout le YDataSet.

dataset→get_unit()

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

dataset→loadMore()

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, et met à jour l'indicateur d'avancement.

dataset→loadMore_async(callback, context)

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

dataset→get_endTimeUTC()**YDataSet****dataset→endTimeUTC()**

Retourne l'heure absolue de la fin des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

js	function get_endTimeUTC ()
cpp	s64 get_endTimeUTC ()
m	-(s64) endTimeUTC
pas	function get_endTimeUTC (): int64
vb	function get_endTimeUTC () As Long
cs	long get_endTimeUTC ()
java	long get_endTimeUTC ()
uwp	Task<long> get_endTimeUTC ()
py	def get_endTimeUTC ()
php	function get_endTimeUTC ()
es	function get_endTimeUTC ()

Lorsque l'objet YDataSet est créé, l'heure de fin est celle qui a été passée en paramètre à la fonction `get_dataSet`. Dès le premier appel à la méthode `loadMore()`, l'heure de fin est mise à jour à la dernière mesure effectivement disponible dans l'enregistreur de données pour la plage spécifiée.

Retourne :

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et la dernière mesure.

dataset→**get_functionId()****dataset**→**functionId()**

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

js	function get_functionId ()
cpp	string get_functionId ()
m	-(NSString*) functionId
pas	function get_functionId (): string
vb	function get_functionId () As String
cs	string get_functionId ()
java	String get_functionId ()
uwp	Task<string> get_functionId ()
py	def get_functionId ()
php	function get_functionId ()
es	function get_functionId ()

Par exemple temperature1.

Retourne :

une chaîne de caractères identifiant la fonction (ex: temperature1)

dataset→**get_hardwareId()****YDataSet****dataset**→**hardwareId()**

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format `SERIAL.FUNCTIONID`.

js	<code>function get_hardwareId()</code>
cpp	<code>string get_hardwareId()</code>
m	<code>-(NSString*) hardwareId</code>
pas	<code>function get_hardwareId(): string</code>
vb	<code>function get_hardwareId() As String</code>
cs	<code>string get_hardwareId()</code>
java	<code>String get_hardwareId()</code>
uwp	<code>Task<string> get_hardwareId()</code>
py	<code>def get_hardwareId()</code>
php	<code>function get_hardwareId()</code>
es	<code>function get_hardwareId()</code>

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par exemple `THRMCPL1-123456.temperature1`).

Retourne :

une chaîne de caractères identifiant la fonction (ex: `THRMCPL1-123456.temperature1`)

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

dataset→get_measures()**YDataSet****dataset→measures()**

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

js	function get_measures ()
cpp	vector<YMeasure> get_measures ()
m	-(NSMutableArray*) measures
pas	function get_measures (): TYMeasureArray
vb	function get_measures () As List
cs	List<YMeasure> get_measures ()
java	ArrayList<YMeasure> get_measures ()
uwp	Task<List<YMeasure>> get_measures ()
py	def get_measures ()
php	function get_measures ()
es	function get_measures ()

Chaque élément contient: - le moment où la mesure a débuté - le moment où la mesure s'est terminée - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Avant d'appeler cette méthode, vous devez appeler `LoadMore( )` pour charger des données depuis l'enregistreur sur le module. L'appel doit être répété plusieurs fois pour charger toutes les données, mais vous pouvez commencer à utiliser les données disponibles avant qu'elles n'aient été toutes chargées

Les mesures les plus anciennes sont toujours chargées les premières, et les plus récentes en dernier. De ce fait, les timestamps dans la table des mesures sont normalement par ordre chronologique. La seule exception est dans le cas où il y a eu un ajustement de l'horloge UTC de l'enregistreur de données pendant l'enregistrement.

Retourne :

un tableau d'enregistrements, chaque enregistrement représentant une mesure effectuée à un moment précis.

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

dataset→get_measuresAt()**YDataSet****dataset→measuresAt()**

Retourne les mesures détaillées pour une mesure résumée précédemment retournée par `get_preview()`.

js	<code>function get_measuresAt(measure)</code>
cpp	<code>vector<YMeasure> get_measuresAt(YMeasure measure)</code>
m	<code>-(NSMutableArray*) measuresAt : (YMeasure*) measure</code>
pas	<code>function get_measuresAt(measure: TYMeasure): TYMeasureArray</code>
vb	<code>function get_measuresAt() As List</code>
cs	<code>List<YMeasure> get_measuresAt(YMeasure measure)</code>
java	<code>ArrayList<YMeasure> get_measuresAt(YMeasure measure)</code>
uwp	<code>Task<List<YMeasure>> get_measuresAt(YMeasure measure)</code>
py	<code>def get_measuresAt(measure)</code>
php	<code>function get_measuresAt(\$measure)</code>
es	<code>function get_measuresAt(measure)</code>

Le résultat est fourni sous forme d'une liste d'objets YMeasure.

Paramètres :

measure mesure résumée extraite de la liste précédemment retournée par `get_preview()`.

Retourne :

un tableau d'enregistrements, chaque enregistrement représentant les mesures observée durant un certain intervalle de temps.

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

dataset→get_preview()**YDataSet****dataset→preview()**

Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.

js	function get_preview ()
cpp	vector<YMeasure> get_preview ()
m	-(NSMutableArray*) preview
pas	function get_preview (): TYMeasureArray
vb	function get_preview () As List
cs	List<YMeasure> get_preview ()
java	ArrayList<YMeasure> get_preview ()
uwp	Task<List<YMeasure>> get_preview ()
py	def get_preview ()
php	function get_preview ()
es	function get_preview ()

Chaque élément contient: - le début d'un intervalle de temps - la fin d'un intervalle de temps - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Le résumé des mesures est disponible dès que `LoadMore( )` a été appelé pour la première fois.

Retourne :

un tableau d'enregistrements, chaque enregistrement représentant les mesures observée durant un certain intervalle de temps.

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

dataset→get_progress()**YDataSet****dataset→progress()**

Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.

js	function get_progress ()
cpp	int get_progress ()
m	-(int) progress
pas	function get_progress (): LongInt
vb	function get_progress () As Integer
cs	int get_progress ()
java	int get_progress ()
uwp	Task<int> get_progress ()
py	def get_progress ()
php	function get_progress ()
es	function get_progress ()

A l'instanciation de l'objet par la fonction `get_dataSet( )`, l'avancement est nul. Au fur et à mesure des appels à `loadMore( )`, l'avancement progresse pour atteindre la valeur 100 lorsque toutes les mesures ont été chargées.

Retourne :

un nombre entier entre 0 et 100 représentant l'avancement du chargement des données demandées.

dataset→get_startTimeUTC()**YDataSet****dataset→startTimeUTC()**

Retourne l'heure absolue du début des mesures disponibles, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

js	function get_startTimeUTC ()
cpp	s64 get_startTimeUTC ()
m	-(s64) startTimeUTC
pas	function get_startTimeUTC (): int64
vb	function get_startTimeUTC () As Long
cs	long get_startTimeUTC ()
java	long get_startTimeUTC ()
uwp	Task<long> get_startTimeUTC ()
py	def get_startTimeUTC ()
php	function get_startTimeUTC ()
es	function get_startTimeUTC ()

Lorsque l'objet YDataSet est créé, l'heure de départ est celle qui a été passée en paramètre à la fonction `get_dataSet`. Dès le premier appel à la méthode `loadMore()`, l'heure de départ est mise à jour à la première mesure effectivement disponible dans l'enregistreur de données pour la plage spécifiée.

Retourne :

un entier positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 et la première mesure enregistrée.

dataset→**get_summary()****YDataSet****dataset**→**summary()**

Retourne un objet YMeasure résumant tout le YDataSet.

js	function get_summary ()
cpp	YMeasure get_summary ()
m	-(YMeasure*) summary
pas	function get_summary (): TYMeasure
vb	function get_summary () As YMeasure
cs	YMeasure get_summary ()
java	YMeasure get_summary ()
uwp	Task<YMeasure> get_summary ()
py	def get_summary ()
php	function get_summary ()
es	function get_summary ()

Il inclut les information suivantes: - le moment de la première mesure - le moment de la dernière mesure - la valeur minimale observée dans l'intervalle de temps - la valeur moyenne observée dans l'intervalle de temps - la valeur maximale observée dans l'intervalle de temps

Ce résumé des mesures est disponible dès que `LoadMore( )` a été appelé pour la première fois.

Retourne :
un objet YMeasure

dataset→**get_unit()****dataset**→**unit()**

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

js	function get_unit ()
cpp	string get_unit ()
m	-(NSString*) unit
pas	function get_unit (): string
vb	function get_unit () As String
cs	string get_unit ()
java	String get_unit ()
uwp	Task<string> get_unit ()
py	def get_unit ()
php	function get_unit ()
es	function get_unit ()

Retourne :

une chaîne de caractères représentant une unité physique.

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

dataset→loadMore()**YDataSet**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, et met à jour l'indicateur d'avancement.

js	function loadMore ()
cpp	int loadMore ()
m	-(int) loadMore
pas	function loadMore (): LongInt
vb	function loadMore () As Integer
cs	int loadMore ()
java	int loadMore ()
uwp	Task<int> loadMore ()
py	def loadMore ()
php	function loadMore ()
es	function loadMore ()

Retourne :

un nombre entier entre 0 et 100 représentant l'avancement du chargement des données demandées, ou un code d'erreur négatif en cas de problème.

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

dataset→loadMore_async()**YDataSet**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

```
js function loadMore_async( callback, context)
```

Paramètres :

callback fonction fournie par l'utilisateur, qui sera appelée lorsque la suite du chargement aura été effectué. La fonction callback doit prendre trois arguments: - la variable de contexte à disposition de l'utilisateur - l'objet YDataSet dont la méthode loadMore_async a été appelée - le résultat de l'appel: soit l'état d'avancement du chargement (0...100), ou un code d'erreur négatif en cas de problème.

context variable de contexte à disposition de l'utilisateur

Retourne :

rien.

22.8. Valeur mesurée

Les objets YMeasure sont utilisés dans l'interface de programmation Yoctopuce pour représenter une valeur observée un moment donnée. Ces objets sont utilisés en particulier en conjonction avec la classe YDataSet.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_api.js'></script></code>
cpp	<code>#include "yocto_api.h"</code>
m	<code>#import "yocto_api.h"</code>
pas	<code>uses yocto_api;</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>
php	<code>require_once('yocto_api.php');</code>
es	<code>in HTML: <script src="../../lib/yocto_api.js"></script></code> <code>in node.js: require('yoctolib-es2017/yocto_api.js');</code>

Méthodes des objets YMeasure

measure→get_averageValue()

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par la mesure.

measure→get_endTimeUTC()

Retourne l'heure absolue de la fin de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

measure→get_maxValue()

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par la mesure.

measure→get_minValue()

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par la mesure.

measure→get_startTimeUTC()

Retourne l'heure absolue du début de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

measure→**get_averageValue()****YMeasure****measure**→**averageValue()**

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par la mesure.

js	function get_averageValue ()
cpp	double get_averageValue ()
m	-(double) averageValue
pas	function get_averageValue (): double
vb	function get_averageValue () As Double
cs	double get_averageValue ()
java	double get_averageValue ()
uwp	double get_averageValue ()
py	def get_averageValue ()
php	function get_averageValue ()
es	function get_averageValue ()

Retourne :

un nombre décimal correspondant à la valeur moyenne observée.

measure→get_endTimeUTC()**YMeasure****measure→endTimeUTC()**

Retourne l'heure absolue de la fin de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

js	function get_endTimeUTC ()
cpp	double get_endTimeUTC ()
m	-(double) endTimeUTC
pas	function get_endTimeUTC (): double
vb	function get_endTimeUTC () As Double
cs	double get_endTimeUTC ()
java	double get_endTimeUTC ()
uwp	double get_endTimeUTC ()
py	def get_endTimeUTC ()
php	function get_endTimeUTC ()
es	function get_endTimeUTC ()

Lors que l'enregistrement de données se fait à une fréquence supérieure à une mesure par seconde, le timestamp peuvent inclure une fraction décimale.

Retourne :

un nombre réel positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 UTC et la fin de la mesure.

measure→**get_maxValue()****YMeasure****measure**→**maxValue()**

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par la mesure.

js	function get_maxValue ()
cpp	double get_maxValue ()
m	-(double) maxValue
pas	function get_maxValue (): double
vb	function get_maxValue () As Double
cs	double get_maxValue ()
java	double get_maxValue ()
uwp	double get_maxValue ()
py	def get_maxValue ()
php	function get_maxValue ()
es	function get_maxValue ()

Retourne :

un nombre décimal correspondant à la plus grande valeur observée.

measure→get_minValue()**YMeasure****measure→minValue()**

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par la mesure.

js	function get_minValue ()
cpp	double get_minValue ()
m	-(double) minValue
pas	function get_minValue (): double
vb	function get_minValue () As Double
cs	double get_minValue ()
java	double get_minValue ()
uwp	double get_minValue ()
py	def get_minValue ()
php	function get_minValue ()
es	function get_minValue ()

Retourne :

un nombre décimal correspondant à la plus petite valeur observée.

measure→**get_startTimeUTC()****YMeasure****measure**→**startTimeUTC()**

Retourne l'heure absolue du début de la mesure, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

js	function get_startTimeUTC ()
cpp	double get_startTimeUTC ()
m	-(double) startTimeUTC
pas	function get_startTimeUTC (): double
vb	function get_startTimeUTC () As Double
cs	double get_startTimeUTC ()
java	double get_startTimeUTC ()
uwp	double get_startTimeUTC ()
py	def get_startTimeUTC ()
php	function get_startTimeUTC ()
es	function get_startTimeUTC ()

Lors que l'enregistrement de données se fait à une fréquence supérieure à une mesure par seconde, le timestamp peuvent inclure une fraction décimale.

Retourne :

un nombre réel positif correspondant au nombre de secondes écoulées entre le 1er janvier 1970 UTC et le début de la mesure.

23. Problèmes courants

23.1. 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'il sur le blog de Yoctopuce. Il y a une section dédiée aux débutants ¹.

23.2. 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 exécutable 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 éviteront 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"
```

23.3. Plateformes ARM: HF et EL

Sur ARM il existe deux grandes familles d'exécutables: 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 ArmHF 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 appartient, essayez simplement de lancer un exécutable de chaque famille.

23.4. 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.

23.5. Module alimenté mais invisible pour l'OS

Si votre Yocto-Altimeter 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.

23.6. 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>

23.7. Déconnexions, comportement erratique

Si votre Yocto-Altimeter se comporte de manière erratique et/ou se déconnecte du bus USB sans raison apparente, vérifiez qu'il est alimenté correctement. Évitez les câbles d'une longueur supérieure à 2 mètres. Au besoin, intercalez un hub USB alimenté ⁵ ⁶.

23.8. Module endommagé

Yoctopuce s'efforce de réduire la production de déchets électroniques. Si vous avez l'impression que votre Yocto-Altimeter 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 débarrasser de votre Yocto-Altimeter, 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.



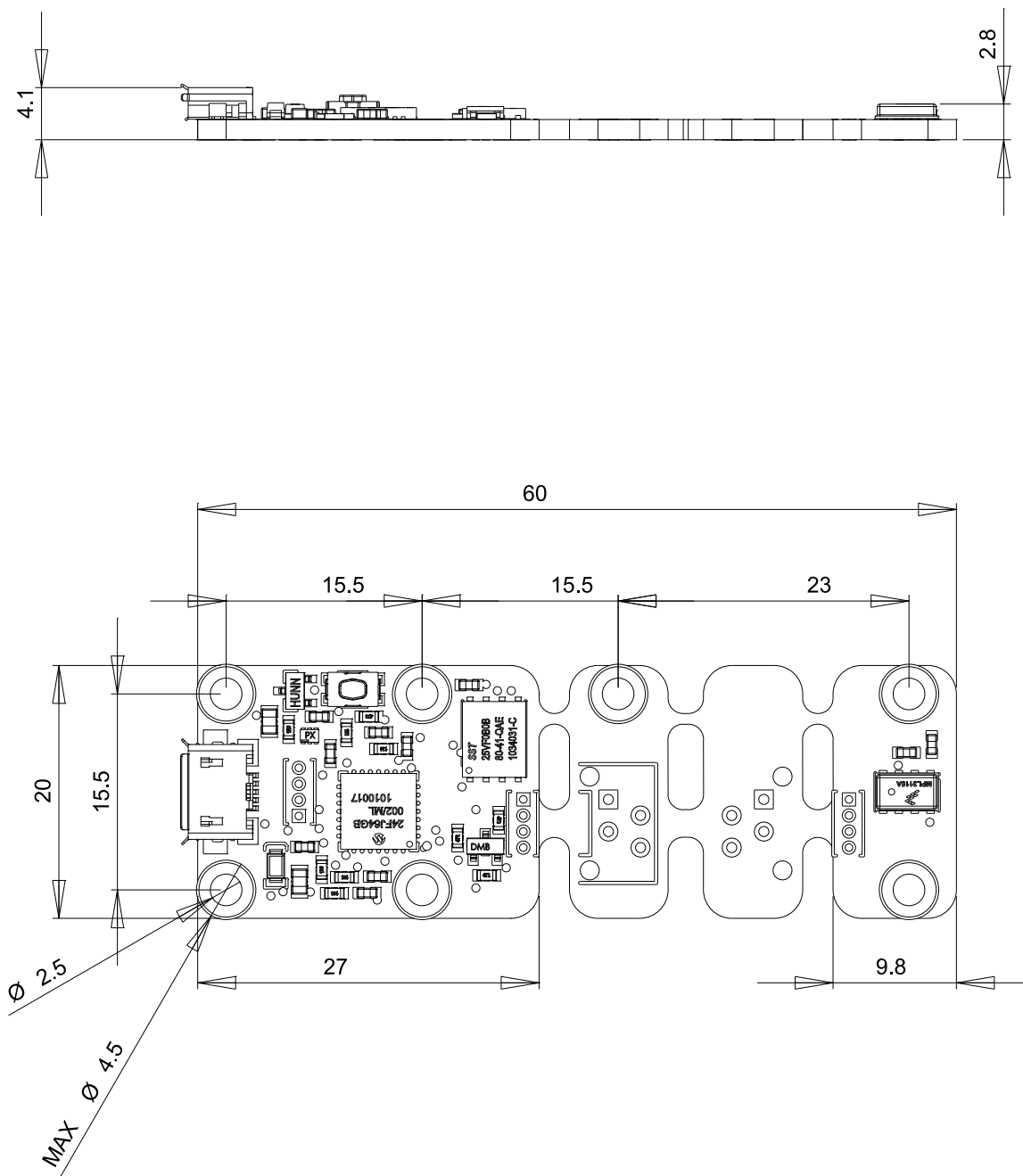
⁵ voir: <http://www.yoctopuce.com/FR/article/cables-usb-la-taille-compte>

⁶ voir: <http://www.yoctopuce.com/FR/article/combien-de-capteurs-usb-peut-on-connecter>

24. Caractéristiques

Vous trouverez résumées ci dessous les principales caractéristiques techniques de votre module Yocto-Altimeter

Connecteur USB	micro-B
Largeur	20 mm
Longueur	60 mm
Poids	4 g
Senseur	MPL3115A2
Fréquence de rafraîchissement	5 Hz
Plage de mesure	0...11500 m
Plage de mesure (P)	200...1100 mbar
Sensibilité	25 cm
Classe de protection IEC	classe III
Temp. de fonctionnement normale	5...40 °C
Temp. de fonctionnement étendue	-10...85 °C
Système d'exploitation supportés	Windows (PC + IoT), Linux (Intel + ARM), macOS, Android
Drivers	Fonctionne sans driver
API / SDK / Librairie	C++, Objective-C, C#, VB.NET, UWP, Delphi, Python, Java, Android
API / SDK / Librairie (seul.TCP)	Javascript, Node.js, PHP
RoHS	Oui
USB Vendor ID	0x24E0
USB Device ID	0x003F
Boîtier recommandé	YoctoBox-Long-Thin-Black-Vents



All dimensions are in mm
Toutes les dimensions sont en mm

Yocto-Altimeter

A4

Scale
2:1
Echelle