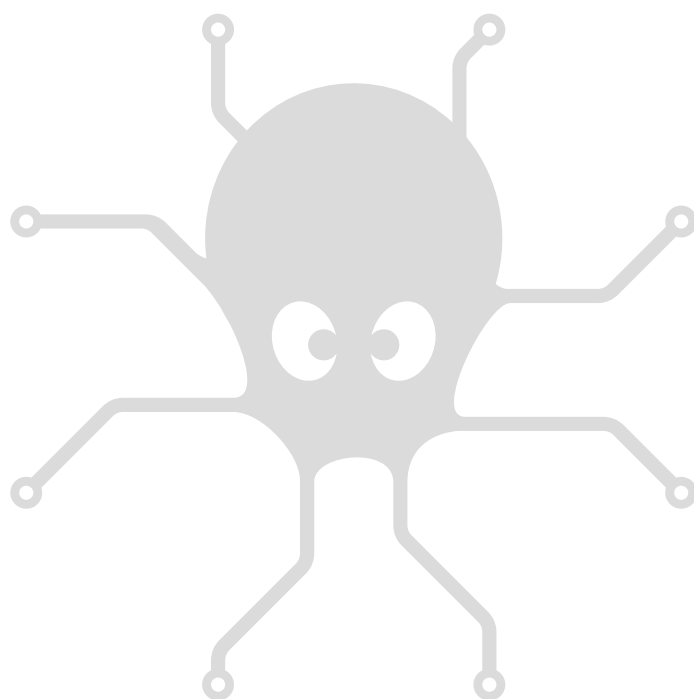




Yocto-MaxiBridge-C

Mode d'emploi

Table des matières

1. Introduction	1
1.1. Informations de sécurité	2
1.2. Conditions environnementales	3
2. Présentation	5
2.1. Les éléments communs	5
2.2. Les éléments spécifiques	7
2.3. Accessoires optionnels	8
3. Mise en place d'une mesure ratiométrique de précision	11
3.1. Conception mécanique	11
3.2. Choix du capteur	12
3.3. Câblage	13
3.4. Configuration logicielle	13
3.5. Correction de la dérive en température	14
4. Premiers pas	17
4.1. Prérequis	17
4.2. Test de la connectivité USB	19
4.3. Localisation	19
4.4. Test du module	19
4.5. Configuration	20
5. Montage et connectique	23
5.1. Fixation	23
5.2. Contraintes d'alimentation par USB	24
5.3. Compatibilité électromagnétique (EMI)	24
6. Programmation, concepts généraux	27
6.1. Paradigme de programmation	27
6.2. Le module Yocto-MaxiBridge-C	29
6.3. Module	31
6.4. MultiCellWeighScale	32
6.5. WeighScale	34

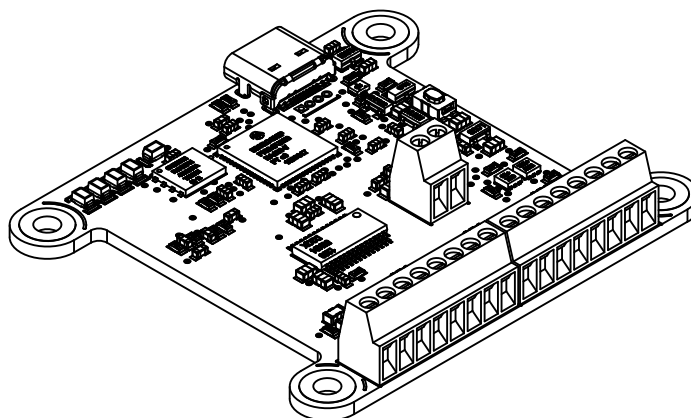
6.6. <i>GenericSensor</i>	36
6.7. <i>Temperature</i>	37
6.8. <i>DataLogger</i>	39
6.9. <i>Quelle interface: Native, DLL ou Service?</i>	40
6.10. <i>Accéder aux modules à travers un hub</i>	42
6.11. <i>Programmation, par où commencer?</i>	43
7. Utilisation du Yocto-MaxiBridge-C en ligne de commande	45
7.1. <i>Installation</i>	45
7.2. <i>Utilisation: description générale</i>	45
7.3. <i>Contrôle de la fonction MultiCellWeighScale</i>	46
7.4. <i>Contrôle de la partie module</i>	47
7.5. <i>Limitations</i>	47
8. Utilisation du Yocto-MaxiBridge-C en Python	49
8.1. <i>Fichiers sources</i>	49
8.2. <i>Librairie dynamique</i>	49
8.3. <i>Contrôle de la fonction MultiCellWeighScale</i>	49
8.4. <i>Contrôle de la partie module</i>	52
8.5. <i>Gestion des erreurs</i>	54
9. Utilisation du Yocto-MaxiBridge-C en C++	55
9.1. <i>Contrôle de la fonction MultiCellWeighScale</i>	55
9.2. <i>Contrôle de la partie module</i>	58
9.3. <i>Gestion des erreurs</i>	60
9.4. <i>Intégration de la librairie Yoctopuce en C++</i>	61
10. Utilisation du Yocto-MaxiBridge-C en C#	63
10.1. <i>Installation</i>	63
10.2. <i>Utilisation l'API yoctopuce dans un projet Visual C#</i>	63
10.3. <i>Contrôle de la fonction MultiCellWeighScale</i>	64
10.4. <i>Contrôle de la partie module</i>	66
10.5. <i>Gestion des erreurs</i>	69
11. Utilisation du Yocto-MaxiBridge-C avec LabVIEW	71
11.1. <i>Architecture</i>	71
11.2. <i>Compatibilité</i>	72
11.3. <i>Installation</i>	72
11.4. <i>Présentation des VIs Yoctopuce</i>	77
11.5. <i>Fonctionnement et utilisation des VIs</i>	80
11.6. <i>Utilisation des objets</i>	82
11.7. <i>Gestion du datalogger</i>	84
11.8. <i>Énumération de fonctions</i>	85
11.9. <i>Un mot sur les performances</i>	86
11.10. <i>Un exemple complet de programme LabVIEW</i>	86
11.11. <i>Différences avec les autres API Yoctopuce</i>	87
12. Utilisation du Yocto-MaxiBridge-C en Java	89
12.1. <i>Préparation</i>	89
12.2. <i>Contrôle de la fonction MultiCellWeighScale</i>	89
12.3. <i>Contrôle de la partie module</i>	91
12.4. <i>Gestion des erreurs</i>	94

13. Utilisation du Yocto-MaxiBridge-C avec Android	95
13.1. Accès Natif et VirtualHub	95
13.2. Préparation	95
13.3. Compatibilité	95
13.4. Activer le port USB sous Android	96
13.5. Contrôle de la fonction MultiCellWeighScale	97
13.6. Contrôle de la partie module	100
13.7. Gestion des erreurs	105
14. Utilisation du Yocto-MaxiBridge-C en TypeScript	107
14.1. Utiliser la librairie Yoctopuce pour TypeScript	108
14.2. Petit rappel sur les fonctions asynchrones en JavaScript	108
14.3. Contrôle de la fonction MultiCellWeighScale	109
14.4. Contrôle de la partie module	113
14.5. Gestion des erreurs	115
15. Utilisation du Yocto-MaxiBridge-C en JavaScript / EcmaScript	117
15.1. Fonctions bloquantes et fonctions asynchrones en JavaScript	118
15.2. Utiliser la librairie Yoctopuce pour JavaScript / EcmaScript 2017	119
15.3. Contrôle de la fonction MultiCellWeighScale	121
15.4. Contrôle de la partie module	125
15.5. Gestion des erreurs	127
16. Utilisation du Yocto-MaxiBridge-C en PHP	129
16.1. Préparation	129
16.2. Contrôle de la fonction MultiCellWeighScale	130
16.3. Contrôle de la partie module	132
16.4. API par callback HTTP et filtres NAT	134
16.5. Gestion des erreurs	138
17. Utilisation du Yocto-MaxiBridge-C en VisualBasic .NET	141
17.1. Installation	141
17.2. Utilisation l'API yoctopuce dans un projet Visual Basic	141
17.3. Contrôle de la fonction MultiCellWeighScale	142
17.4. Contrôle de la partie module	144
17.5. Gestion des erreurs	146
18. Utilisation du Yocto-MaxiBridge-C en Delphi / Lazarus	149
18.1. Préparation	149
18.2. Contrôle de la fonction MultiCellWeighScale	150
18.3. Contrôle de la partie module	152
18.4. Gestion des erreurs	155
19. Utilisation du Yocto-MaxiBridge-C avec Universal Windows Platform ..	157
19.1. Fonctions bloquantes et fonctions asynchrones	157
19.2. Installation	158
19.3. Utilisation l'API Yoctopuce dans un projet Visual Studio	158
19.4. Contrôle de la fonction MultiCellWeighScale	159
19.5. Un exemple concret	160
19.6. Contrôle de la partie module	161

19.7. Gestion des erreurs	163
20. Utilisation du Yocto-MaxiBridge-C en Objective-	165
20.1. Contrôle de la fonction <i>MultiCellWeighScale</i>	165
20.2. Contrôle de la partie module	168
20.3. Gestion des erreurs	170
21. Utilisation avec des langages non supportés	173
21.1. Utilisation en ligne de commande	173
21.2. <i>Assembly .NET</i>	173
21.3. <i>Virtual Hub</i> et <i>HTTP GET</i>	175
21.4. Utilisation des librairies dynamiques	177
21.5. Port de la librairie haut niveau	180
22. Programmation avancée	181
22.1. Programmation par événements	181
22.2. L'enregistreur de données	184
22.3. Calibration des senseurs	187
23. Mise à jour du firmware	191
23.1. Le <i>VirtualHub</i> ou le <i>YoctoHub</i>	191
23.2. La librairie ligne de commandes	191
23.3. L'application <i>Android Yocto-Firmware</i>	191
23.4. La librairie de programmation	192
23.5. Le mode "mise à jour"	194
24. Référence de l'API de haut niveau	195
24.1. La classe <i>YAPI</i>	196
24.2. La classe <i>YModule</i>	200
24.3. La classe <i>YMultiCellWeighScale</i>	206
24.4. La classe <i>YWeighScale</i>	214
24.5. La classe <i>YGenericSensor</i>	222
24.6. La classe <i>YTemperature</i>	230
24.7. La classe <i>YDataLogger</i>	237
24.8. La classe <i>YDataSet</i>	242
24.9. La classe <i>YMeasure</i>	245
24.10. La classe <i>YDataStream</i>	246
25. Problèmes courants	249
25.1. Par où commencer ?	249
25.2. Linux et USB	249
25.3. Plateformes ARM: HF et EL	250
25.4. Les exemples de programmation n'ont pas l'air de marcher	250
25.5. Module alimenté mais invisible pour l'OS	250
25.6. <i>Another process named xxx is already using yAPI</i>	250
25.7. Déconnexions, comportement erratique	251
25.8. Le module ne marche plus après une mise à jour ratée	251
25.9. <i>RegisterHub</i> d'une instance de <i>VirtualHub</i> déconnecte la précédente	251
25.10. Commandes ignorées	251
25.11. Module endommagé	251
26. Caractéristiques	253

1. Introduction

Le Yocto-MaxiBridge-C est un module électronique de 50x58.3mm qui permet de lire précisément jusqu'à quatre cellules de poids, cellules de force ou autre capteur avec une sortie ratiométrique en pont de Wheatstone. Il permet typiquement de fabriquer une balance connectée par USB ou par le réseau. Le Yocto-MaxiBridge-C utilise un convertisseur analogique/numérique de précision pour lire le signal très faible retourné par ce genre de capteurs. Il inclut les filtres nécessaires à la suppression des bruits électriques courants, et peut même appliquer automatiquement des corrections pour contrer la dérive en température inhérente aux cellules de charge, grâce à un dispositif de mesure de température interne ou externe. Le Yocto-MaxiBridge-C peut alimenter la cellule ou le pont avec une tension régulée de 4.7V, continue ou alternée, compatible avec la plupart des cellules de poids. Il est néanmoins aussi possible d'utiliser une source d'excitation externe si désiré.



Le module Yocto-MaxiBridge-C

Le Yocto-MaxiBridge-C 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-MaxiBridge-C et espère sincèrement qu'il vous donnera entière satisfaction. Les ingénieurs Yoctopuce se sont donné beaucoup de mal pour que votre Yocto-MaxiBridge-C 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:	Route de Cartigny 33
Localité:	1236 Cartigny
Pays:	Suisse

1.1. Informations de sécurité

Le Yocto-MaxiBridge-C 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

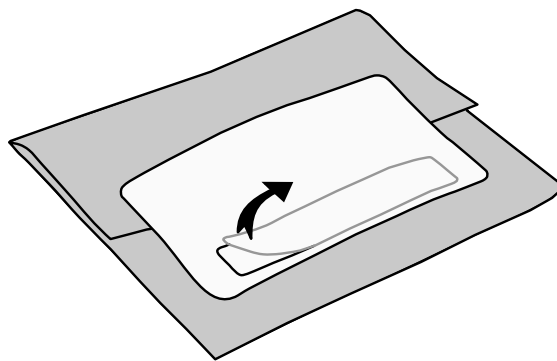
Le Yocto-MaxiBridge-C ne doit pas être utilisé sans boîtier de protection, en raison des composants électriques à nu. Pour une sécurité optimale, il devrait être mis dans un boîtier non métallique, non-inflammable, résistant à un choc de 5 J, par exemple en polycarbonate (LEXAN ou autre) d'indice de 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-MaxiBridge-C, fournie avec le circuit électronique, à proximité immédiate du module. Si le module est dans un boîtier dédié, l'étiquette devrait être collée sur la surface extérieur du boîtier. L'étiquette est résistante à l'humidité et au frottement usuel qui peut survenir durant un entretien normal.



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

Applications

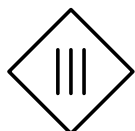
La norme de sécurité vérifiée correspond aux instruments de laboratoire, pour le contrôle de procédés industriels, ou pour des applications similaires en milieu résidentiel ou commercial. Si vous comptez utiliser le Yocto-MaxiBridge-C pour un autre type d'applications, vous devrez vérifier les critères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

En particulier, le Yocto-MaxiBridge-C 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-MaxiBridge-C n'est *pas* certifié pour utilisation dans les zones dangereuses, ni pour les environnements explosifs. Les conditions environnementales assignées sont décrites ci-dessous.

Classe de protection III (IEC 61140)



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

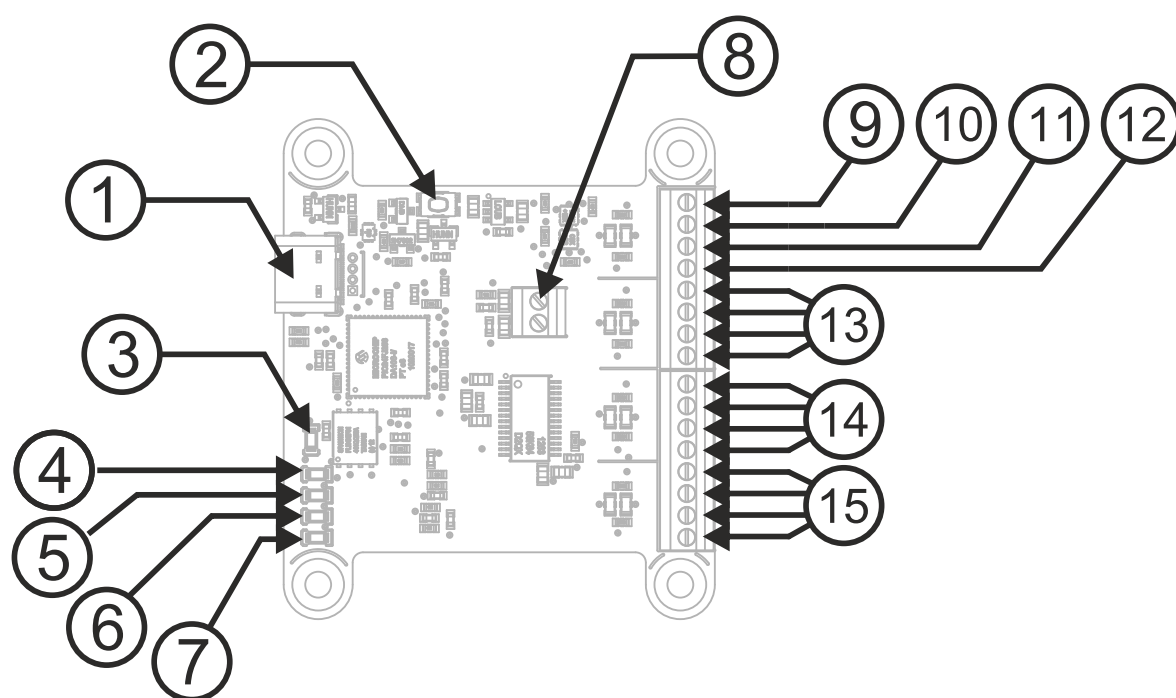
1.2. Conditions environnementales

Les produits Yoctopuce sont conçus pour une utilisation intérieure dans un environnement usuel de bureau ou de laboratoire (*degré de pollution* 2 selon IEC 60664): la pollution de l'air doit être faible et essentiellement non conductrice. L'humidité relative prévue est de 10% à 90% RH, sans condensation. L'utilisation dans un environnement avec une pollution solide ou conductrice significative exige de protéger le module contre cette pollution par un boîtier certifié IP67 ou IP68. Les produits Yoctopuce 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-MaxiBridge-C est -30...85°C. Cette plage de température a été déterminée en fonction des recommandations officielles des fabricants des composants utilisés dans le Yocto-MaxiBridge-C, et par des tests de durée limitée (1h) dans les conditions extrêmes, en environnement contrôlé. Si vous envisagez d'utiliser le Yocto-MaxiBridge-C 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-C
- 2: Yocto-bouton
- 3: Yocto-led
- 4: Led Canal 1
- 5: Led Canal 2
- 6: Led Canal 3
- 7: Led Canal 4
- 8: Connecteur Sonde temperature
- 9: Canal 1, Excitation +
- 10: Canal 1, Signal +
- 11: Canal 1, Signal -
- 12: Canal 1, Excitation -
- 13: Canal 2
- 14: Canal 3
- 15: Canal 4

2.1. Les éléments communs

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

Le connecteur USB

Le Yocto-MaxiBridge-C est équipé d'une connectique USB au format type-C mais il utilise le protocole USB 1.1, il est donc compatible avec n'importe quel hôte USB. Alternativement vous pouvez souder un câble USB ou un connecteur au pas 1.27mm directement dans les trous prévus à cet effet, derrière le connecteur USB-C.

Si, lorsque que vous branchez votre Yocto-MaxiBridge-C au bout d'un câble USB-C, votre module s'allume mais n'est pas détecté par votre ordinateur, vérifiez que la fiche USB-C est bien enfoncée et que vous utilisez un câble USB normal et non pas un câble de charge sans lignes de données.

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

- **Tension min.:** 4.75 V DC
- **Tension max.:** 5.25 V DC
- **Protection contre les surintensités:** max. 5.0 A

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

Le Yocto-bouton

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

La Yocto-Led

En temps normal la Yocto-Led sert à indiquer le bon fonctionnement du module: elle émet alors une faible lumière bleue qui varie lentement mimant ainsi une respiration. La Yocto-Led cesse de respirer lorsque le module ne communique plus, par exemple s'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-MaxiBridge-C ce numéro commence par WMBRDG1C. Le module peut être piloté par logiciel en utilisant ce numéro de série. Ce numéro de série ne peut pas être changé.

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

² support@yoctopuce.com

Le nom logique

Le nom logique est similaire au numéro de série, c'est une chaîne de caractères 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 module à l'aide de son nom logique. A leur sortie d'usine, les modules n'ont pas de nom logique assigné, c'est à vous de le définir.

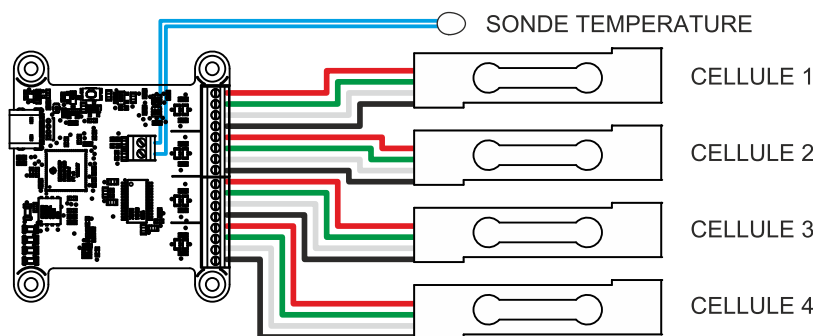
2.2. Les éléments spécifiques

Les borniers de mesure ratiométrique

Le Yocto-MaxiBridge-C dispose de quatre entrées ratiométriques pour y brancher des cellules de poids, cellules de charge ou autre capteur avec une sortie en pont de Wheatstone.

Le principe d'une mesure ratiométrique consiste à envoyer une tension d'excitation sur le capteur, et à mesurer la proportion de tension mesurée entre les fils intermédiaires. La valeur mesurée est donc donnée par le rapport entre la tension observée sur les fils intermédiaires et la tension d'excitation. Par exemple, une cellule de poids peut typiquement générer 2 [mV/V] à charge maximale, c'est à dire 2 mV par Volt d'excitation reçu.

Le Yocto-MaxiBridge-C peut être configuré pour fournir lui-même la tension d'excitation (4.7V). C'est en général la meilleure option, car il peut dans ce cas procéder à une alternance de polarité d'excitation, ce qui permet d'annuler certaines erreurs d'offsets introduites par les effets thermocouples dans le système. Dans ce cas, les cellules doivent être connectées comme suit:



Connection des cellules de poids au Yocto-MaxiBridge-C

Dans les rares cas où une excitation par une tension externe est désirée, elle doit alors être continue et ne pas dépasser la tension d'alimentation du module (5V). Notez que même si certaines cellules supportent une alimentation jusqu'à 15V, il est en général inutile de les alimenter avec une tension aussi haute et l'on obtient des mesures plus précises avec un capteur bien filtré à basse tension qu'avec un capteur médiocre et une cellule excitée au maximum.

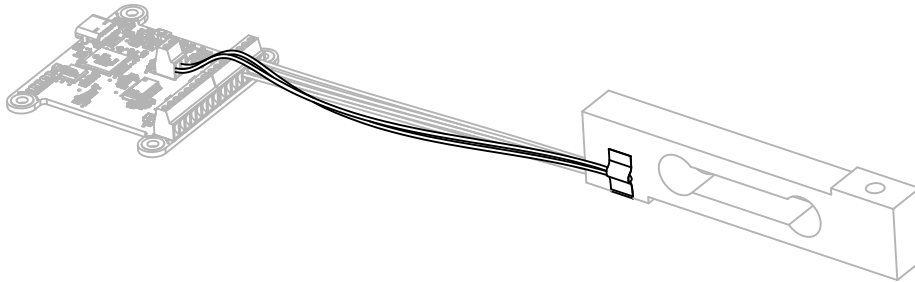
Le circuit de mesure est un circuit de très basse tension de sécurité (TBTS). Il ne doit pas être connecté à des tensions supérieures à 5V, ni être mis en commun avec un circuit d'alimentation réseau.

Utilisez des fils aussi courts que possible entre la cellule de charge et le Yocto-MaxiBridge-C, idéalement 50cm ou moins. L'utilisation de longs fils diminue la précision du système. Le module n'a été ni conçu ni testé pour des fils de plus de 3m.

Le Yocto-MaxiBridge-C effectue sa mesure à l'aide d'un convertisseur analogique/numérique 32 bits AD1263 de Texas Instruments. Un filtre à réponse impulsionnelle finie incorporé permet un excellent filtrage des perturbations 50Hz et 60Hz. Le circuit du Yocto-MaxiBridge-C implémente les recommandations de Texas Instruments pour réduire toutes les perturbations pouvant affecter les mesures, y compris les effets de la température sur l'électronique.

La sonde de température externe

Le Yocto-MaxiBridge-C est capable d'appliquer automatiquement une correction linéaire par morceau pour compenser la dérive en température des cellules de poids elles-même. Pour ce faire, le module dispose de deux dispositifs alternatifs pour mesurer la température. Pour les cas où le Yocto-MaxiBridge-C est situé exactement au même endroit que les cellules de poids, on peut utiliser la sonde de température interne du convertisseur AD1263. Pour une mesure plus directement liée aux cellules de poids elles-même, on peut fixer un thermistor NTC directement contre une cellule de poids.



Utilisation d'un thermistor avec une cellule de poids

Votre Yocto-MaxiBridge-C est livré avec un thermistor NTC NXFT15XH103 fabriqués par Murata. Il a une résistance nominale de 10K Ω à 25°C et un coefficient β nominal de 3380 entre 25 et 50°C. Vous pouvez l'utiliser sur la plage de température standard, et même jusqu'à 0°C. Si vous désirez mesurer des températures inférieures à 0°C, vous devriez vous procurer un autre thermistor dont la résistance ne dépasse pas 30K Ω à la température la plus froide.

Le circuit de mesure de la sonde de température externe est un circuit de très basse tension de sécurité (TBTS). Il ne doit pas être connecté à une quelconque source de tension, mais uniquement être raccordé à un thermistor. Il ne doit en aucun cas être mis en commun avec un circuit d'alimentation réseau.

Au besoin, vous pouvez rallonger les fils du thermistor fourni avec votre Yocto-MaxiBridge-C, la différence de précision ne sera marginale. Cependant n'oubliez pas que les fils entre les cellules de poids et le Yocto-MaxiBridge-C sont censés être aussi courts que possible.

Les LEDs témoin (verte)

Le Yocto-MaxiBridge-C dispose de quatre entrées ratiométriques qui sont mesurées l'une après l'autre. Chaque entrée du Yocto-MaxiBridge-C est dotée d'une LED témoin verte, qui s'allume lorsque l'entrée correspondante est mesurée. Si les quatre entrées ne sont pas utilisées, vous pouvez réduire le nombre d'entrées à parcourir dans la configuration du module.

2.3. Accessoires optionnels

Les accessoires ci-dessous ne sont pas nécessaires à l'utilisation du module Yocto-MaxiBridge-C, 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-MaxiBridge-C à un support, vous pouvez placer des petites vis de 3mm avec une tête de 8mm au maximum dans les trous prévus. 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 multi-TT

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-MaxiBridge-C 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 l'application VirtualHub³.

Connecteurs 1.27mm (ou 1.25mm)

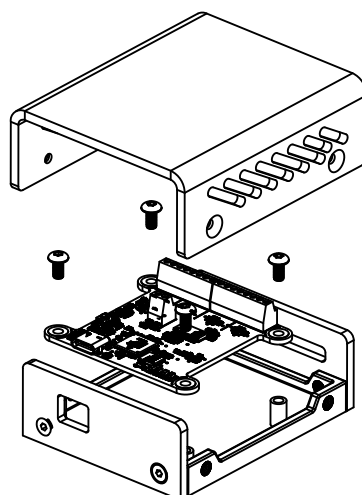
Si vous désirez raccorder le module Yocto-MaxiBridge-C à un Micro-hub USB ou à un YoctoHub en évitant l'encombrement d'un vrai câble USB, vous pouvez utiliser les 4 pads au pas 1.27mm juste derrière le connecteur USB. Vous avez alors deux possibilités.

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-MaxiBridge-C.

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-MaxiBridge-C a été conçu pour pouvoir être installé tel quel dans votre projet. Néanmoins Yoctopuce commercialise des boîtiers spécialement conçus pour les modules Yoctopuce. Vous trouverez plus d'informations à propos de ces boîtiers sur le site de Yoctopuce⁴. Le boîtier recommandé pour votre Yocto-MaxiBridge-C est le modèle YoctoBox-MaxiIO-Transp



Vous pouvez installer votre Yocto-MaxiBridge-C dans un boîtier optionnel.

³ <http://www.yoctopuce.com/FR/virtualhub.php>

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

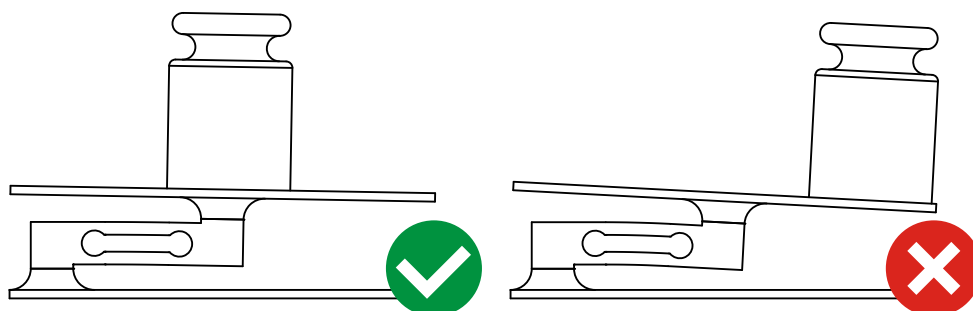
3. Mise en place d'une mesure ratiométrique de précision

Ce chapitre vous expliquera pas à pas les différentes étapes pour la mise en place d'une mesure ratiométrique de précision, en prenant l'exemple le plus courant et le plus complet de l'utilisation de cellules de poids pour fabriquer une balance connectée.

3.1. Conception mécanique

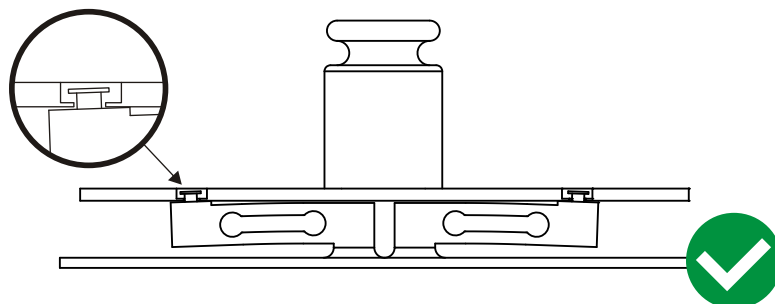
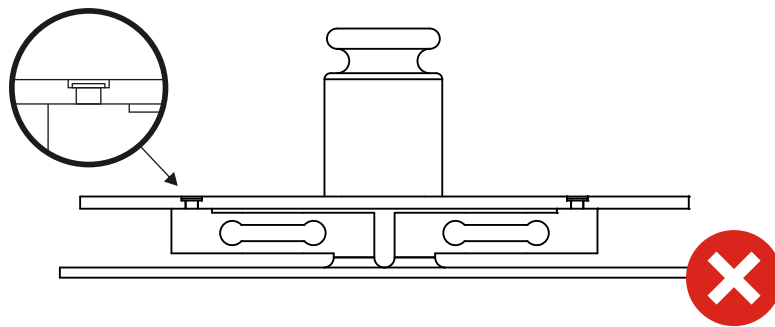
Pour commencer, il faut se souvenir qu'une balance mesure le poids d'un objet en mesurant la déformation des cellules de charge causée par le poids de l'objet. Le plateau de pesée est donc un objet mobile, même si le mouvement paraît infime, et la disposition des cellules de charge doit être faite de sorte à ce que le plateau de pesée puisse bouger sans entraves.

Le système le plus simple lorsque l'objet à mesurer est petit et pas trop lourd consiste à utiliser une seule cellule de charge, sur laquelle est directement fixé le plateau de pesée. C'est un système utilisé dans nombreuses petites balances de cuisine. Il faut par contre être conscient que la pesée sera faussée si les objets ne sont pas centrés sur le plateau.



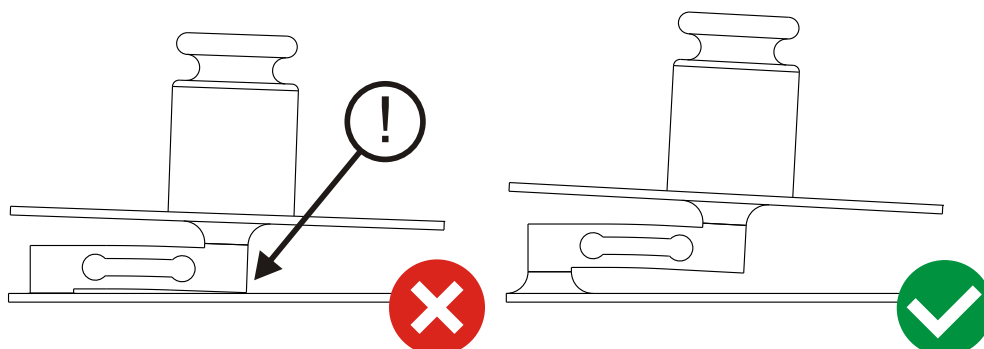
Il est préférable de centrer la charge sur le plateau

Un autre système simple à construire consiste à mettre quatre cellules de poids, une à chaque angle du plateau de pesée, et à sommer leurs mesures. Ceci peut d'ailleurs être fait automatiquement par le Yocto-MaxiBridge. Si vous utilisez ce système, prenez garde à orienter les cellules de sorte à ce que, lorsqu'elles sont en charge, la torsion des cellules puisse se faire librement.



La fixation du plateau ne doit pas entraver la déformation des cellules

Quelle que soit la géométrie choisie, prenez garde à ce que l'extrémité de la cellule de poids puisse bouger librement, même à pleine charge. En particulier, sachez que l'extrémité mobile de certaines cellules de poids rectangulaires descend parfois plus bas que le point de fixation.



Prenez garde à ce que la cellule ne soit pas bloquée à pleine charge.

Les mesures d'une balance dépendent fortement de son inclinaison. Les balances commerciales sont calibrées en usine pour fonctionner en étant parfaitement à l'horizontal. Une balance basée sur un Yocto-MaxiBridge-C fonctionnera parfaitement si elle est posée sur une surface en pente, à condition d'être non seulement re-tarée, mais aussi recalibrée, à chaque fois que son inclinaison est changée.

Les vibrations, même imperceptibles, peuvent avoir une forte influence sur une cellule de charge. Tenez en compte si vous souhaitez fabriquer une balance de précision.

3.2. Choix du capteur

L'étape suivante consiste à choisir de bonnes cellules de charge. En effet, même si certaines dérives peuvent être compensées *a posteriori* par des corrections logicielles, rien ne vaut un bon capteur pour obtenir des bonnes mesures. Méfiez-vous donc des cellules de poids à moins de \$10 achetées chez un revendeur inconnu à l'autre bout du monde, il y a probablement une raison pour laquelle elles sont vendues si bon marché.

La plupart des cellules de charge incluent un premier niveau de compensation de la dérive en température, mais la qualité de cette compensation dépend énormément de l'effort qui a été fait pour la calibrer. Une mauvaise compensation peut même empirer la dérive en température.

Toute cellule de poids est normalement livrée avec un certificat de calibration qui documente ses caractéristiques. La première donnée importante est l'amplitude du signal pour le poids maximal (*full scale output* en anglais). Évitez les cellules avec un signal de moins de $\sim 2\text{mV/V}$ si vous voulez une mesure précise. Ensuite, regardez les facteur d'erreur documentés de la cellule. La plupart sont donnés en proportion de la valeur maximale mesurée (*full scale*, abrégé *FS*). Par exemple, pour une cellule de poids donnée pour 30Kg, une erreur de 0.05%FS correspond à 15 grammes d'erreur. Ce n'est pas énorme sur le poids total, mais lorsque c'est une erreur d'offset, cela affecte toutes les mesures, même les plus basses. Une cellule décentement compensée en température ne devrait pas avoir une variation du zéro en température supérieure à 0.02%FS/10°C.

Notez qu'on trouve parfois dans certaines balances bon marché des capteurs de force en demi-pont (trois fils au lieu de quatre) à chaque coin du plateau de pesée, qui sont combinées pour former un pont entier unique qui peut être mesuré avec une seule mesure ratiométrique. Bien que théoriquement intéressante, cette variante est à proscrire pour une balance de précision car les caractéristiques des quatre demi-ponts ne sont jamais suffisamment identiques pour obtenir des résultats précis et stables en température.

3.3. Câblage

Une bonne cellule de poids est normalement fournie avec un câble gainé dans laquelle les fils sont torsadés, pour limiter les interférences électromagnétiques. Cependant, n'hésitez pas à raccourcir le câble à la taille strictement nécessaire pour votre application, car plus il sera court, meilleure sera votre mesure. En effet, en plus de faire antenne, un long câble présente aussi le défaut de faire office de thermistor, et donc d'introduire une erreur variant en fonction de la température.

La connexion entre les cellules de poids sur le Yocto-MaxiBridge-C doit se faire comme décrit au chapitre précédent. En cas d'inversion des deux fils d'excitation ou des deux fils de mesure, la balance fonctionnera, mais à l'envers (c'est-à-dire qu'elle indiquera des poids négatifs).

3.4. Configuration logicielle

Vous trouverez des indication détaillées sur la manière d'accéder à la fenêtre de configuration du Yocto-MaxiBridge-C au chapitre suivant. Le présent chapitre porte lui sur les choix que vous devez faire au moment de la conception de votre système et sur la procédure de calibration.

Excitation

Pour obtenir une mesure de vos cellules de charges, vous devez activer une tension d'excitation. Vous avez la possibilité d'utiliser une tension externe jusqu'à 5V si vous le désirez, mais la manière recommandée est d'utiliser l'excitation alternée de 4.7V produite directement par le Yocto-MaxiBridge-C. L'intérêt d'une excitation en polarité alternée est qu'elle annule par moyennage les éventuels décalages de mesure causés par divers phénomènes thermoélectrique sur le circuit électronique¹

Tare initiale

Une fois la ou les cellules sous tension d'excitation, vous pourrez en obtenir des mesures. Mais il faudra toujours commencer par tarer la mesure à vide, c'est-à-dire enregistrer dans le Yocto-MaxiBridge-C la constante de décalage à soustraire pour la cellule. Pour tarer une cellule de charge, il suffit de presser sur le bouton correspondant dans l'interface du Yocto-MaxiBridge-C.

Calibration de la plage de mesure

Il faut ensuite indiquer au Yocto-MaxiBridge-C la correspondance entre la mesure ratiométrique et un poids physique mesuré. Vous pourriez pour cela utiliser la valeur théorique qui figure sur le certificat de calibration, mais il est plus précis de mesurer une valeur exacte en tenant compte de tous les système de mesure à l'aide d'un poids étalon. Posez donc un poids étalon sur la cellule de charge que vous voulez calibrer, et indiquez dans l'interface du Yocto-MaxiBridge-C la valeur du poids

¹ voir l'article de Christopher Hall intitulé "High-Resolution, Low-Drift, Precision Weigh-Scale Reference Design with AC Bridge Excitation" dans la série TI Designs - Precision: <http://www.ti.com/lit/ug/tiduac1/tiduac1.pdf>

étalon et la valeur maximale que la cellule de charge peut mesurer. Sur la base de ces informations, le Yocto-MaxiBridge-C pourra choisir la valeur optimale de l'amplificateur de gain interne et les facteurs multiplicatifs à appliquer.

Après la première calibration de la plage de mesure, il est recommandé de répéter l'opération de tare une seconde fois, car dans le cas où le réglage de l'amplificateur de gain interne aurait été changé, cela peut produire un léger décalage du zéro. La calibration de la plage avec le poids étalon doit ensuite être répétée elle-aussi pour s'assurer que les facteurs multiplicatifs tiennent compte de la tare corrigée.

Ces opérations de calibration devraient dans l'idéal être faits dans un pièce à température stable, en s'assurant aussi que la cellule de charge est à la température ambiante dès le début des mesures. Pour les meilleurs résultats, il faut laisser à la cellule de charge le temps de s'adapter aux changements de poids, avant de presser sur le bouton qui sauvegarde les réglages. Si vous observez que la valeur ne se stabilise pas, c'est probablement que votre cellule est soit de mauvaise qualité, soit que la température ambiante n'est pas stable.

Si vous construisez une balance faite de plusieurs cellules, il est recommandé d'effectuer la calibration initiale cellule par cellule pour obtenir la meilleure précision. Par contre, une fois la calibration initiale des cellules effectuée, vous pouvez fixer le plateau de pesée sur toutes les cellules et effectuer les tares ultérieures simultanément sur toutes les cellules sans risquer de perdre en précision.

3.5. Correction de la dérive en température

Les cellules de poids ont la malheureuse habitude de dériver avec les changements de température. Bien qu'elles disposent quasiment toutes d'un dispositif de correction à base de résistances thermosensibles, il ne compense jamais exactement la dérive, et l'erreur résultante est donc documentée dans les spécifications de la cellule.

Même lorsque ce n'est que de quelques grammes, voir une balance à vide indiquer des chiffres qui varient tout seul sans raison apparente n'est pas de nature à inspirer confiance. C'est pourquoi la majorité des balances implémentent un mécanisme de compensation logiciel supplémentaire.

Compensation par zero-tracking

La méthode utilisée par les balances domestiques consiste à effectuer une tare automatique à l'enclenchement, pour toujours partir de zéro, puis à appliquer une tare automatique continue à chaque occasion où le poids mesuré est inférieur à un seuil pré-configuré (par exemple 1g). Ainsi, chaque seconde, si la balance mesure un poids de 0,5g, elle se re-tarera à 0g, et mettra cette différence de 0,5g sur le compte d'une dérive en température. Si une seconde plus tard elle mesure à nouveau 0,5g, elle n'affichera pas 1g mais annulera à nouveau cette différence. Ce n'est que lorsque le poids varie de manière plus significative que la balance considère que l'utilisateur a posé un objet sur le plateau de pesée.

C'est une méthode rudimentaire mais assez efficace pour les applications où l'on sait que la balance retourne régulièrement au point zéro. Elle a l'avantage d'être très facilement configurée. Vous pouvez l'activer dans le Yocto-MaxiBridge-C en configurant simplement le seuil de *zero-tracking*. Si vous ne voulez pas appliquer cette correction automatique, il suffit de mettre le seuil à zéro et aucune mesure ne sera automatiquement ignorée.

Compensation par mesure de la température

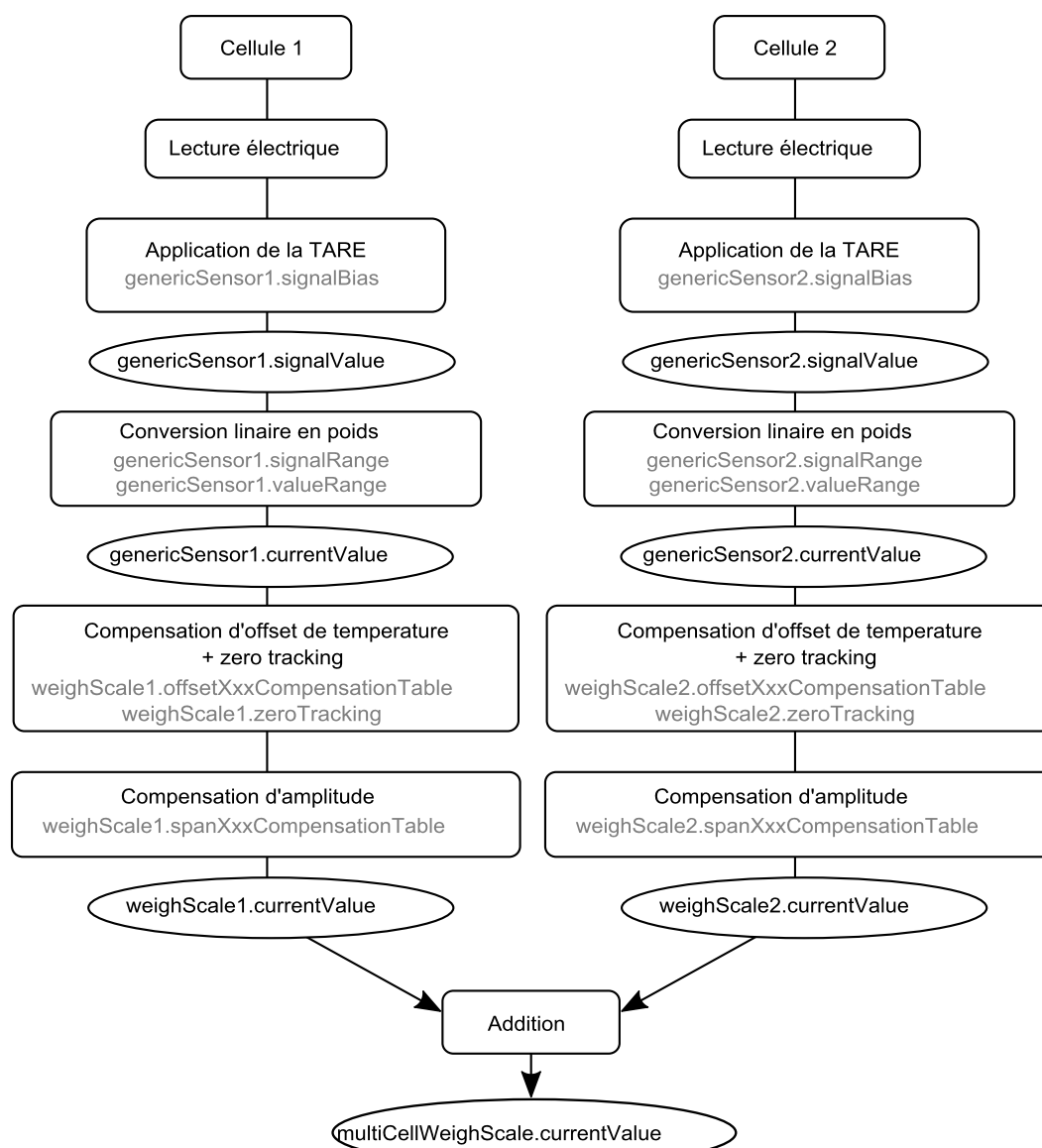
Il existe des applications où la balance ne revient jamais au zéro mais reste en permanence en charge, avec un poids variant progressivement. C'est par exemple le cas des applications où l'on surveille à moyen terme le poids d'un objet affecté par des actions externes, comme une ruche par exemple. Dans ces cas, on peut tenter de compenser l'effet de la dérive de la cellule de poids par une correction logicielle appliquée en fonction d'une mesure de la température ambiante. En effet, si l'on observe l'évolution de la mesure de poids à vide en fonction de la température, on constate une très forte corrélation entre ces deux variables. En pratique, une compensation logicielle peut donc selon les cas permettre une réduction de la dérive en température d'un facteur dix, et ainsi ramener une dérive de plusieurs grammes à un simple bruit inférieur au gramme.

Il existe deux possibilités pour mesurer la température: utiliser la température ambiante sur le Yocto-MaxiBridge-C, ou mesurer directement la température d'une cellule de charge à l'aide d'un thermistor collé contre une cellule. Le Yocto-MaxiBridge-C permet les deux méthodes puisque l'on peut configurer le type de mesure à effectuer, et qu'un connecteur spécifique permet le branchement d'un thermistor externe qui peut facilement être déporté jusqu'à la cellule de charge. Au final, les deux méthodes peuvent donner des bons résultats, ce sera à vous de déterminer la meilleure en fonction de votre application.

La modélisation de la compensation en température est faite de la manière suivante. Le Yocto-MaxiBridge-C détermine toutes les dix secondes une *température de compensation*, basée sur la température ambiante mesurée, mais en appliquant une inertie de propagation définie sous la forme d'un coefficient d'adaptation. Le coefficient standard proposé est de 0.4% d'adaptation de la *température de compensation* chaque 10 secondes, mais ce coefficient peut être changé selon les applications si nécessaire. A partir de cette température de compensation, on définit une correction linéaire par morceau à appliquer à la tare en fonction de la température absolue, mais aussi par rapport à la *variation* de cette température, qui est aussi un facteur significatif des tensions internes qui se produiront dans la cellule lors de changements rapides de température.

Cette modélisation peut paraître complexe à première vue, mais elle se comprend et s'applique relativement aisément par la pratique. En configurant le Yocto-MaxiBridge-C pour enregistrer les mesures de poids à vide et la température mesurée pendant plusieurs jours, on peut facilement observer sur un graphique la corrélation entre les mesures et simuler les corrections idéales à appliquer par rapport à la température de compensation et à la variation de la température de compensation. Une fois les paramètres optimaux obtenus par simulation, ils peuvent être injectés dans le Yocto-MaxiBridge-C comme correction d'*offset*, qui effectue alors la correction de manière autonome.

Une fois la dérive de la tare compensée, il est possible aussi d'employer la même méthode pour une compensation en température du facteur d'échelle, si nécessaire. Pour cela, répétez le test d'enregistrement pendant plusieurs jours de la température sur la balance, chargée d'un poids étalon. Si une corrélation est visible, déterminez là aussi le facteur de correction par simulation et injectez-le dans la correction de *span*.



Etapes de calcul du poids effectuées par le Yocto-MaxiBridge-C pour deux cellules

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-MaxiBridge-C, 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 besoins, 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, Linux, macOS 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.

La règle générale concernant les versions de système d'exploitation supportées est la suivante: les outils de développement Yoctopuce sont supportés pour toutes les versions couvertes par le support de l'éditeur du système d'exploitation, y compris la durée du support étendu (*long term support* ou LTS). Yoctopuce attache une attention particulière au support à long terme, et lorsque c'est possible avec un effort raisonnable, nos outils sont construits de sorte à pouvoir être utilisés sur des anciens systèmes même plusieurs années encore après la fin du support étendu par le fabricant.

De plus, les bibliothèques de programmation pour piloter nos modules étant disponibles en code source, il vous est en général possible de les recompiler pour fonctionner sur des systèmes d'exploitation encore plus anciens. A ce jour, notre bibliothèque de programmation peut toujours être compilée pour fonctionner sur des systèmes d'exploitation publiés en 2008, tels que Windows XP SP3 ou Linux Debian Squeeze.

Les architectures supportées par les bibliothèques logicielles de Yoctopuce sont les suivantes:

- Windows: Intel 64 bits et 32 bits

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

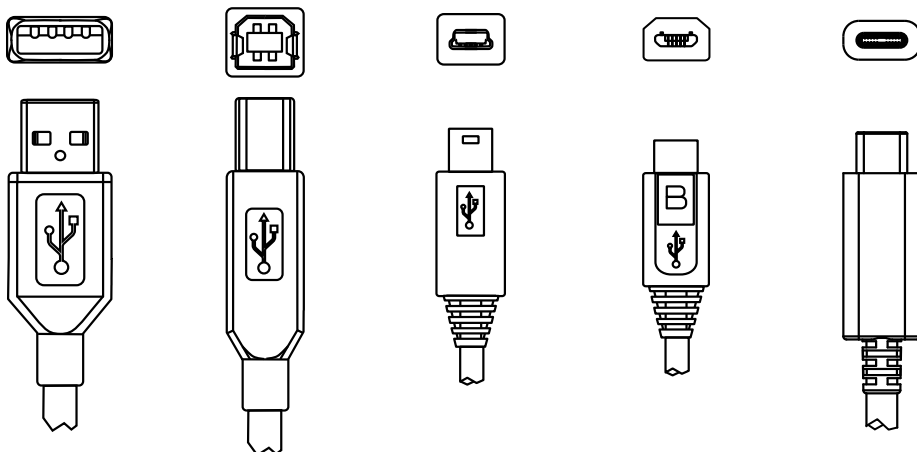
- Linux: Intel 64 bits et 32 bits, ARM 64 bits et 32 bits, y compris Raspberry Pi OS.
- macOS: Intel 64 bits et Apple Silicon (ARM)

Sous Linux, la communication avec nos modules USB requiert impérativement la librairie libusb en version 1.0 ou plus récente, qui est disponible sur toutes les distributions courantes. Les librairies et les outils en ligne de commande devraient pouvoir être facilement recompilés sur n'importe quelle variante d'UNIX (Linux, FreeBSD, ...) datant des quinze dernières années pour laquelle libusb-1.0 est disponible et fonctionnel.

Sous Android, la possibilité de connecter un module USB dépend du fait que la tablette ou le téléphone supporte le *mode USB Host*.

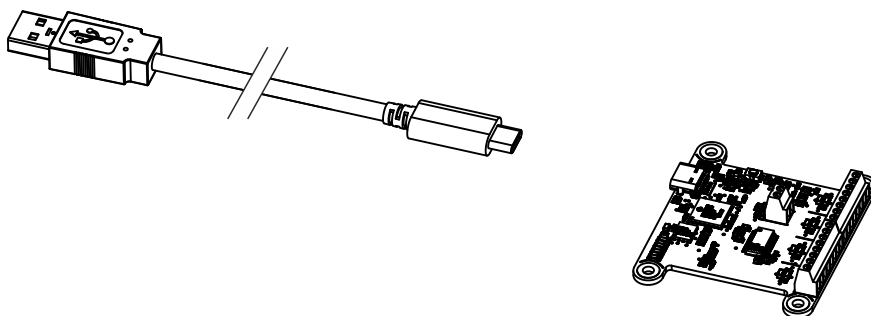
Un cable USB type A-USB-C

Il existe plusieurs formes de connecteurs USB. La taille "normale" correspond à celle que vous utilisez probablement pour brancher votre imprimante. La taille "mini" a plus ou moins disparu. La taille "micro" était la plus petite au moment où les premiers modules Yoctopuce ont été conçus. Depuis quelques années, les connecteurs USB-C sont apparus et sont en passe de supplanter tous les autres. C'est pourquoi, depuis 2024, Yoctopuce a entrepris de migrer progressivement ses produits vers USB-C².



Les connecteurs USB 2.0 les plus courants: A, B, Mini B, Micro B et USB-C.

Pour connecter votre module Yocto-MaxiBridge-C à un ordinateur, vous avez donc besoin d'un cable USB de type A-USB-C ou éventuellement type C- USB-C. Vous trouverez ce cable en vente à des prix très variables selon les sources, sous la dénomination *USB A to USB-C 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-MaxiBridge-C à l'aide d'un cable USB 2.0 de type A - USB-C

Si vous branchez un hub USB entre l'ordinateur et le module Yocto-MaxiBridge-C, 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 détails à ce sujet dans le chapitre concernant le montage et la connectique.

² www.yoctopuce.com/FR/article/etes-vous-interesse-par-usb-c

4.2. Test de la connectivité USB

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

Rendez-vous sur le site de Yoctopuce et téléchargez le programme *VirtualHub*³, Il est disponible pour Windows, Linux et macOS. En temps normal le programme VirtualHub 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 VirtualHub 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
VIRTHUB0-3880db7f12		VirtualHub	configure view log file
WMBRDG1C-2C0597		Yocto-MaxiBridge-C	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-MaxiBridge-C.

³ www.yoctopuce.com/FR/virtualhub.php

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

WMBRDG1C-2C0597 is a 50x60mm board with a precision ratiometric sensor designed to read four Wheatstone bridges, and a temperature sensor.

Module

Serial # WMBRDG1C-2C0597
 Product name: Yocto-MaxiBridge-C
 Logical name:
 Firmware: 64616
 Consumption: 26 mA
 Beacon: Inactive turn on
 Luminosity: 50%

Sensors

	Signal	Weight
Bridge/Cell 1:	0.00005 mV/V	0 g
Bridge/Cell 2:	0.00001 mV/V	0 g
Bridge/Cell 3:	0 mV/V	0 g
Bridge/Cell 4:	-0.00005 mV/V	0 g
Multi-Cell:		0 g

Temperature: 31.9 °C

Misc

[Open API browser](#)
[Get user manual from yoctopuce.com](#)

Close

Propriétés du module Yocto-MaxiBridge-C.

Cette fenêtre vous permet entre autres de jouer avec votre module pour en vérifier son fonctionnement, les valeurs mesurées par le Yocto-MaxiBridge-C y sont en effet affichées en temps réel.

4.5. Configuration

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

WMBRDG1C-2C0597

Edit parameters for device WMBRDG1C-2C0597, and click on the Save button.

Serial # WMBRDG1C-2C0597
 Product name: Yocto-MaxiBridge-C
 Firmware: 64616 upgrade

Export Settings Import Settings

Logical name:
 Luminosity: (signal leds only)

Device functions

Each function of the device has a physical name and a logical name. You can change the logical name using the **rename** button.

WMBRDG1C-2C0597.temperature / rename
 Value: Temperature in °C, from A/D internal sensor configure

WMBRDG1C-2C0597.multiCellWeighScale / rename
 Number of cells: 4
 Use Input4 as external sense: disabled
 Bridge excitation: DC
 Current weight: 0g text
 Display resolution: 0.001 g
 Zero tracking: under 0.3 g
 Weight unit: g

WMBRDG1C-2C0597.weighScale1 / rename
 WMBRDG1C-2C0597.weighScale2 / rename
 WMBRDG1C-2C0597.weighScale3 / rename
 WMBRDG1C-2C0597.weighScale4 / rename
 WMBRDG1C-2C0597.dataLogger / rename

Datalogger and Timed reports configure

Timed reports are disabled
 Recording to flash memory is disabled
 no recorded data

Save Cancel

Configuration du module Yocto-MaxiBridge-C.

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 VirtualHub⁵ en ligne de commande ⁶.

Nom logique du module

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

Luminosité

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

Nom logique des fonctions

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

Les fonctions configurables pour le module Yocto-MaxiBridge-C sont la fonction "multiCellWeighScale" combinant plusieurs cellules de poids pour une mesure unique, les quatre fonctions "weighScale1" à "weighScale4" indépendantes correspondant à chaque cellule de poids considérée séparément et la fonction "temperature" qui correspond à la mesure de température utilisée pour la compensation.

Pour accéder aux paramètres individuels de chaque cellule, pressez sur le bouton "+" à gauche du nom correspondant. Notez que certains paramètres comme le type d'excitation sont partagés entre toutes les entrées: si vous changez l'excitation sur une cellule, toutes seront affectées.

Configuration pour des cellules de poids

Un panneau de configuration spécifique facilite la configuration des paramètres d'une cellule de poids. Vous pouvez ainsi simplement renseigner la plage de mesure de la balance, effectuer une tare et une mesure avec un poids étalon pour effectuer une calibration fine. Vous pouvez aussi définir le paramètre de suivi automatique de tare (zero tracking) si vous désirez effectuer une compensation par ce biais.

Pour plus de détails sur la manière de concevoir une balance et de paramétrer le Yocto-MaxiBridge-C, référez-vous au chapitre précédent.

Configuration ratiométrique générique

Pour une utilisation autre que pour des cellules de poids, vous pouvez accéder à tous les paramètres individuels définissant la conversion entre tension ratiométrique et la valeur physique correspondante du capteur raccordé au le Yocto-MaxiBridge-C.

⁵ www.yoctopuce.com/FR/virtualhub.php

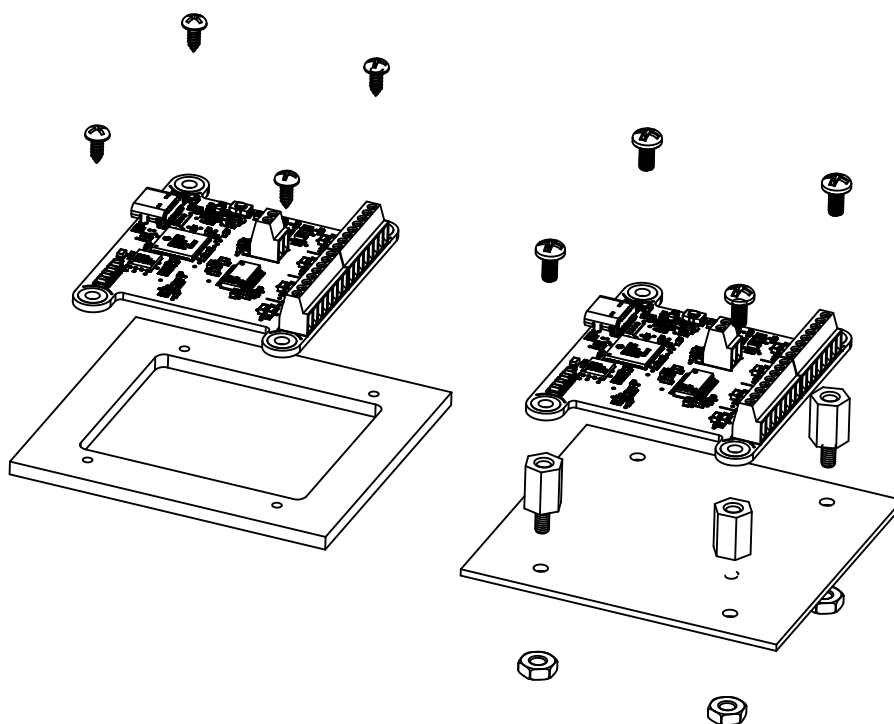
⁶ Consultez la documentation de VirtualHub pour plus de détails

5. Montage et connectique

Ce chapitre fournit des explications importantes pour utiliser votre module Yocto-MaxiBridge-C 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-MaxiBridge-C 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. 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 s'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 long¹. 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.

5.3. Compatibilité électromagnétique (EMI)

Les choix de connectique pour intégrer le Yocto-MaxiBridge-C 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,

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

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 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 tests d'immunité électromagnétique à effectuer pour respecter les normes.

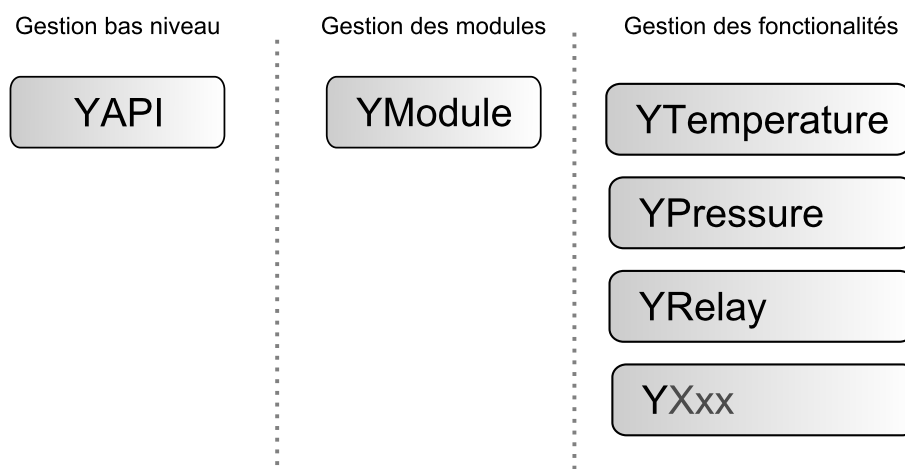
6. Programmation, concepts généraux

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-MaxiBridge-C 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-MaxiBridge-C 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-MaxiBridge-C

Le module Yocto-MaxiBridge-C offre quatre mesures de poids avec compensées, ainsi que quatre fonctions *genericSensor* décrivant directement la mesure de quatre ponts de Wheatstone.

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

multiCellWeighScale : MultiCellWeighScale

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
cellCount	Nombre entier	modifiable
externalSense	Booléen	modifiable

excitation	Type énuméré	modifiable
tempAvgAdaptRatio	Nombre (virgule fixe)	modifiable
tempChgAdaptRatio	Nombre (virgule fixe)	modifiable
compTempAvg	Nombre (virgule fixe)	lecture seule
compTempChg	Nombre (virgule fixe)	lecture seule
compensation	Nombre (virgule fixe)	lecture seule
zeroTracking	Nombre (virgule fixe)	modifiable
command	Texte	modifiable

weighScale1 : WeighScale**weighScale2 : WeighScale****weighScale3 : WeighScale****weighScale4 : WeighScale**

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
excitation	Type énuméré	modifiable
tempAvgAdaptRatio	Nombre (virgule fixe)	modifiable
tempChgAdaptRatio	Nombre (virgule fixe)	modifiable
compTempAvg	Nombre (virgule fixe)	lecture seule
compTempChg	Nombre (virgule fixe)	lecture seule
compensation	Nombre (virgule fixe)	lecture seule
zeroTracking	Nombre (virgule fixe)	modifiable
command	Texte	modifiable

genericSensor1 : GenericSensor**genericSensor2 : GenericSensor****genericSensor3 : GenericSensor****genericSensor4 : GenericSensor**

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
signalValue	Nombre (virgule fixe)	lecture seule
signalUnit	Texte	lecture seule
signalRange	Plage de valeurs	modifiable

valueRange	Plage de valeurs	modifiable
signalBias	Nombre (virgule fixe)	modifiable
signalSampling	Type énuméré	modifiable
enabled	Booléen	modifiable

temperature : Temperature

attribut	type	modifiable ?
logicalName	Texte	modifiable
advertisedValue	Texte	modifiable
unit	Texte	modifiable
currentValue	Nombre (virgule fixe)	lecture seule
lowestValue	Nombre (virgule fixe)	modifiable
highestValue	Nombre (virgule fixe)	modifiable
currentRawValue	Nombre (virgule fixe)	lecture seule
logFrequency	Fréquence	modifiable
reportFrequency	Fréquence	modifiable
advMode	Type énuméré	modifiable
calibrationParam	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

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
usage	0..100%	lecture seule
clearHistory	Booléen	modifiable

6.3. Module

Interface de contrôle des paramètres généraux des modules Yoctopuce

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

productName

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

serialNumber

Chaîne de caractères contenant le numéro de série, unique et préprogrammé en usine. Pour un module Yocto-MaxiBridge-C, ce numéro de série commence toujours par WMBRDG1C. Il peut servir comme point de départ pour accéder par programmation à un module particulier.

logicalName

Chaîne 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 194 en usine.

productRelease

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

firmwareRelease

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

persistentSettings

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

luminosity

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

beacon

Etat de la balise de localisation du module.

upTime

Temps écoulé depuis la dernière mise sous tension du module.

usbCurrent

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

rebootCountdown

Compte à rebours pour déclencher un redémarrage spontané du module.

userVar

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

6.4. MultiCellWeighScale

Interface pour interagir avec les capteurs de poids multi-cellules, disponibles par exemple dans le Yocto-MaxiBridge

La classe `YMultiCellWeighScale` permet d'obtenir une mesure de poids à partir d'un groupe de capteurs ratiométrique. Elle permet de contrôler globalement le mode d'excitation des cellules, pour éviter les dérives liées aux changements de température de l'électronique, et d'appliquer

automatiquement une correction supplémentaire en fonction de la température pour compenser la dérive des cellules de poids elles-mêmes.

logicalName

Chaîne de caractères contenant le nom logique du capteur de poids multi-cellules, 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 poids multi-cellules. Si deux capteurs de poids multi-cellules 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 poids multi-cellules, et qui sera publiée automatiquement jusqu'au hub parent. Pour un capteur de poids multi-cellules, la valeur publiée est la valeur courante de la mesure.

unit

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

currentValue

Valeur actuelle de la mesure, en l'unité spécifiée, sous forme de nombre à virgule.

lowestValue

Valeur minimale de la mesure, en l'unité spécifiée, sous forme de nombre à virgule.

highestValue

Valeur maximale de la mesure, en l'unité spécifiée, 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).

cellCount

Nombre de cellules de poids utilisées.

externalSense

Utilisation de l'entrée 4 comme référence externe pour les cellules à 6 fils.

excitation

Méthode d'excitation de la cellule de poids.

tempAvgAdaptRatio

Taux d'adaptation pour le calcul de la température moyenne, en pour mille.

tempChgAdaptRatio

Taux d'adaptation pour le calcul de la variation de température, en pour mille.

compTempAvg

Température moyennée utilisée pour la compensation thermique.

compTempChg

Variation actuelle de température pour la compensation thermique.

compensation

Valeur de la compensation thermique courante.

zeroTracking

Valeur seuil pour la tare continue à zéro.

command

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

6.5. WeighScale

Interface pour interagir avec les capteurs de poids, disponibles par exemple dans le Yocto-Bridge et le Yocto-MaxiBridge

La classe `YWeighScale` permet d'obtenir une mesure de poids à partir d'un capteur ratiométrique. Elle permet de contrôler le mode d'excitation de la cellule, pour éviter les dérives liées aux changements de température de l'électronique, et d'appliquer automatiquement une correction supplémentaire en fonction de la température pour compenser la dérive de la cellule de poids elle-même.

logicalName

Chaîne de caractères contenant le nom logique du capteur de poids, 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 au capteur de poids. Si deux capteurs de poids 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 poids, et qui sera publiée automatiquement jusqu'au hub parent. Pour un capteur de poids, la valeur publiée est la valeur courante de la mesure.

unit

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

currentValue

Valeur actuelle de la mesure, en l'unité spécifiée, sous forme de nombre à virgule.

lowestValue

Valeur minimale de la mesure, en l'unité spécifiée, sous forme de nombre à virgule.

highestValue

Valeur maximale de la mesure, en l'unité spécifiée, 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).

excitation

Méthode d'excitation de la cellule de poids.

tempAvgAdaptRatio

Taux d'adaptation pour le calcul de la température moyenne, en pour mille.

tempChgAdaptRatio

Taux d'adaptation pour le calcul de la variation de température, en pour mille.

compTempAvg

Température moyennée utilisée pour la compensation thermique.

compTempChg

Variation actuelle de température pour la compensation thermique.

compensation

Valeur de la compensation thermique courante.

zeroTracking

Valeur seuil pour la tare continue à zéro.

command

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

6.6. GenericSensor

Interface pour interagir avec les capteurs de type `GenericSensor`, disponibles par exemple dans le Yocto-0-10V-Rx, le Yocto-4-20mA-Rx, le Yocto-Bridge et le Yocto-milliVolt-Rx

La classe `YGenericSensor` permet de lire et de configurer les transducteurs de signaux Yoctopuce. Elle hérite de la classe `YSensor` toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, elle permet de configurer une conversion automatique entre le signal mesuré et la grandeur physique représentée.

logicalName

Chaîne de caractères contenant le nom logique du capteur générique, 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 générique. Si deux capteurs génériques 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 générique, et qui sera publiée automatiquement jusqu'au hub parent. Pour un capteur générique, la valeur publiée est la valeur courante de la mesure.

unit

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

currentValue

Valeur actuelle de la mesure, en l'unité spécifiée, sous forme de nombre à virgule.

lowestValue

Valeur minimale de la mesure, en l'unité spécifiée, sous forme de nombre à virgule.

highestValue

Valeur maximale de la mesure, en l'unité spécifiée, 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).

signalValue

Valeur actuelle du signal électrique mesuré par le capteur, sous forme de nombre à virgule.

signalUnit

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

signalRange

Plage de valeurs électriques utilisées par le capteur.

valueRange

Plage de valeurs physiques mesurées par le capteur, utilisée pour la conversion du signal.

signalBias

Biais du signal électrique pour la correction du point zéro.

signalSampling

Méthode d'échantillonnage du signal à utiliser.

enabled

Activation/désactivation de la mesure.

6.7. Temperature

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

La classe `YTemperature` permet de lire et de configurer les capteurs de température Yoctopuce. Elle hérite de la classe `YSensor` toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, elle permet de configurer les paramètres spécifiques de certains types de capteur (type de connexion, 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 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 capteur utilisé pour réaliser la mesure de température. Le capteur peut être digital, un type de thermocouple, un type de PT100, un thermistor ou encore un capteur infrarouge

signalValue

Valeur actuelle du signal électrique mesuré par le capteur (sauf pour les capteurs 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.

6.8. DataLogger

Interface de contrôle de l'enregistreur de données, présent sur la plupart des capteurs Yoctopuce.

La plupart des capteurs Yoctopuce sont équipés d'une mémoire non-volatile. Elle permet de mémoriser les données mesurées d'une manière autonome, sans nécessiter le suivi permanent d'un ordinateur. La classe `YDataLogger` contrôle les paramètres globaux de cet enregistreur de données. Le contrôle de l'enregistrement (start / stop) et la récupération des données se fait au niveau des objets qui gèrent les senseurs.

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 activé ou désactivé 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

Activation 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. Attention si le module n'a pas de source de temps à sa disposition, il va attendre environ 8 sec avant de démarrer automatiquement l'enregistrement

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.

usage

Pourcentage d'utilisation de la mémoire d'enregistrement.

clearHistory

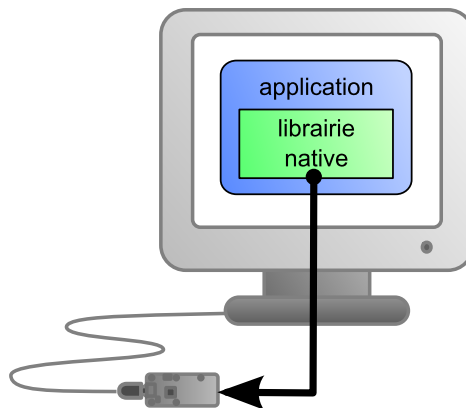
Attribut qui peut être mis à vrai pour effacer l'historique des mesures.

6.9. Quelle interface: Native, DLL ou Service?

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

Contrôle natif

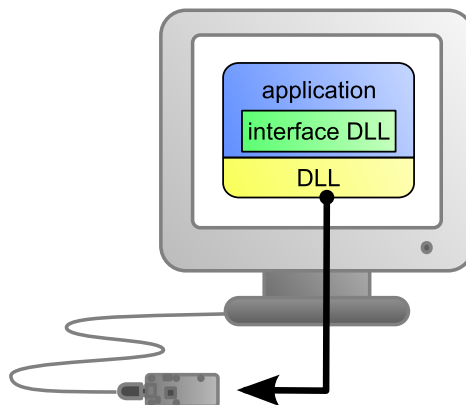
Dans ce cas de figure le programme pilotant votre projet est directement compilé avec une librairie qui offre le contrôle des modules. C'est objectivement la solution la plus simple et la plus élégante pour l'utilisateur final. Il lui suffira de brancher le câble USB et de lancer votre programme pour 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 différences: il faut simplement être sûr que la DLL sera installée sur son ordinateur en même temps que le programme principal.

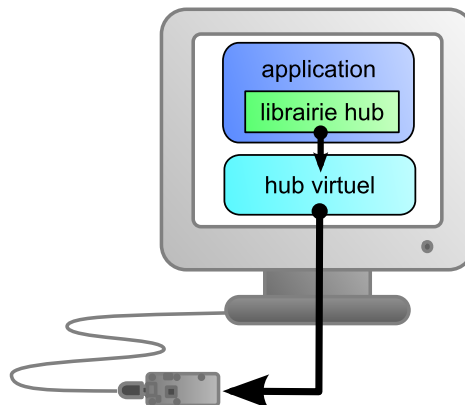


L'application utilise la DLL pour contrôler nativement le module connecté en local

Contrôle par un service

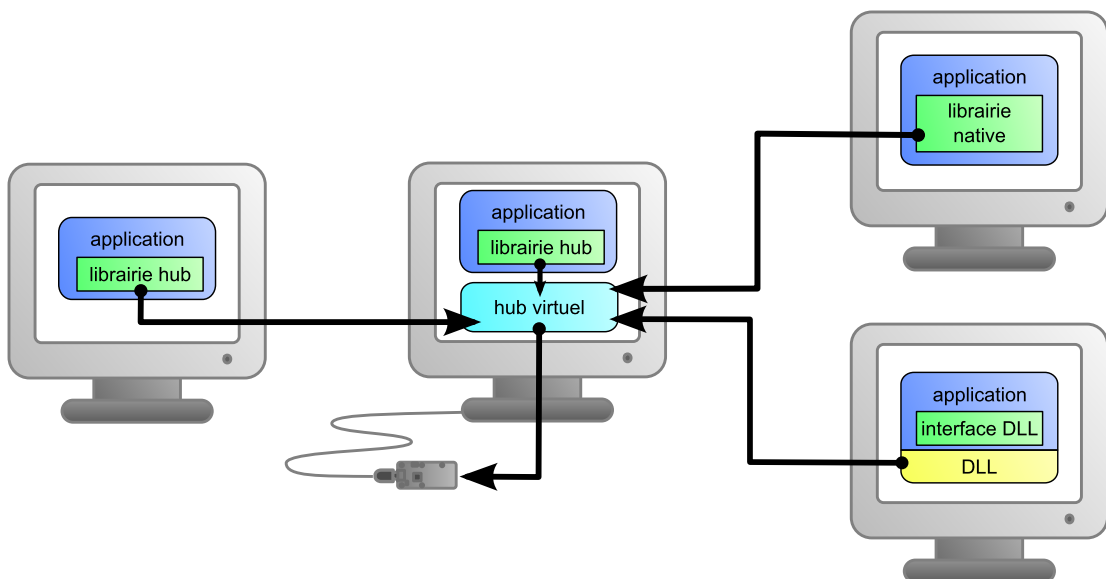
Certains langages ne permettent tout simplement pas d'accéder facilement au niveau matériel de la machine. C'est le cas de Javascript par exemple. Pour gérer ce cas, Yoctopuce offre la solution sous la forme d'un petit service, appelé VirtualHub qui, lui, est capable d'accéder aux modules, et votre application 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 VirtualHub avant de lancer le programme de contrôle du projet proprement dit, à moins qu'il ne décide d'installer VirtualHub sous la forme d'un service/démon, auquel cas 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 obligée de tourner sur la machine où se trouvent les modules: elle peut parfaitement se trouver sur une autre machine qui se connectera au service pour piloter les modules. De plus, les librairies natives et DLL évoquées plus haut sont aussi capables de se connecter à distance à une ou plusieurs instances de VirtualHub.



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

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

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

Méthode de support pour les différents langages.

6.10. Accéder aux modules à travers un hub

VirtualHub pour contourner la limitation d'accès à USB

Une seule application à la fois peut avoir accès nativement aux modules Yoctopuce. Cette limitation est liée au fait que deux processus différents ne peuvent pas parler en même temps à un périphérique USB. En général, ce type de problème est réglé par un driver qui se charge de faire la police pour éviter que plusieurs processus ne se battent pour le même périphérique. Mais les produits Yoctopuce n'utilisent pas de drivers. Par conséquent, le premier processus qui arrive à accéder au mode natif le garde pour lui jusqu'à ce que `UnregisterHub` ou `FreeApi` soit appelé.

Si votre application essaie de communiquer en mode natif avec les modules Yoctopuce, mais qu'une autre application vous empêche d'y accéder, vous recevrez le message d'erreur suivant:

```
Another process is already using yAPI
```

La solution est d'utiliser VirtualHub localement sur votre machine et de vous en servir comme passerelle pour vos applications. Ainsi, si toutes vos application utilisent VirtualHub, vous n'aurez plus de conflit et vous pourrez accéder en tout temps à tous vos modules.

Pour passer du mode natif au mode réseau sur votre machine locale, il vous suffit de changer le paramètre de l'appel à `YAPI.RegisterHub` et d'indiquer `127.0.0.1` à la place de `usb`.

Avec un YoctoHub

Un YoctoHub se comporte exactement comme un ordinateur faisant tourner VirtualHub. La seule différence entre un programme utilisant l'API Yoctopuce utilisant des modules en USB natif et ce même programme utilisant des modules Yoctopuce connectés à un YoctoHub se situe au niveau de l'appel à `RegisterHub`. Pour utiliser des modules USB connectés en natif, le paramètre de `RegisterHub` est `usb`, pour utiliser des modules connectés à un YoctoHub, il suffit de remplacer ce paramètre par l'adresse IP du YoctoHub.

Il y a donc trois cas de figure: le mode natif, le mode réseau à travers VirtualHub sur votre machine locale, ou à travers un YoctoHub. Pour passer de l'un à l'autre, il vous suffit de changer le paramètre de l'appel à `YAPI.RegisterHub` comme dans les exemples ci-dessous:

```
YAPI.RegisterHub("usb",errmsg); // utilisation en mode natif USB

YAPI.RegisterHub("127.0.0.1",errmsg); // utilisation en mode réseau local avec VirtualHub

YAPI.RegisterHub("192.168.0.10",errmsg); // utilisation avec YoctoHub dont l'adresse IP est 192.168.0.10
```

6.11. Programmation, par où commencer?

Arrivé à ce point du manuel, vous devriez connaître l'essentiel de la théorie à propos de votre Yocto-MaxiBridge-C. 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.

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-MaxiBridge-C.

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

7. Utilisation du Yocto-MaxiBridge-C en ligne de commande

Lorsque vous désirez effectuer une opération ponctuelle sur votre Yocto-MaxiBridge-C, comme la lecture d'une valeur, le changement d'un nom logique, etc.. vous pouvez bien sûr utiliser VirtualHub, 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-MaxiBridge-C, ouvrir un shell et commencer à travailler en tapant par exemple:

```
C:\>YMultiCellWeighScale any set_excitationVoltage EXCITATION_AC
C:\>YMultiCellWeighScale any tare
C:\>YMultiCellWeighScale 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 MultiCellWeighScale

Pour contrôler la fonction MultiCellWeighScale de votre Yocto-MaxiBridge-C, vous avez besoin de l'exécutable YMultiCellWeighScale.

Vous pouvez par exemple lancer:

```
C:\>YMultiCellWeighScale any set_excitationVoltage EXCITATION_AC
C:\>YMultiCellWeighScale any tare
C:\>YMultiCellWeighScale any get_currentValue
```

Cet exemple utilise la cible "any" pour signifier que l'on désire travailler sur la première fonction MultiCellWeighScale 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-MaxiBridge-C avec le numéros de série WMBRDG1C-123456 que vous auriez appelé "MonModule" et dont vous auriez nommé la fonction multiCellWeighScale "MaFonction", les cinq appels suivants seront strictement équivalents (pour autant que MaFonction ne soit définie qu'une fois, pour éviter toute ambiguïté).

```
C:\>YMultiCellWeighScale WMBRDG1C-123456.multiCellWeighScale describe
C:\>YMultiCellWeighScale WMBRDG1C-123456.MaFonction describe
C:\>YMultiCellWeighScale MonModule.multiCellWeighScale describe
C:\>YMultiCellWeighScale MonModule.MaFonction describe
C:\>YMultiCellWeighScale MaFonction describe
```

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

```
C:\>YMultiCellWeighScale all describe
```

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

```
C:\>YMultiCellWeighScale /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 WMBRDG1C-12346 set_logicalName MonPremierModule
C:\>YModule WMBRDG1C-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 WMBRDG1C-12346 set_logicalName MonPremierModule
C:\>YModule WMBRDG1C-12346 saveToFlash
```

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

```
C:\>YModule -s WMBRDG1C-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 VirtualHub: il suffit de faire tourner 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 VirtualHub 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 VirtualHub. Notez que VirtualHub compte comme une application native.

8. Utilisation du Yocto-MaxiBridge-C 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 2.7 et 3.x jusqu'aux toutes dernières versions officielles. Elle fonctionne sous Windows, macOS et Linux tant Intel qu'ARM. Les interpréteurs Python sont disponibles sur le site de Python¹.

8.1. 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.

8.2. Librairie dynamique

Une partie de la librairie de bas-niveau est écrite en C, mais vous n'aurez a priori pas besoin d'interagir directement avec elle: cette partie est fournie sous forme de DLL sous Windows, de fichier *.so* sous Unix et de fichier *.dylib* sous macOS. 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.

8.3. Contrôle de la fonction MultiCellWeighScale

¹ <http://www.python.org/download/>

² www.yoctopuce.com/FR/libraries.php

Il suffit de quelques lignes de code pour piloter un Yocto-MaxiBridge-C. Voici le squelette d'un fragment de code Python qui utilise la fonction `MultiCellWeighScale`.

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

# On récupère l'objet permettant d'interagir avec le module
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.multiCellWeighScale")

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

YMultiCellWeighScale.FindMultiCellWeighScale

La fonction `YMultiCellWeighScale.FindMultiCellWeighScale` permet de retrouver un capteur de poids multi-cellules 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-MaxiBridge-C avec le numéros de série *WMBRDG1C-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *multiCellWeighScale* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.multiCellWeighScale")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.MaFonction")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("MonModule.multiCellWeighScale")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MonModule.MaFonction")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MaFonction")
```

`YMultiCellWeighScale.FindMultiCellWeighScale` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur de poids multi-cellules.

isOnline

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

A propos des imports Python

Cette documentation suppose que vous utilisez la librairie Python téléchargée directement depuis le site web de Yoctopuce, mais si vous avez installé la librairie Yoctopuce avec PIP, alors vous devrez préfixer tous les imports avec *yoctopuce*.. Ainsi tous les exemples donnés dans la documentation, tels que:

```
from yocto_api import *
```

doivent être convertis, lorsque que la librairie Yoctopuce a été installée par PIP, en:

```
from yoctopuce.yocto_api import *
```

set_excitation

La méthode `set_excitation()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'activer sur le Yocto-MaxiBridge-C la source de tension permettant d'effectuer la mesure ratiométrique (mise sous tension de la cellule de charge et de force). L'utilisation d'une excitation alternative (AC) donne les meilleurs résultats pour une cellule de poids standard.

tare

La méthode `tare()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet de calibrer la position au repos de la balance. Elle doit être appelée lorsque la balance n'est pas en charge.

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'obtenir la valeur actuelle mesurée par le Yocto-MaxiBridge-C, calculée comme la somme de toutes les entrées actives du module. La valeur de poids dépend bien entendu de la configuration des paramètres des cellules de poids, qui devrait être faite précédemment pour obtenir une mesure correcte.

Un exemple réel

Lancez votre interpréteur Python et ouvrez le script correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-MaxiBridge-C** 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_multicellweighscale 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 genericSensor sensor
    sensor = YMultiCellWeighScale.FirstMultiCellWeighScale()
    if sensor is None:
        die('No module connected')
else:
    sensor = YMultiCellWeighScale.FindMultiCellWeighScale(target + '.multiCellWeighScale')
```

```

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

# On startup, enable excitation and tare weigh scale
print("Resetting tare weight...");
sensor.set_excitation(YMultiCellWeighScale.EXCITATION_AC);
YAPI.Sleep(3000);
sensor.tare();

while sensor.isOnline():
    print("Weight:  %f %s" % (sensor.get_currentValue(), sensor.get_unit()))
    print("  (Ctrl-C to stop)")
    YAPI.Sleep(1000)
YAPI.FreeAPI()

```

8.4. Contrôle de la partie module

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

```

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

from yocto_api import *

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

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

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

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

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

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

```

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

Modifications des réglages du module

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

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

from yocto_api import *

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

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

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

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

Attention, le nombre de cycles d'écriture de la mémoire non volatile du module est limité. Passé cette limite plus rien ne garantit que la sauvegarde des réglages se passera correctement. Cette limite, liée à la technologie employée par le micro-processeur du module se situe aux 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

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

from yocto_api import *

errmsg = YRefParam()

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

print('Device list')

module = YModule.FirstModule()
while module is not None:
```

```
print(module.get_serialNumber() + ' (' + module.get_productName() + ')')
module = module.nextModule()
YAPI.FreeAPI()
```

8.5. Gestion des erreurs

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

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

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

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

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

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

9. Utilisation du Yocto-MaxiBridge-C 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, C++ est supporté avec Microsoft Visual Studio 2017 et les versions plus récentes. Sous macOS, nous supportons les versions de XCode supportées par Apple. Sous Linux, nous supportons toutes les versions de gcc publiées depuis 2008. Par ailleurs, aussi bien sous macOS 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` vous permet de compiler les exemples en ligne de commande, en pleine connaissance des arguments de compilation et link.

Les bibliothèques Yoctopuce¹ 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 les 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 linké avec les bibliothèques Yoctopuce.

9.1. Contrôle de la fonction MultiCellWeighScale

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

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

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

¹ www.yoctopuce.com/FR/libraries.php

```
[...]

// On récupère l'objet permettant d'interagir avec le module
YMultiCellWeighScale *multicellweighscale;
multicellweighscale = YMultiCellWeighScale::FindMultiCellWeighScale
("WMBRDG1C-123456.multiCellWeighScale");

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.h et yocto_multicellweighscale.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_multicellweighscale.h` est nécessaire pour gérer les modules contenant un capteur de poids multi-cellules, comme le Yocto-MaxiBridge-C.

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.

YMultiCellWeighScale::FindMultiCellWeighScale

La fonction `YMultiCellWeighScale::FindMultiCellWeighScale` permet de retrouver un capteur de poids multi-cellules 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-MaxiBridge-C avec le numéros de série *WMBRDG1C-123456* que vous auriez appelé *"MonModule"* et dont vous auriez nommé la fonction *multiCellWeighScale* *"MaFonction"*, les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YMultiCellWeighScale *multicellweighscale = YMultiCellWeighScale::FindMultiCellWeighScale
("WMBRDG1C-123456.multiCellWeighScale");
YMultiCellWeighScale *multicellweighscale = YMultiCellWeighScale::FindMultiCellWeighScale
("WMBRDG1C-123456.MaFonction");
YMultiCellWeighScale *multicellweighscale = YMultiCellWeighScale::FindMultiCellWeighScale
("MonModule.multiCellWeighScale");
YMultiCellWeighScale *multicellweighscale = YMultiCellWeighScale::FindMultiCellWeighScale
("MonModule.MaFonction");
YMultiCellWeighScale *multicellweighscale = YMultiCellWeighScale::FindMultiCellWeighScale
("MaFonction");
```

`YMultiCellWeighScale::FindMultiCellWeighScale` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur de poids multi-cellules.

isOnline

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

set_excitation

La méthode `set_excitation()` de l'objet renvoyée par `yFindMultiCellWeighScale` permet d'activer sur le Yocto-MaxiBridge-C la source de tension permettant d'effectuer la mesure ratiométrique (mise sous tension de la cellule de charge et de force). L'utilisation d'une excitation alternative (AC) donne les meilleurs résultats pour une cellule de poids standard.

tare

La méthode `tare()` de l'objet renvoyée par `yFindMultiCellWeighScale` permet de calibrer la position au repos de la balance. Elle doit être appelée lorsque la balance n'est pas en charge.

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyée par `yFindMultiCellWeighScale` permet d'obtenir la valeur actuelle mesurée par le Yocto-MaxiBridge-C, calculée comme la somme de toutes les entrées actives du module. La valeur de poids dépend bien entendu de la configuration des paramètres des cellules de poids, qui devrait être faite précédemment pour obtenir une mesure correcte.

Un exemple réel

Lancez votre environnement C++ et ouvrez le projet exemple correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-MaxiBridge-C** 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_multicellweighscale.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 = YAPI::GetTickCount();
    while (YAPI::GetTickCount() - now < 3000) {
        // wait 3 sec to show the message
    }
    exit(1);
}

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

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

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

    if (target == "any") {
        sensor = YMultiCellWeighScale::FirstMultiCellWeighScale();
        if (sensor == NULL) {
            cout << "No module connected (check USB cable)" << endl;
            return 1;
        }
    } else {
        sensor = YMultiCellWeighScale::FindMultiCellWeighScale(
            target + ".multiCellWeighScale");
    }

    string unit;

    if (sensor->isOnline()) {
        // On startup, enable excitation and tare weigh scale
    }
}
```

```

    cout << "Resetting tare weight..." << endl;
    sensor->set_excitation(YMultiCellWeighScale::EXCITATION_AC);
    YAPI::Sleep(3000, errmsg);
    sensor->tare();
    unit = sensor->get_unit();
}

// Show measured weight continuously
while (sensor->isOnline()) {
    cout << "Weight : " << sensor->get_currentValue() << unit;
    cout << " (press Ctrl-C to exit)" << endl;
    YAPI::Sleep(1000, errmsg);
};
YAPI::FreeAPI();
cout << "Module not connected (check identification and USB cable)";
return 0;
}

```

9.2. Contrôle de la partie module

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

```

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

#include "yocto_api.h"

using namespace std;

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

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

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

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

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

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

```

```

        << endl;
    }
    YAPI::FreeAPI();
    return 0;
}

```

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

```

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

#include "yocto_api.h"

using namespace std;

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

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

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

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

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

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

```

Attention, le nombre de cycles d'écriture de la mémoire non volatile du module est limité. Passé cette limite plus rien ne garantit que la sauvegarde des réglages se passera correctement. Cette limite, liée à la technologie employée par le micro-processeur du module se situe aux 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();
    }
    YAPI::FreeAPI();
    return 0;
}
```

9.3. Gestion des erreurs

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

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

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

- Si votre code attrape l'exception au vol et la gère, et tout se passe bien.

- Si votre programme tourne dans le debugger, vous pourrez relativement facilement déterminer où le problème s'est produit, et voir le message explicatif lié à l'exception.
- Sinon... l'exception va crasher votre programme, boum!

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

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

9.4. Intégration de la librairie Yoctopuce en C++

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

Intégration au format source (recommandé)

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

- Elle garantit 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 macOS: **IOKit.framework** et **CoreFoundation.framework**
- Pour Linux: **libm**, **libpthread**, **libusb1.0** et **libstdc++**

Intégration en librairie statique

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

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

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

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

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

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

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

Intégration en librairie dynamique

L'intégration de la librairie Yoctopuce sous forme de librairie dynamique permet de produire un exécutable plus petit que les deux méthodes précédentes, et de mettre éventuellement à jour cette librairie si un correctif s'avérait nécessaire sans devoir recompilier 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 macOS et Linux).

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

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

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

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

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

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

10. Utilisation du Yocto-MaxiBridge-C 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. C# est supporté sous Windows Visual Studio 2017 et ses versions plus récentes.

Notre librairie est aussi compatible avec *Mono*, la version open source de C# qui fonctionne sous Linux et macOS. Sous Linux, utilisez la version 5.20 ou plus récente. Le support de Mono sous macOS est limité aux systèmes 32bits, ce qui le rend de nos jours à peu près inutile. Vous trouverez sur notre site web différents articles qui décrivent comment indiquer à Mono comment accéder à notre librairie.

10.1. Installation

Téléchargez la librairie Yoctopuce pour Visual C# depuis le site web de Yoctopuce¹. 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.

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

La librairie Yoctopuce pour Visual C# .NET se présente sous la forme d'une DLL et de fichiers sources en Visual C#. La DLL n'est pas une DLL .NET mais une DLL classique, écrite en C, qui gère les communications à bas niveau avec les modules². 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 www.yoctopuce.com/FR/libraries.php

² 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`³. 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.

10.3. Contrôle de la fonction MultiCellWeighScale

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

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

// On récupère l'objet permettant d'interagir avec le module
YMultiCellWeighScale multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale(
    "WMBRDG1C-123456.multiCellWeighScale");

// Pour gérer le hot-plug, on vérifie que le module est là
if (multicellweighscale.IsOnline())
{
    // Utiliser multicellweighscale.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 `errmsg` une explication du problème.

YMultiCellWeighScale.FindMultiCellWeighScale

La fonction `YMultiCellWeighScale.FindMultiCellWeighScale` permet de retrouver un capteur de poids multi-cellules 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-MaxiBridge-C avec le numéros de série `WMBRDG1C-123456` que vous auriez appelé `"MonModule"` et dont vous auriez nommé la fonction `multiCellWeighScale` `"MaFonction"`, les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

³ Pensez à changer le filtre de la fenêtre de sélection de fichiers, sinon la DLL n'apparaîtra pas

```

multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.multiCellWeighScale");
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.MaFonction");
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("MonModule.multiCellWeighScale");
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MonModule.MaFonction");
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MaFonction");

```

`YMultiCellWeighScale.FindMultiCellWeighScale` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur de poids multi-cellules.

isOnline

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

set_excitation

La méthode `set_excitation()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'activer sur le Yocto-MaxiBridge-C la source de tension permettant d'effectuer la mesure ratiométrique (mise sous tension de la cellule de charge et de force). L'utilisation d'une excitation alternative (AC) donne les meilleurs résultats pour une cellule de poids standard.

tare

La méthode `tare()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet de calibrer la position au repos de la balance. Elle doit être appelée lorsque la balance n'est pas en charge.

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'obtenir la valeur actuelle mesurée par le Yocto-MaxiBridge-C, calculée comme la somme de toutes les entrées actives du module. La valeur de poids dépend bien entendu de la configuration des paramètres des cellules de poids, qui devrait être faite précédemment pour obtenir une mesure correcte.

Un exemple réel

Lancez Visual C# et ouvrez le projet exemple correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-MaxiBridge-C** 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 errmsg = "";
string target;

YMultiCellWeighScale sensor;

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

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

if (target == "ANY") {
    sensor = YMultiCellWeighScale.FirstMultiCellWeighScale();
    if (sensor == null) {
        Console.WriteLine("No module connected (check USB cable) ");
        Environment.Exit(0);
    }

    Console.WriteLine("Using: " + sensor.get_module().get_serialNumber());
} else {
    sensor = YMultiCellWeighScale.FindMultiCellWeighScale(target +
".multiCellWeighScale");
}

if (sensor.isOnline()) {
    string unit = "";
    // On startup, enable excitation and tare weigh scale
    Console.WriteLine("Resetting tare weight...");
    sensor.set_excitation(YMultiCellWeighScale.EXCITATION_AC);
    YAPI.Sleep(3000, ref errmsg);
    sensor.tare();
    unit = sensor.get_unit();

    // Show measured weight continuously
    while (sensor.isOnline()) {
        Console.Write("Weight : " + sensor.get_currentValue().ToString() + unit);
        Console.WriteLine(" (press Ctrl-C to exit)");
        YAPI.Sleep(1000, ref errmsg);
    }
}

YAPI.FreeAPI();
Console.WriteLine("Module not connected");
Console.WriteLine("check identification and USB cable");
}
}
}

```

10.4. Contrôle de la partie module

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

```

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

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

```

```

    }

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

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

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

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

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

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

Modifications des réglages du module

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

```

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.WriteLine("Module: serial= " + m.get_serialNumber());
        Console.WriteLine(" / name= " + m.get_logicalName());
    } else {
        Console.WriteLine("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 errmsg = "";

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

```



```

    }

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

```

10.5. Gestion des erreurs

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

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

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

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

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

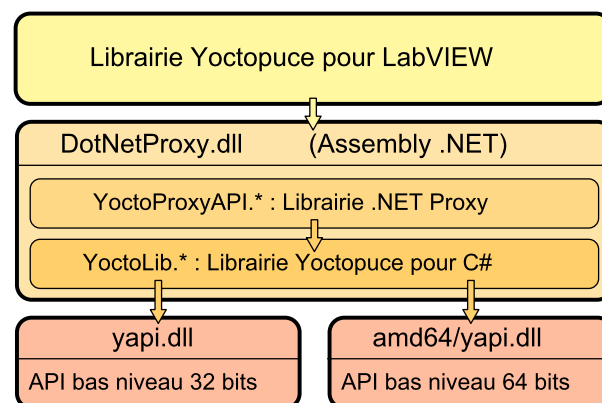
Quand vous travaillez sans les exceptions, il est possible d'obtenir un code d'erreur et un message expliquant l'origine de l'erreur en le demandant à l'objet qui a retourné une erreur à l'aide des méthodes `errType()` et `errMessage()`. Ce sont les mêmes informations qui auraient été associées à l'exception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

11. Utilisation du Yocto-MaxiBridge-C avec LabVIEW

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

11.1. Architecture

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



Architecture de l'API Yoctopuce pour LabVIEW

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

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

11.2. Compatibilité

Firmwares

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

LabVIEW pour Linux et MacOS

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

LabVIEW NXG

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

A propos de DotNetProxyLibrary.dll

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

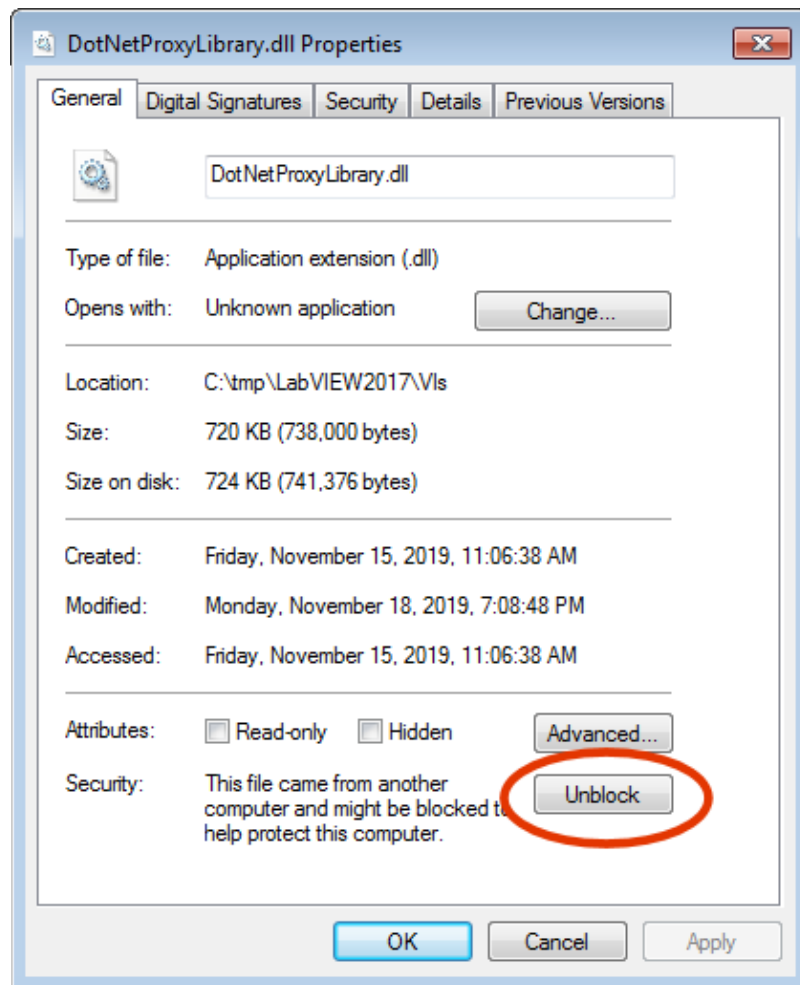
11.3. Installation

Téléchargez la librairie pour LabVIEW depuis le site web de Yoctopuce¹. Il s'agit d'un fichier ZIP dans lequel vous trouverez un répertoire par version de LabVIEW. Chacun de ses répertoires contient deux sous-répertoires. Le premier contient des exemples de programmation pour chaque produit Yoctopuce; le second, nommé *VI*s, contient tous les VI de l'API et les DLL nécessaires.

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

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

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



Débloquer la DLL DotNetProxyLibrary.dll.

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

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

Veillez à choisir le bon répertoire en fonction de la version de LabVIEW que vous utilisez (32 bits vs 64 bits). Vous trouverez plus d'information à propos de ce fichier sur le site web de National Instrument².

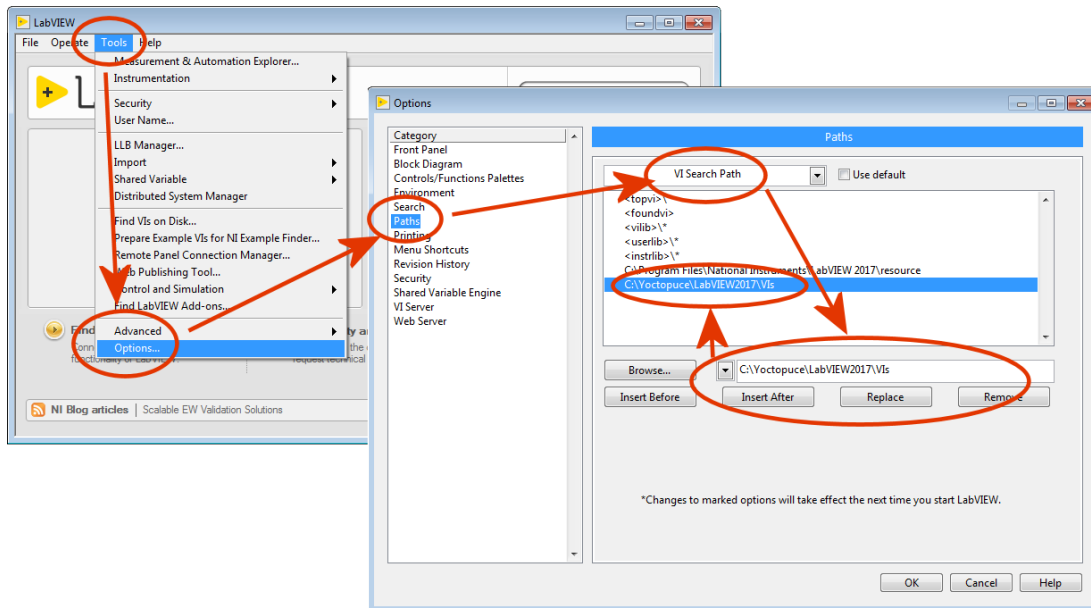
Pour installer l'API Yoctopuce pour LabVIEW vous avez plusieurs méthodes à votre disposition.

Méthode 1 : Installation "à l'emporter"

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

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

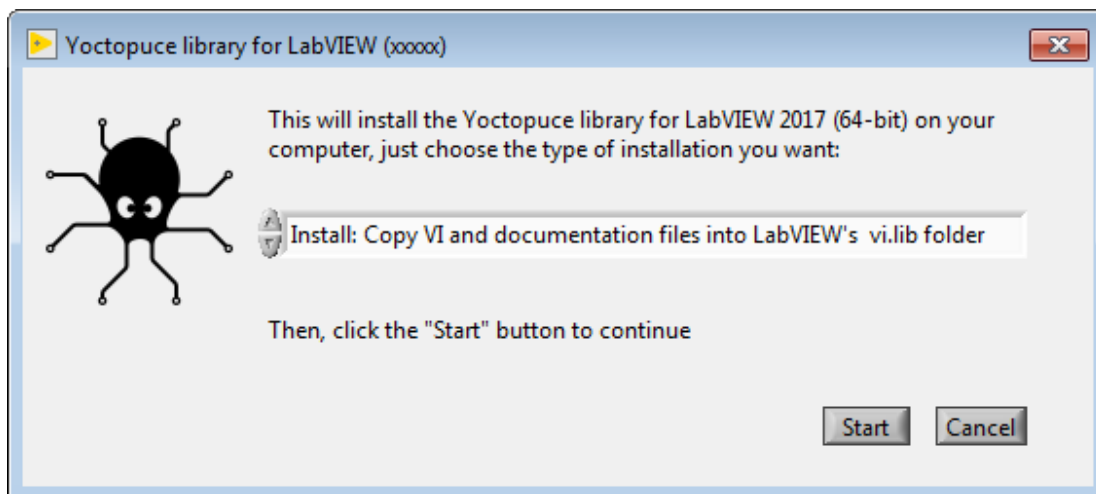
² <https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P8XnSAK>



Configuration du "VI Search Path"

Méthode 2 : Installateur fourni avec la librairie

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



L'installateur fourni avec la librairie

Cet installateur offre trois options d'installation:

Install: Keep VI and documentation files where they are.

Avec cette option, les VI sont conservés à l'endroit où la librairie a été décompressée. Vous aurez donc à faire en sorte qu'ils ne soit pas effacés tant que vous en aurez besoin. Voici ce que fait exactement l'installateur quand cette option est choisie:

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

Install: Copy VI and documentation files into LabVIEW's vi.lib folder

Dans ce cas, tous les fichiers nécessaires au bon fonctionnement de la librairie sont copiés dans le répertoire d'installation de LabVIEW. Vous pourrez donc effacer les fichiers originaux une fois

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

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

Uninstall Yoctopuce Library

Cette option supprime la Librairie Yoctopuce de votre installation LabVIEW:

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

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

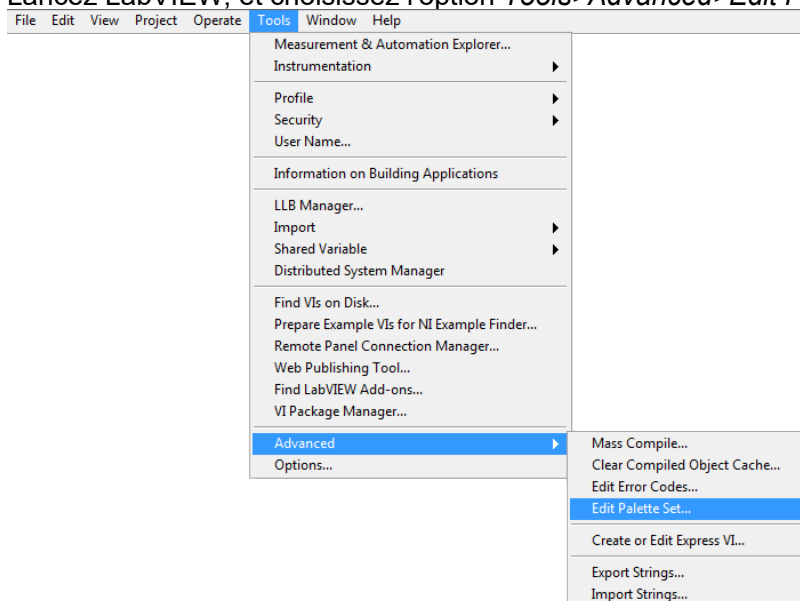
L'installateur reconnaît les répertoires contenant la librairie Yoctopuce en testant l'existence du fichier *YRegisterHub.vi*.

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

Méthode 3 Installation manuelle dans la palette LabVIEW

Les étapes pour installer manuellement les VIs directement dans la Palette LabView sont un peu plus complexes, vous trouverez la procédure complète sur le site de National Instruments³, mais voici un résumé:

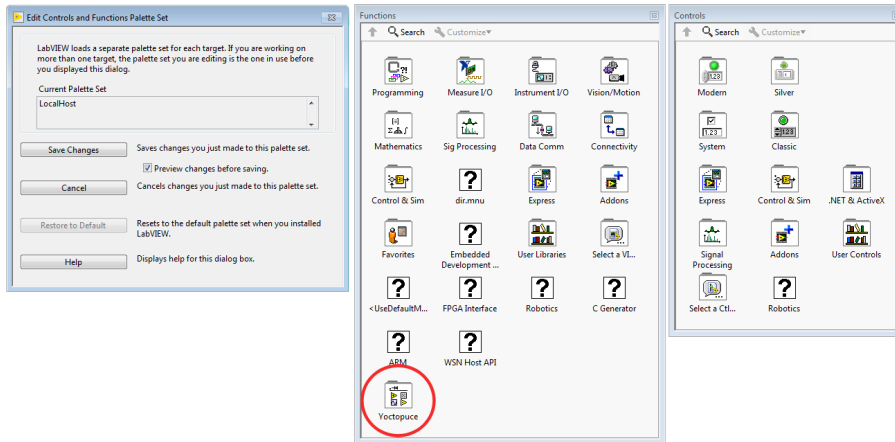
1. Créez un répertoire *Yoctopuce\API* dans le répertoire *C:\Program Files\National Instruments\LabVIEW xxxx\vi.lib*, et copiez tous les VIs et les DLL du répertoire *VIs* dedans.
2. Créez un répertoire *Yoctopuce* dans le répertoire *C:\Program Files\National Instruments\LabVIEW xxxx\menus\Categories*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



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

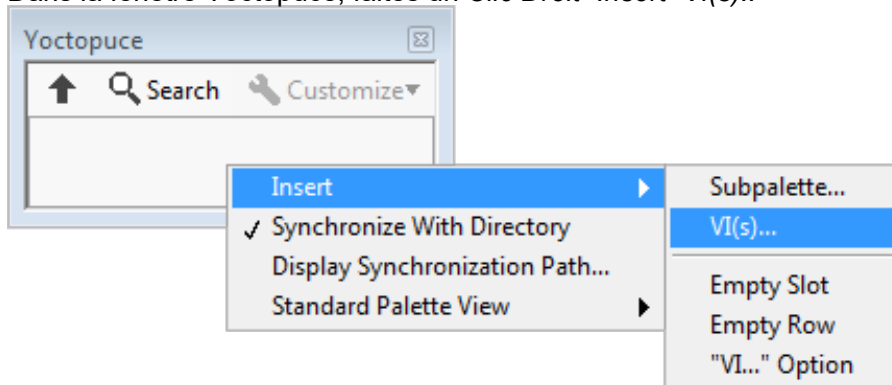
Trois fenêtres vont apparaître:

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

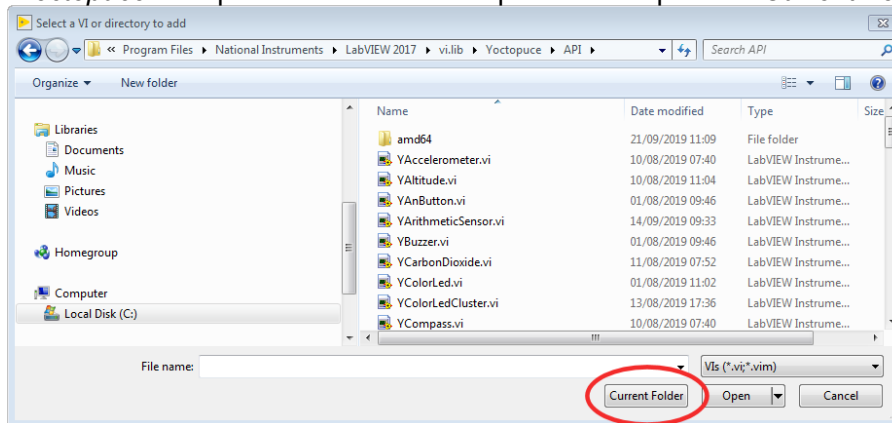


Dans la fenêtre *Function*, vous trouverez une icône *Yoctopuce*. Double-cliquez dessus, ce qui fera apparaître une fenêtre "Yoctopuce" vide.

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

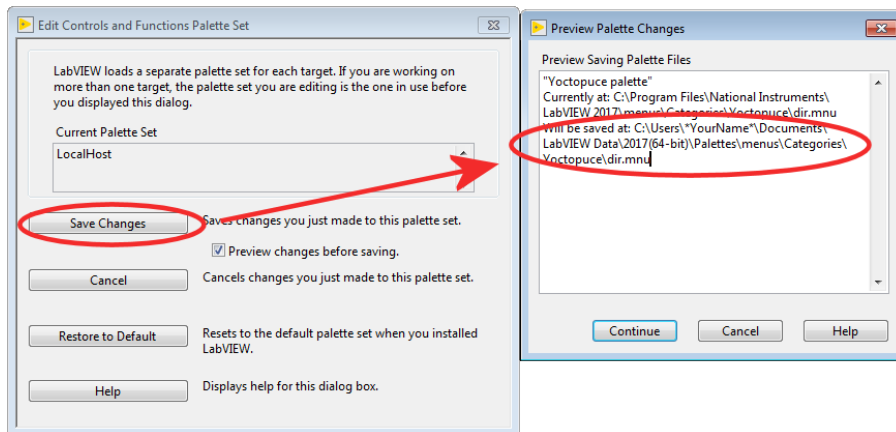


ce qui fera apparaître un sélecteur de fichier. Placer le sélecteur dans le répertoire *vi.lib* \Yoctopuce\API que vous avez créé au point 1 et cliquez sur *Current Folder*



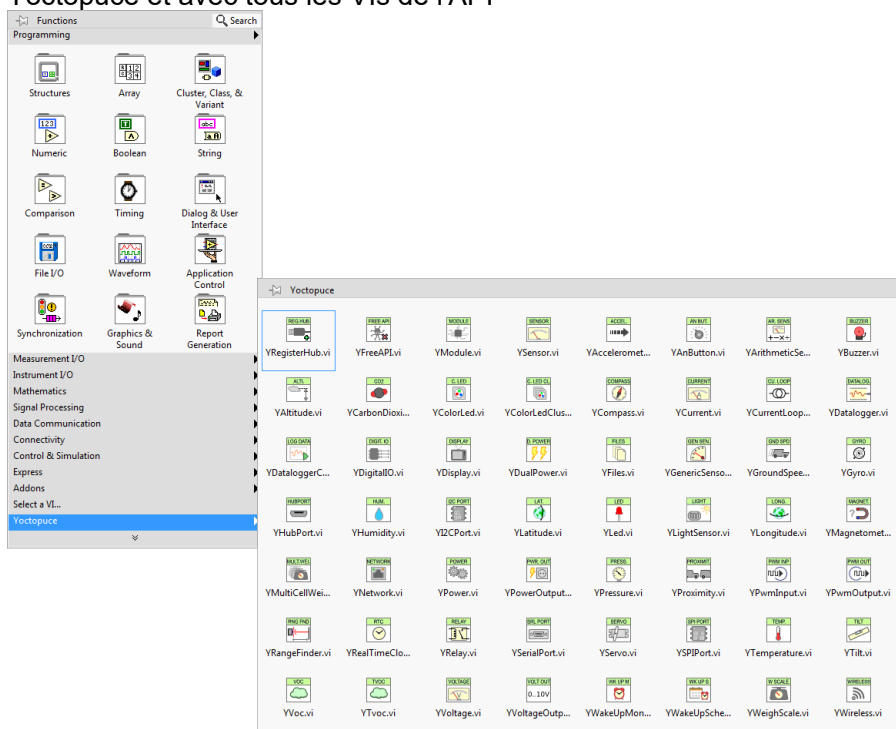
Tous les VIs Yoctopuce vont apparaître dans la fenêtre Yoctopuce. Par défaut, ils sont triés dans l'ordre alphabétique, mais vous pouvez les arranger comme bon vous semble en les glissant avec la souris. Pour que la palette soit bien utilisable, nous vous suggérons de réorganiser les icônes sur 8 colonnes.

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



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

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

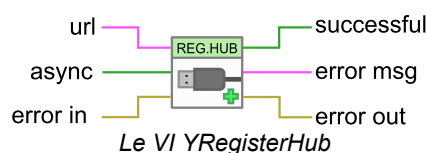


11.4. Présentation des VIs Yoctopuce

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

YRegisterHub

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



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

- La chaîne de caractères "usb" pour indiquer que l'on souhaite travailler avec des modules locaux directement par USB
- Une adresse IP pour indiquer que l'on souhaite travailler avec des modules accessibles via une connexion réseau. Cette adresse IP peut être celle d'un YoctoHub⁴ ou encore celle d'une machine sur laquelle tourne l'application VirtualHub⁵.

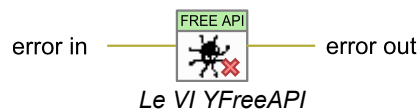
Dans le cas d'une adresse IP, le VI `YRegisterHub` va essayer de contacter cette adresse et générera une erreur s'il n'y arrive pas, à moins que le paramètre *async* ne soit mis à TRUE. Si *async* est mis à TRUE, aucune erreur ne sera générée, et les modules Yoctopuce correspondant à cette adresse IP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

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

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

YFreeAPI

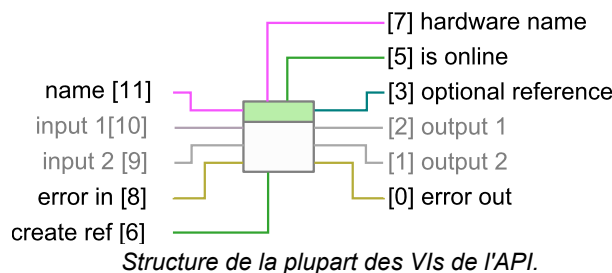
Le VI `YFreeAPI` permet de libérer les ressources allouées par l'API Yoctopuce.



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

Structure des VI correspondant à une classe

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



- Connecteur [11]: *name* doit contenir le nom hardware ou le nom logique de la fonction visée.
- Connecteur [10] et [9]: paramètres d'entrée qui dépendent de la nature du VI
- Connecteur [8] et [0]: *error in* et *error out*.
- Connecteur [7]: Nom hardware unique de la fonction trouvée.
- Connecteur [5]: *is online* contient TRUE si la fonction est accessible, FALSE sinon.
- Connecteur [2] et [1]: valeurs de sortie qui dépendent de la nature du VI.
- Connecteur [6]: Si cette entrée est mise à TRUE, le connecteur [3] contiendra une référence à l'objet *Proxy* implémenté par le VI⁶. Cette entrée est initialisée à FALSE par défaut.

⁴ www.yoctopuce.com/FR/products/category/extensions-and-networking

⁵ <http://www.yoctopuce.com/EN/virtualhub.php>

⁶ voir section *Utilisation objets Proxy*

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

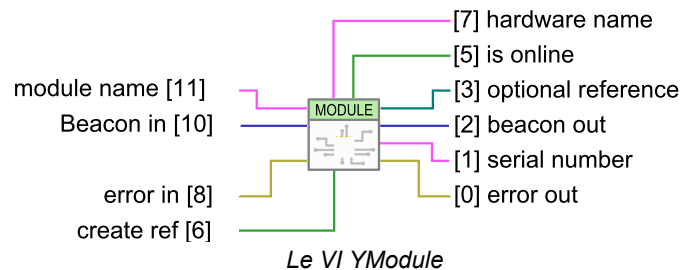
Vous trouverez la liste des fonctions disponibles sur votre Yocto-MaxiBridge-C au chapitre *Programmation, concepts généraux*.

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

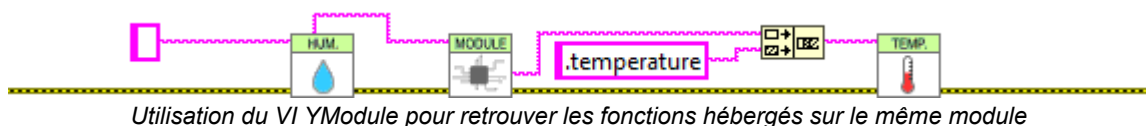
Si le paramètre *name* contient une chaîne vide, le VI ciblera la première fonction disponible du même type qu'il trouvera. Si aucune fonction n'est disponible, *is online* contiendra FALSE.

Le VI YModule

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

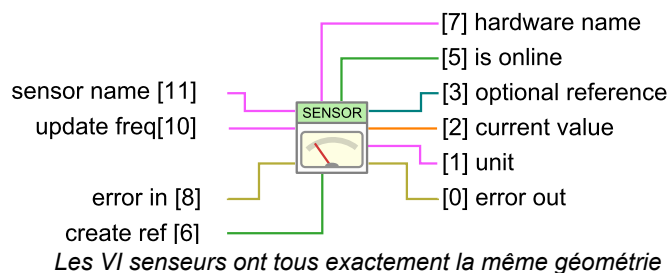


L'entrée *name* fonctionne de manière légèrement différente des autres VIs. S'il est appelé avec le paramètre *name* correspondant à un nom de fonction, le VI `YModule` trouvera la fonction *Module* du module hébergeant la fonction. Il est donc possible de trouver facilement le numéro de série du module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



Les VI senseurs

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

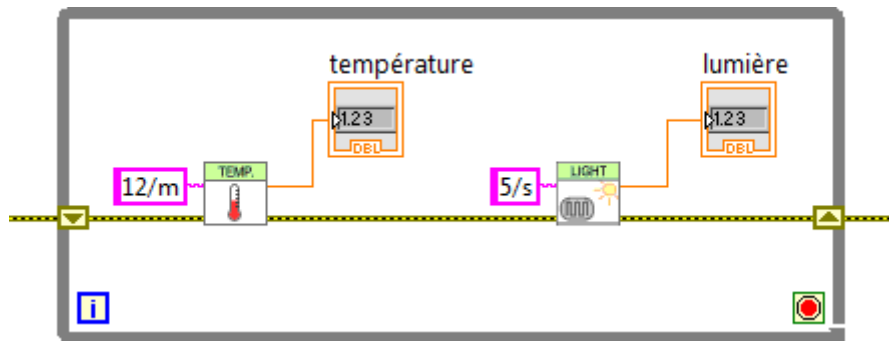


Le paramètre d'entrée *update freq* est une chaîne de caractères qui permet de configurer la façon dont la valeur de sortie est mis à jour:

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

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

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

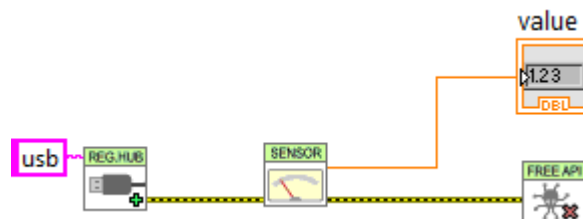


Changement de la fréquence de mise à jour du même module

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

11.5. Fonctionnement et utilisation des VIs

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

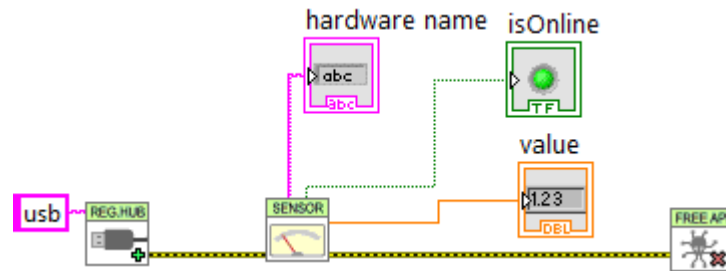


Exemple minimal d'utilisation de l'API Yoctopuce pour LabVIEW

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

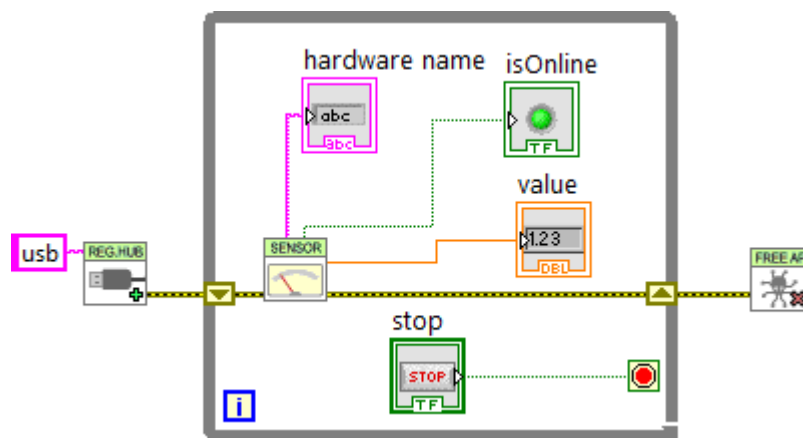
1. Il initialise l'API en mode natif ("usb") avec le VI `YRegisterHub`
2. Il affiche la valeur du premier capteur Yoctopuce qu'il trouve à l'aide du VI `YSensor`
3. Il libère l'API grâce au VI `YFreeAPI`

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



Utilisation des sorties hardware name et isOnline

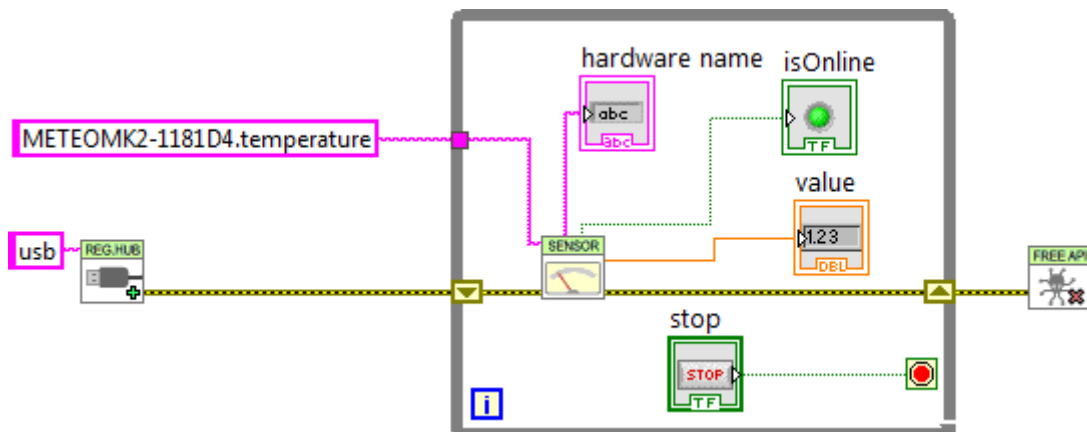
Les VIs de l'API Yoctopuce ne sont qu'une porte d'entrée sur la mécanique interne de la librairie Yoctopuce. Cette mécanique fonctionne indépendamment des VIs Yoctopuce. En effet, la plupart des communications avec les modules électroniques sont gérées automatiquement en arrière plan. C'est pourquoi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



Les VIs Yoctopuce peuvent être utilisés dans une boucle non temporisée

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

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

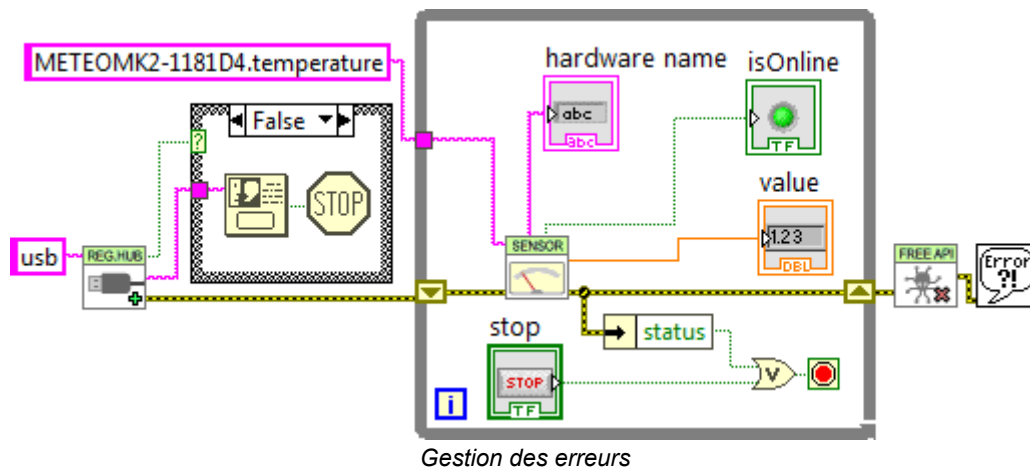


Utilisation de noms pour identifier les fonctions à utiliser

Gestion des erreurs

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

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

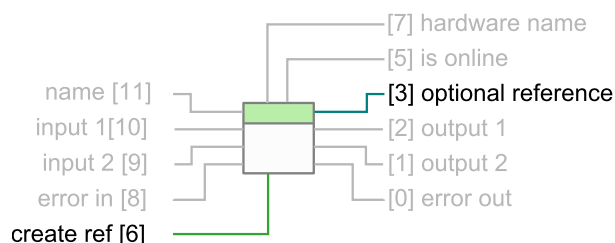


Le message d'erreur le plus courant est "*Another process is already using yAPI*". Il signifie qu'une autre application, LabVIEW ou autre, utilise déjà l'API en module USB natif. En effet, pour des raisons techniques, l'API USB native ne peut être utilisée que par une seule application à la fois sur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

11.6. Utilisation des objets Proxy

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

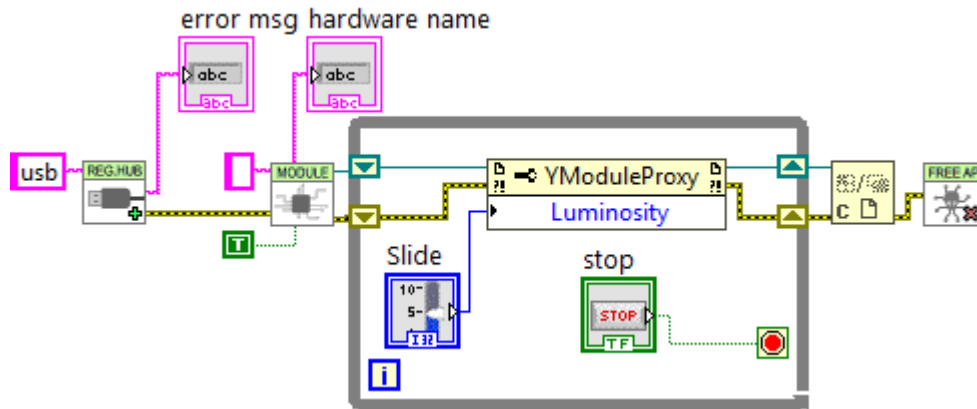
Chaque VI correspondant à une classe dispose de deux connecteurs *create ref* et *optional ref* qui permettent d'obtenir une référence sur l'objet *Proxy* de l'API .NET Proxy sur laquelle est construite la librairie LabVIEW.



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

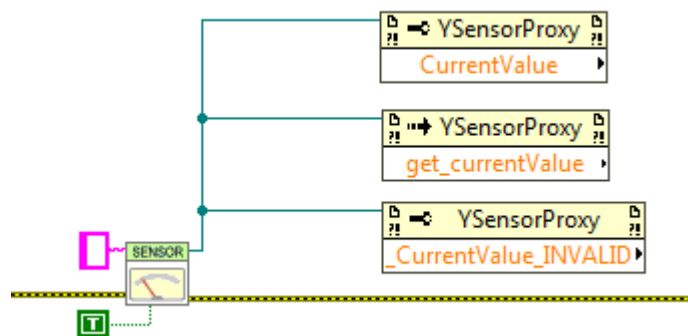
Pour obtenir cette référence, il suffit de mettre *optional ref* à TRUE. Attention, il est impératif de fermer toute référence créée de cette manière, sous peine de saturer rapidement la mémoire de l'ordinateur.

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



Contrôle de la luminosité des LEDs d'un module

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

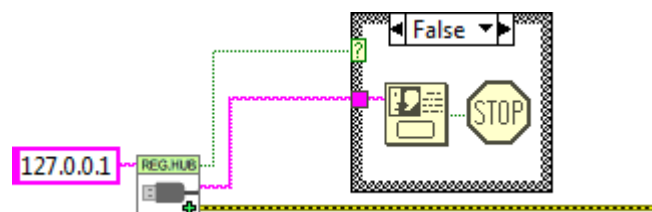


Noeuds Property et Invoke: Utilisation de propriétés, méthodes et constantes

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

Utilisation en réseau

Sur une même machine, il ne peut y avoir qu'un seul processus qui accède aux modules Yoctopuce locaux directement par USB (url mis à "usb"). Par contre, plusieurs processus peuvent se connecter en parallèle à des YoctoHubs⁷ ou à une machine sur laquelle tourne le logiciel *VirtualHub*⁸, y compris la machine locale. Si vous utilisez l'adresse réseau locale de votre machine (127.0.0.1) et qu'un *VirtualHub* tourne dessus, vous pourrez ainsi contourner la limitation qui empêche l'utilisation en parallèle de l'API native USB.

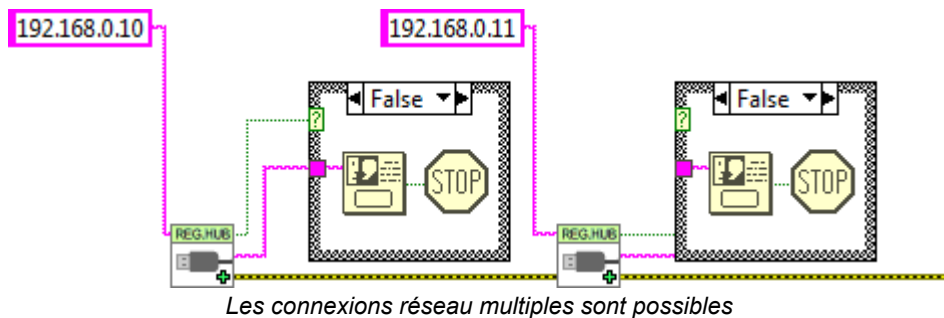


Utilisation en mode réseau

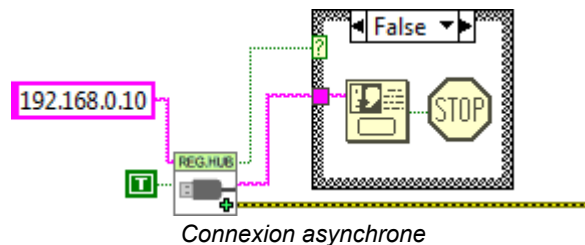
⁷ www.yoctopuce.com/FR/products/category/extensions-et-reseau

⁸ www.yoctopuce.com/FR/virtualhub.php

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



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

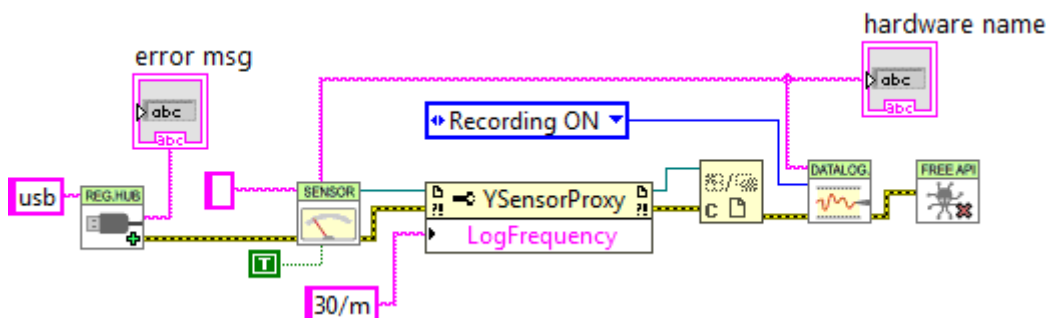


11.7. Gestion du datalogger

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

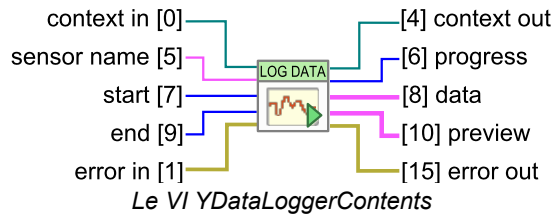
Enregistrement

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



Lecture

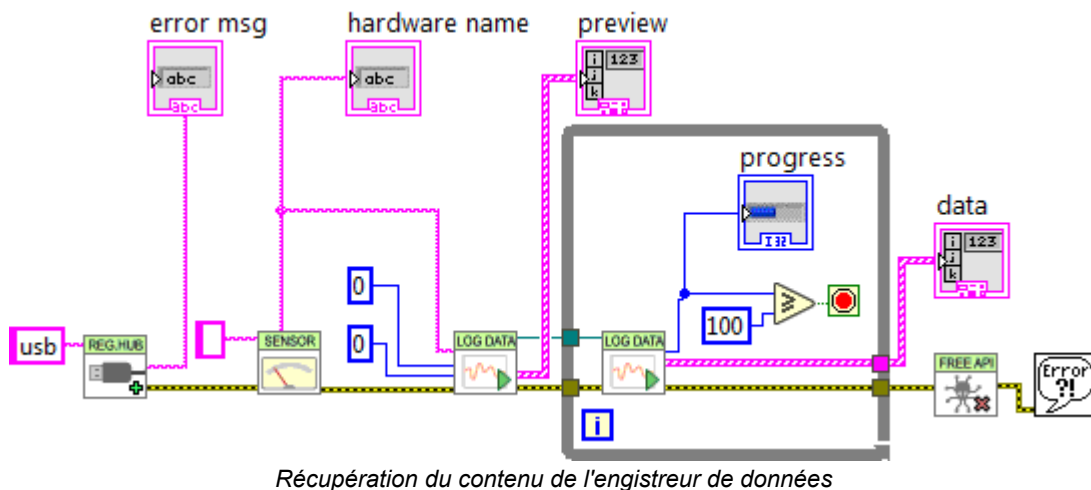
La récupération des données de l'enregistreur se fait à l'aide du VI YDataLoggerContents.



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

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

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



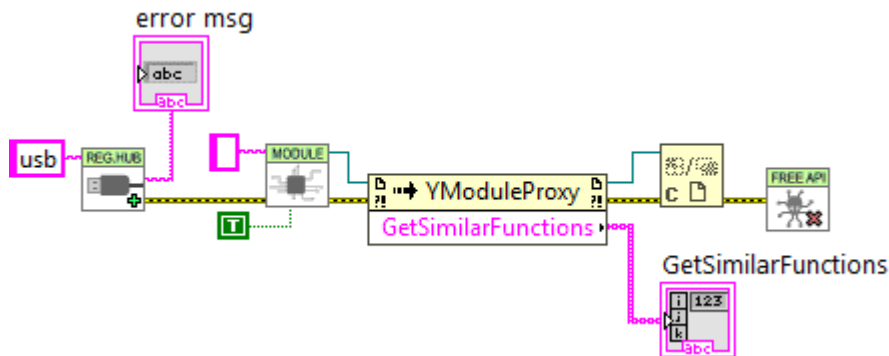
Les résultats et le résumé sont rendus sous la forme d'un tableau de structures qui contiennent les champs suivants:

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

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

11.8. Énumération de fonctions

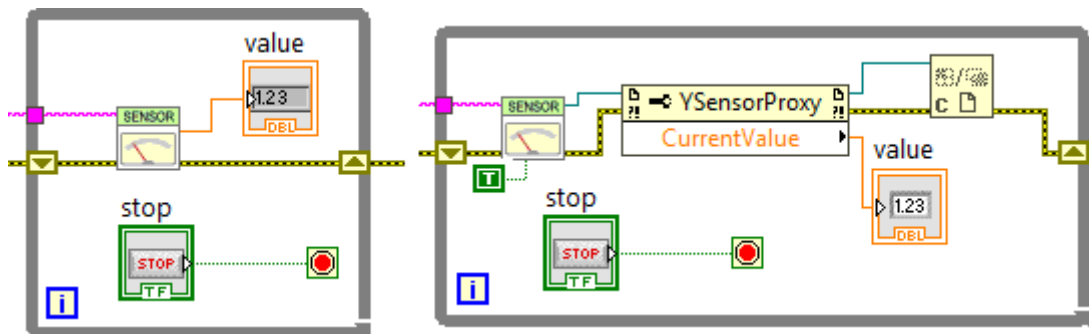
Chaque VI correspondant à un objet de l'API .NET Proxy permet de faire une énumération de toutes les fonctions de la même classe via la méthode *getSimilarfunctions()* de l'objet Proxy correspondant. Ainsi il est ainsi aisé de faire un inventaire de tous les modules connectés, de tous les capteurs connectés,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



Récupération de la liste de tous les modules connectés

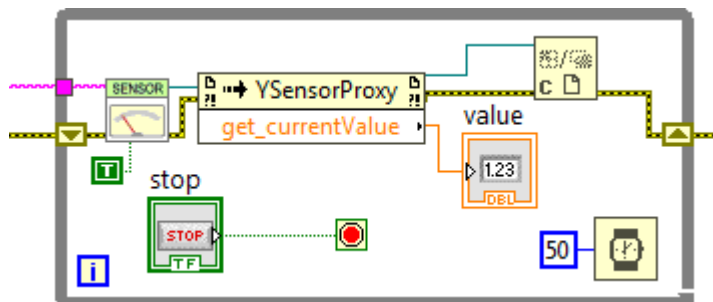
11.9. Un mot sur les performances

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



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

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

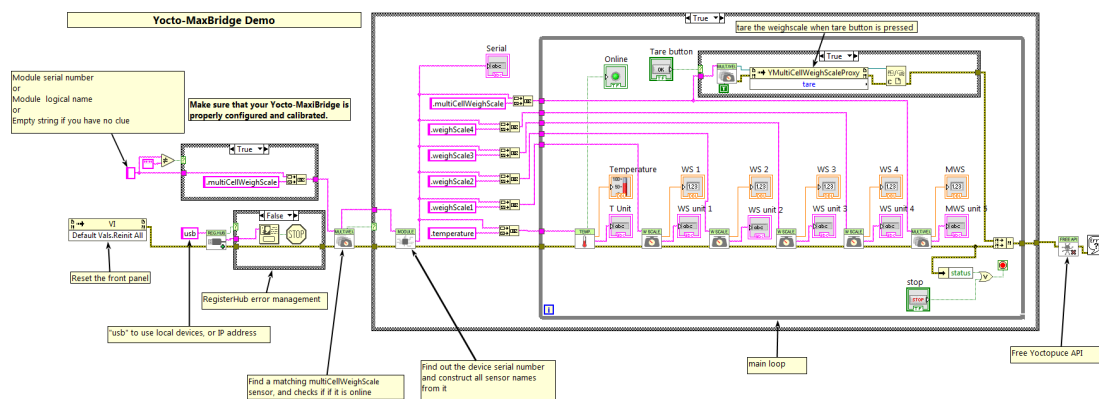


Cette boucle, qui utilise une méthode, doit être ralentie

11.10. Un exemple complet de programme LabVIEW

Voici un exemple qui illustre l'utilisation d'un Yocto-MaxiBridge-C dans LabVIEW. Après un appel au VI *RegisterHub*, le VI *YMultiCellWeighScale* trouve le premier multi-capteur de pression disponible, et utilise le VI *YModule* pour trouver le numéro de série du module. Ce numéro de série est utilisé pour construire le nom hardware de tous les autres capteurs hébergés par le module. Ces noms sont utilisés comme paramètres pour initialiser les VI correspondant à chaque capteur. Cette technique évite les ambiguïtés au cas où plusieurs Yocto-MaxiBridge-C seraient branchés. Une fois les VIs correspondants aux capteurs initialisés, il ne reste plus qu'à afficher leur valeur. A chaque fois que le bouton *tare*, une référence sur l'objet *YWeighScaleProxy* permet d'appeler la méthode *tare*. Cette

référence est ensuite refermée aussitôt. Une fois l'application terminée, l'API Yoctopuce est libérée à l'aide du VI `YFreeAPI`.



Exemple d'utilisation du Yocto-MaxiBridge-C dans LabVIEW

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

11.11. Différences avec les autres API Yoctopuce

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

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

YFreeAPI

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

Propriétés

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

Callback vs Propriétés

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

12. Utilisation du Yocto-MaxiBridge-C 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 de VirtualHub pour pouvoir communiquer avec les modules Yoctopuce.

12.1. Préparation

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

- La librairie de programmation pour Java¹
- VirtualHub² pour Windows, macOS 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 VirtualHub et vous pouvez commencer vos premiers tests. Vous n'avez pas besoin d'installer de driver.

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

12.2. Contrôle de la fonction MultiCellWeighScale

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

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

// On récupère l'objet permettant d'interagir avec le module
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.multiCellWeighScale");

// Pour gérer le hot-plug, on vérifie que le module est là
```

¹ www.yoctopuce.com/FR/libraries.php

² www.yoctopuce.com/FR/virtualhub.php

```

if (multicellweighscale.isOnline())
{
    // Utiliser multicellweighscale.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'initialisation se passe mal, une exception sera générée.

YMultiCellWeighScale.FindMultiCellWeighScale

La fonction `YMultiCellWeighScale.FindMultiCellWeighScale` permet de retrouver un capteur de poids multi-cellules 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-MaxiBridge-C avec le numéro de série *WMBRDG1C-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *multiCellWeighScale* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```

multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.multiCellWeighScale")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.MaFonction")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("MonModule.multiCellWeighScale")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MonModule.MaFonction")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MaFonction")

```

`YMultiCellWeighScale.FindMultiCellWeighScale` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur de poids multi-cellules.

isOnline

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

set_excitation

La méthode `set_excitation()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'activer sur le Yocto-MaxiBridge-C la source de tension permettant d'effectuer la mesure ratiométrique (mise sous tension de la cellule de charge et de force). L'utilisation d'une excitation alternative (AC) donne les meilleurs résultats pour une cellule de poids standard.

tare

La méthode `tare()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet de calibrer la position au repos de la balance. Elle doit être appelée lorsque la balance n'est pas en charge.

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'obtenir la valeur actuelle mesurée par le Yocto-MaxiBridge-C, calculée comme la somme de toutes les entrées actives du module. La valeur de poids dépend bien entendu de la configuration des paramètres des cellules de poids, qui devrait être faite précédemment pour obtenir une mesure correcte.

Un exemple réel

Lancez votre environnement java et ouvrez le projet correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-MaxiBridge-C** 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.YAPI;
import com.yoctopuce.YoctoAPI.YAPI_Exception;
import com.yoctopuce.YoctoAPI.YMultiCellWeighScale;

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);
        }

        YMultiCellWeighScale sensor;
        if (args.length > 0) {
            sensor = YMultiCellWeighScale.FindMultiCellWeighScale(args[0]);
        } else {
            sensor = YMultiCellWeighScale.FirstMultiCellWeighScale();
        }
        if (sensor == null) {
            System.out.println("No module connected (check USB cable)");
            System.exit(1);
        }
        try {
            if (sensor.isOnline()) {
                // On startup, enable excitation and tare weigh scale
                System.out.println("Resetting tare weight...");
                sensor.set_excitation(YMultiCellWeighScale.EXCITATION_AC);
                YAPI.Sleep(3000);
                sensor.tare();
            }

            while (sensor.isOnline()) {
                System.out.println("Weight : " + sensor.get_currentValue() + " " +
sensor.get_unit());
                System.out.println("  (press Ctrl-C to exit)");
                YAPI.Sleep(1000);
            }
        } catch (YAPI_Exception ex) {
            System.out.println("Module " + sensor.describe() + " disconnected (check
identification and USB cable)");
        }
        YAPI.FreeAPI();
    }
}
```

12.3. Contrôle de la partie module

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

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

public class Demo {
```

```

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

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

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

```

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

Modifications des réglages du module

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

```

import com.yoctopuce.YoctoAPI.*;

public class Demo {

    public static void main(String[] args)

```



```

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

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

    YModule m;
    String newname;

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

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

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

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

    YAPI.FreeAPI();
}
}

```

Attention, le nombre de cycles d'écriture de la mémoire non volatile du module est limité. Passé cette limite plus rien ne garantit que la sauvegarde des réglages se passera correctement. Cette limite, liée à la technologie employée par le micro-processeur du module se situe aux 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();
}
```

12.4. Gestion des erreurs

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

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

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

13. Utilisation du Yocto-MaxiBridge-C avec Android

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

13.1. Accès Natif et VirtualHub

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 VirtualHub tournant sur un autre OS.

13.2. Préparation

Connectez-vous sur le site de Yoctopuce et téléchargez la librairie de programmation pour Java pour 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.

13.3. Compatibilité

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

¹ www.yoctopuce.com/FR/libraries.php

Version d'Android

Notre librairie peut être compilée pour fonctionner avec les anciennes versions aussi longtemps que les outils Android nous permettent de les supporter, soit environ les versions des dix dernières années.

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.

13.4. 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" />
...
</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>

```

13.5. Contrôle de la fonction MultiCellWeighScale

Il suffit de quelques lignes de code pour piloter un Yocto-MaxiBridge-C. Voici le squelette d'un fragment de code Java qui utilise la fonction `MultiCellWeighScale`.

```

[...]
```

```

// On active la détection des modules sur USB
YAPI.EnableUSBHost(this);
YAPI.RegisterHub("usb");
[...]
```

```

// On récupère l'objet permettant de communiquer avec le module
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale(
    ("WMBRDG1C-123456.multiCellWeighScale"));

// Pour gérer le hot-plug, on vérifie que le module est là
if (multicellweighscale.isOnline())
{
    // Utilisez multicellweighscale.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.

YMultiCellWeighScale.FindMultiCellWeighScale

La fonction `YMultiCellWeighScale.FindMultiCellWeighScale` permet de retrouver un capteur de poids multi-cellules 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-MaxiBridge-C avec le numéros de série *WMBRDG1C-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *multiCellWeighScale* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.multiCellWeighScale")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.MaFonction")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("MonModule.multiCellWeighScale")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MonModule.MaFonction")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MaFonction")
```

`YMultiCellWeighScale.FindMultiCellWeighScale` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur de poids multi-cellules.

isOnline

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

set_excitation

La méthode `set_excitation()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'activer sur le Yocto-MaxiBridge-C la source de tension permettant d'effectuer la mesure ratiométrique (mise sous tension de la cellule de charge et de force). L'utilisation d'une excitation alternative (AC) donne les meilleurs résultats pour une cellule de poids standard.

tare

La méthode `tare()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet de calibrer la position au repos de la balance. Elle doit être appelée lorsque la balance n'est pas en charge.

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'obtenir la valeur actuelle

mesurée par le Yocto-MaxiBridge-C, calculée comme la somme de toutes les entrées actives du module. La valeur de poids dépend bien entendu de la configuration des paramètres des cellules de poids, qui devrait être faite précédemment pour obtenir une mesure correcte.

Un exemple réel

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

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

```
package com.yoctopuce.doc_examples;

import android.app.Activity;
import android.os.Bundle;
import android.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.YMultiCellWeighScale;

public class GettingStarted_Yocto_MaxiBridge extends Activity implements
OnItemClickListener
{
    private YMultiCellWeighScale _multiCellWeighScale = null;
    private ArrayAdapter<String> aa;
    private Handler handler = null;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.gettingstarted_yocto_maxibridge);
        Spinner my_spin = (Spinner) findViewById(R.id.spinner1);
        my_spin.setOnItemClickListener(this);
        aa = new ArrayAdapter<>(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");
            YMultiCellWeighScale wc = YMultiCellWeighScale.FirstMultiCellWeighScale();
            while (wc != null) {
                String hwid = wc.get_hardwareId();
                aa.add(hwid);
                wc = wc.nextMultiCellWeighScale();
            }
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
        // refresh Spinner with detected relay
        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)
{
    String hwid = parent.getItemAtPosition(pos).toString();
    _multiCellWeighScale = YMultiCellWeighScale.FindMultiCellWeighScale(hwid);
    // On startup, enable excitation and tare weigh scale
    try {
        _multiCellWeighScale.set_excitation(YMultiCellWeighScale.EXCITATION_AC);
    } catch (YAPI_Exception e) {
        e.printStackTrace();
    }
}

@Override
public void onNothingSelected(AdapterView<?> arg0)
{
    _multiCellWeighScale = null;
}

final Runnable r = new Runnable()
{
    public void run()
    {
        if (_multiCellWeighScale != null) {
            try {
                TextView view = (TextView) findViewById(R.id.weightfield);
                view.setText(String.format("%.1f %s",
                    _multiCellWeighScale.getCurrentValue(), _multiCellWeighScale.getUnit()));
            } catch (YAPI_Exception e) {
                e.printStackTrace();
            }
        }
        handler.postDelayed(this, 1000);
    }
};

public void onTare(View view)
{
    if (_multiCellWeighScale != null) {
        try {
            _multiCellWeighScale.tare();
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
    }
}
}

```

13.6. Contrôle de la partie module

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

```

package com.yoctopuce.doc_examples;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.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.setOnItemSelectedListener(this);
        aa = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item);
        aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        my_spin.setAdapter(aa);
    }

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

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

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

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

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

    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int pos, long id)

```

```

    {
        String hwid = parent.getItemAtPosition(pos).toString();
        module = YModule.FindModule(hwid);
        DisplayModuleInfo();
    }

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

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

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

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

```

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

Modifications des réglages du module

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

```

package com.yoctopuce.doc_examples;

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

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

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

```

```

@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.savesettings);
    Spinner my_spin = (Spinner) findViewById(R.id.spinner1);
    my_spin.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 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 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();
}
}

```

13.7. Gestion des erreurs

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

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

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

14. Utilisation du Yocto-MaxiBridge-C en TypeScript

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

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

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

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

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

14.1. Utiliser la librairie Yoctopuce pour TypeScript

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

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

```
npm install -g typescript
```

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

- La librairie de programmation pour TypeScript¹
- Le programme VirtualHub² pour Windows, macOS ou Linux selon l'OS que vous utilisez. En effet, TypeScript et JavaScript font partie de ces langages qui ne vous permettront pas d'accéder directement aux périphériques USB. C'est pourquoi si vous désirez travailler avec des modules branchés par USB, vous devrez faire tourner la passerelle de Yoctopuce appelée VirtualHub sur la machine à laquelle sont branchés les modules. Vous n'avez en revanche pas besoin d'installer de driver.

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

```
npm install
```

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

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

14.2. Petit rappel sur les fonctions asynchrones en JavaScript

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

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

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

¹ www.yoctopuce.com/FR/libraries.php

² www.yoctopuce.com/FR/virtualhub.php

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

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

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

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

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

14.3. Contrôle de la fonction `MultiCellWeighScale`

Il suffit de quelques lignes de code pour piloter un Yocto-MaxiBridge-C. Voici le squelette d'un fragment de code TypeScript qui utilise la fonction `MultiCellWeighScale`.

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

[...]
```

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

// On récupère l'objet permettant d'interagir avec le module
`let multicellweighscale: YMultiCellWeighScale =`
`YMultiCellWeighScale.FindMultiCellWeighScale("WMBRDG1C-123456.multiCellWeighScale");`

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

Voyons maintenant en détail ce que font ces quelques lignes.

Import de `yocto_api` et `yocto_multicellweighscale`

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

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

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

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

YAPI.RegisterHub

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

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

YMultiCellWeighScale.FindMultiCellWeighScale

La méthode `FindMultiCellWeighScale` permet de retrouver un capteur de poids multi-cellules 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-MaxiBridge-C avec le numéros de série `WMBRDG1C-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `multiCellWeighScale` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.multiCellWeighScale")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.MaFonction")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("MonModule.multiCellWeighScale")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MonModule.MaFonction")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MaFonction")
```

`YMultiCellWeighScale.FindMultiCellWeighScale` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur de poids multi-cellules.

isOnline

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

set_excitation

La méthode `set_excitation()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'activer sur le Yocto-MaxiBridge-C la source de tension permettant d'effectuer la mesure ratiométrique (mise sous tension de la cellule de charge et de force). L'utilisation d'une excitation alternative (AC) donne les meilleurs résultats pour une cellule de poids standard.

tare

La méthode `tare()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet de calibrer la position au repos de la balance. Elle doit être appelée lorsque la balance n'est pas en charge.

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'obtenir la valeur actuelle mesurée par le Yocto-MaxiBridge-C, calculée comme la somme de toutes les entrées actives du module. La valeur de poids dépend bien entendu de la configuration des paramètres des cellules de poids, qui devrait être faite précédemment pour obtenir une mesure correcte.

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-MaxiBridge-C** de la librairie Yoctopuce pour TypeScript. Vous y trouverez un fichier nommé `demo.ts` avec le code d'exemple ci-dessous, qui reprend les fonctions expliquées précédemment, mais cette fois utilisées avec le décorum nécessaire à en faire un petit programme d'exemple concret.

Si le Yocto-MaxiBridge-C 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-MaxiBridge-C et où vous avez lancé le VirtualHub.

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

let sensor: YMultiCellWeighScale;

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

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

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

    if (await sensor.isOnline()) {
        // On startup, enable excitation and tare weigh scale
        console.log('Resetting tare weight...');
        await sensor.set_excitation(YMultiCellWeighScale.EXCITATION_AC);
        await YAPI.Sleep(3000);
        await sensor.tare();
    }

    refresh();
}

async function refresh(): Promise<void>
{
    if (await sensor.isOnline()) {
        console.log('Weight : '+(await sensor.get_currentValue())
            + (await sensor.get_unit()));
    } else {
        console.log('Module not connected');
    }
    setTimeout(refresh, 500);
}
```

```
startDemo();
```

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

```
npm install
```

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

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

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

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

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

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

```
npm install yoctolib-cjs
```

Le même exemple, mais dans un navigateur

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

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

```
npm run app-server
```

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

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

14.4. Contrôle de la partie module

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

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

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

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

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

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

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

Modifications des réglages du module

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

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

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

```

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

// Select the device to use
let module: YModule = YModule.FindModule(args[0]);
if(await module.isOnline()) {
    if(args.length > 1) {
        let newname: string = args[1];
        if (!await YAPI.CheckLogicalName(newname)) {
            console.log("Invalid name (" + newname + ")");
            process.exit(1);
        }
        await module.set_logicalName(newname);
        await module.saveToFlash();
    }
    console.log('Current name: '+await module.get_logicalName());
} else {
    console.log("Module not connected (check identification and USB cable)\n");
}
await YAPI.FreeAPI();
}

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

```

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

Énumération des modules

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

```

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

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

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

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

        let module = YModule.FirstModule();
        while(module) {
            let line: string = await module.get_serialNumber();
            line += '(' + (await module.get_productName()) + ')';
            console.log(line);
            module = module.nextModule();
        }
    }
}

```

```

    }
    setTimeout(refresh, 500);
  } catch(e) {
    console.log(e);
  }
}

startDemo();

```

14.5. Gestion des erreurs

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

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

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

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

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

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

15. Utilisation du Yocto-MaxiBridge-C 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` dès lors que `async / await` sont standardisés.

15.1. Fonctions bloquantes et fonctions asynchrones en JavaScript

JavaScript a été conçu pour éviter toute situation de *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.

15.2. Utiliser la librairie Yoctopuce pour JavaScript / EcmaScript 2017

JavaScript fait partie de ces langages qui ne vous permettront pas d'accéder directement aux couches matérielles de votre ordinateur. C'est pourquoi si vous désirez travailler avec des modules USB branchés par USB, vous devrez faire tourner la passerelle de Yoctopuce appelée VirtualHub sur la machine à laquelle sont branchés les modules.

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

- La librairie de programmation pour Javascript / EcmaScript 2017¹
- VirtualHub² pour Windows, macOS 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);
```

15.3. Contrôle de la fonction MultiCellWeighScale

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

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

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

// On récupère l'objet permettant d'interagir avec le module
let multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale(
    "WMBRDG1C-123456.multiCellWeighScale");

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

Voyons maintenant en détail ce que font ces quelques lignes.

Require de yocto_api et yocto_multicellweighscale

Ces deux imports permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api` doit toujours être inclus, `yocto_multicellweighscale` est nécessaire pour gérer les modules contenant un capteur de poids multi-cellules, comme le Yocto-MaxiBridge-C. 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.

YMultiCellWeighScale.FindMultiCellWeighScale

La méthode `FindMultiCellWeighScale` permet de retrouver un capteur de poids multi-cellules 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-MaxiBridge-C avec le numéros de série *WMBRDG1C-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *multiCellWeighScale* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.multiCellWeighScale")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.MaFonction")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("MonModule.multiCellWeighScale")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MonModule.MaFonction")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MaFonction")
```

`YMultiCellWeighScale.FindMultiCellWeighScale` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur de poids multi-cellules.

isOnline

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

set_excitation

La méthode `set_excitation()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'activer sur le Yocto-MaxiBridge-C la source de tension permettant d'effectuer la mesure ratiométrique (mise sous tension de la cellule de charge et de force). L'utilisation d'une excitation alternative (AC) donne les meilleurs résultats pour une cellule de poids standard.

tare

La méthode `tare()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet de calibrer la position au repos de la balance. Elle doit être appelée lorsque la balance n'est pas en charge.

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'obtenir la valeur actuelle mesurée par le Yocto-MaxiBridge-C, calculée comme la somme de toutes les entrées actives du module. La valeur de poids dépend bien entendu de la configuration des paramètres des cellules de poids, qui devrait être faite précédemment pour obtenir une mesure correcte.

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-MaxiBridge-C** 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-MaxiBridge-C 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-MaxiBridge-C et où vous avez lancé le VirtualHub.

```
"use strict";

require('yoctolib-es2017/yocto_api.js');
require('yoctolib-es2017/yocto_multicellweighscale.js');
```

```

let sensor;

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 = YMultiCellWeighScale.FirstMultiCellWeighScale();
        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);
    sensor = YMultiCellWeighScale.FindMultiCellWeighScale(serial+".multiCellWeighScale");

    if (await sensor.isOnline()) {
        // On startup, enable excitation and tare weigh scale
        cout << "Resetting tare weight..." << endl;
        sensor.set_excitation(YMultiCellWeighScale.EXCITATION_AC);
        ySleep(3000);
        sensor.tare();
    }

    refresh();
}

async function refresh()
{
    if (await sensor.isOnline()) {
        console.log('Weight : '+ (await sensor.get_currentValue()) + (await sensor.get_unit(
    )));
    } 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-MaxiBridge-C**. 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_multicellweighscale.js"></script>
  <script>
    let sensor = null;

    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 = YMultiCellWeighScale.FirstMultiCellWeighScale();
        if(anysensor) {
          let module = await anysensor.module();
          serial = await module.get_serialNumber();
          document.getElementById('serial').value = serial;
        }
      }
      sensor = YMultiCellWeighScale.FindMultiCellWeighScale(serial+".multiCellWeighScale");

      if (await sensor.isOnline()) {
        if(await sensor.get_excitation() == YMultiCellWeighScale.EXCITATION_OFF) {
          await sensor.set_excitation(YMultiCellWeighScale.EXCITATION_AC);
        }
        document.getElementById('msg').value = '';
        document.getElementById("sensor-val").value = (await sensor.get_currentValue()) +
(await sensor.get_unit());
      } else {
        document.getElementById('msg').value = 'Module not connected';
      }
      setTimeout(refresh, 500);
    }

    window.tare = function()
    {
      sensor.tare();
    };

    startDemo();
  </script>
</head>
<body>
Module to use: <input id='serial'>
<input id='msg' style='color:red;border:none;' readonly><br>
Weight : <input id='sensor-val' readonly><a href='javascript:tare();' >Tare</a><br>
</body>
</html>
```

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

15.4. Contrôle de la partie module

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

```
"use strict";

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

async function startDemo(args)
{
    await YAPI.LogUnhandledPromiseRejections();

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

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

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

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

```

15.5. Gestion des erreurs

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

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

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

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

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

Pour démarrer vos essais en PHP, vous allez avoir besoin d'un serveur PHP 7.1 ou plus récent 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.

16.1. Préparation

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

- La librairie de programmation pour PHP²
- VirtualHub³ pour Windows, macOS ou Linux selon l'OS que vous utilisez

Notre librairie PHP est basée sur PHP 8.x. C'est-à-dire que notre librairie fonctionne parfaitement avec n'importe quelle version de PHP actuellement encore supportée. Toutefois, afin de ne pas abandonner nos clients qui ont des installations plus anciennes, nous maintenons une version compatible avec PHP 7.1. qui date de 2016.

Par ailleurs, nous proposons également une version de la librairie qui suit les recommandations PSR. Pour simplifier, cette version est de même code que la version php8 mais chaque classe est stockée dans un fichier séparé. De plus, cette version utilise un namespace `Yoctopuce\YoctoAPI`. Ces changements rendent notre librairie beaucoup plus facilement utilisable avec des installations qui utilisent l'autoload.

Notez que les exemples de la documentation n'utilisent pas la version PSR.

¹ Quelques serveurs PHP gratuits: easyPHP pour Windows, MAMP pour macOS

² www.yoctopuce.com/FR/libraries.php

³ www.yoctopuce.com/FR/virtualhub.php

Dans l'archive de la librairie, il y a donc trois sous-répertoire :

- php7
- php8
- phpPSR

Choisissez le bon répertoire en fonction de la version de la librairie que vous souhaitez utiliser, décompressez les fichiers de ce répertoire dans un répertoire de votre choix accessible à votre serveur web, branchez vos modules, lancez VirtualHub, et vous pouvez commencer vos premiers tests. Vous n'avez pas besoin d'installer de driver.

16.2. Contrôle de la fonction MultiCellWeighScale

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

```
include('yocto_api.php');
include('yocto_multicellweighscale.php');

[...]
// On active l'accès aux modules locaux à travers le VirtualHub
YAPI::RegisterHub('http://127.0.0.1:4444/', $errmsg);
[...]

// On récupère l'objet permettant d'interagir avec le module
$multicellweighscale = YMultiCellWeighScale::FindMultiCellWeighScale(
    "WMBRDG1C-123456.multiCellWeighScale");

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.php et yocto_multicellweighscale.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_multicellweighscale.php` est nécessaire pour gérer les modules contenant un capteur de poids multi-cellules, comme le Yocto-MaxiBridge-C.

YAPI::RegisterHub

La fonction `YAPI::RegisterHub` 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.

YMultiCellWeighScale::FindMultiCellWeighScale

La fonction `YMultiCellWeighScale::FindMultiCellWeighScale` permet de retrouver un capteur de poids multi-cellules 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-MaxiBridge-C avec le numéros de série `WMBRDG1C-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *multiCellWeighScale* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
$multicellweighscale = YMultiCellWeighScale::FindMultiCellWeighScale(
    "WMBRDG1C-123456.multiCellWeighScale");
$multicellweighscale = YMultiCellWeighScale::FindMultiCellWeighScale
```

```

("WMBRDG1C-123456.MaFonction");
$multicellweighscale = YMultiCellWeighScale::FindMultiCellWeighScale
("MonModule.multiCellWeighScale");
$multicellweighscale = YMultiCellWeighScale::FindMultiCellWeighScale("MonModule.MaFonction"
);
$multicellweighscale = YMultiCellWeighScale::FindMultiCellWeighScale("MaFonction");

```

`YMultiCellWeighScale::FindMultiCellWeighScale` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur de poids multi-cellules.

isOnline

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

set_excitation

La méthode `set_excitation()` de l'objet renvoyée par `yFindMultiCellWeighScale` permet d'activer sur le Yocto-MaxiBridge-C la source de tension permettant d'effectuer la mesure ratiométrique (mise sous tension de la cellule de charge et de force). L'utilisation d'une excitation alternative (AC) donne les meilleurs résultats pour une cellule de poids standard.

tare

La méthode `tare()` de l'objet renvoyée par `yFindMultiCellWeighScale` permet de calibrer la position au repos de la balance. Elle doit être appelée lorsque la balance n'est pas en charge.

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyée par `yFindMultiCellWeighScale` permet d'obtenir la valeur actuelle mesurée par le Yocto-MaxiBridge-C, calculée comme la somme de toutes les entrées actives du module. La valeur de poids dépend bien entendu de la configuration des paramètres des cellules de poids, qui devrait être faite précédemment pour obtenir une mesure correcte.

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 browser favori. Vous trouverez aussi ce code dans le répertoire **Examples/Doc-GettingStarted-Yocto-MaxiBridge-C** 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('../php8/yocto_api.php');
include('../php8/yocto_multicellweighscale.php');

// Use explicit error handling rather than exceptions
YAPI::DisableExceptions();

// Setup the API to use the VirtualHub on local machine
if(YAPI::RegisterHub('http://127.0.0.1:4444/', $errmsg) != YAPI::SUCCESS) {
    die("Cannot contact VirtualHub on 127.0.0.1");
}

@$serial = $_GET['serial'];
if ($serial != '') {
    // Check if a specified module is available online
    $sensor = YMultiCellWeighScale::FindMultiCellWeighScale("$serial.multiCellWeighScale"

```

⁴ Si vous n'avez pas d'éditeur de texte, utilisez Notepad plutôt que Microsoft Word.

```

);
    if (!$sensor->isOnline()) {
        die("Module not connected (check serial and USB cable)");
    }
} else {
    // or use any connected module suitable for the demo
    $sensor = YMultiCellWeighScale::FirstMultiCellWeighScale();
    if(is_null($sensor)) {
        die("No module connected (check USB cable)");
    } else {
        $serial = $sensor->module()->get_serialnumber();
    }
}
Print("Module to use: <input name='serial' value='$serial'><br>");
$sensor = YMultiCellWeighScale::FindMultiCellWeighScale("$serial.multiCellWeighScale");

if($sensor->get_excitation() == Y_EXCITATION_OFF) {
    $sensor->set_excitation(Y_EXCITATION_AC);
}
Printf("Weight: %.1f %s<br>", $sensor->get_currentValue(), $sensor->get_unit());

YAPI::FreeAPI();

// 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>

```

16.3. Contrôle de la partie module

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

```

<HTML>
<HEAD>
<TITLE>Module Control</TITLE>
</HEAD>
<BODY>
<FORM method='get'>
<?php
    include('../..//php8/yocto_api.php');

    // Use explicit error handling rather than exceptions
    YAPI::DisableExceptions();

    // Setup the API to use the VirtualHub on local machine
    if(YAPI::RegisterHub('http://127.0.0.1:4444/', $errmsg) != YAPI::SUCCESS) {
        die("Cannot contact VirtualHub on 127.0.0.1 : ".$errmsg);
    }

    @$serial = $_GET['serial'];
    if ($serial != '') {
        // Check if a specified module is available online
        $module = YModule::FindModule("$serial");
        if (!$module->isOnline()) {
            die("Module not connected (check serial and USB cable)");
        }
    } else {
        // or use any connected module suitable for the demo
        $module = YModule::FirstModule();
        if($module) { // skip VirtualHub
            $module = $module->nextModule();
        }
        if(is_null($module)) {
            die("No module connected (check USB cable)");
        } else {
            $serial = $module->get_serialnumber();
        }
    }
}

```



```

Print("Module to use: <input name='serial' value='$serial'><br>");

if (isset($_GET['beacon'])) {
    if ($_GET['beacon']=='ON')
        $module->set_beacon(Y_BEACON_ON);
    else
        $module->set_beacon(Y_BEACON_OFF);
}
printf('serial: %s<br>', $module->get_serialNumber());
printf('logical name: %s<br>', $module->get_logicalName());
printf('luminosity: %s<br>', $module->get_luminosity());
print('beacon: ');
if($module->get_beacon() == Y_BEACON_ON) {
    printf("<input type='radio' name='beacon' value='ON' checked>ON ");
    printf("<input type='radio' name='beacon' value='OFF'>OFF<br>");
} else {
    printf("<input type='radio' name='beacon' value='ON'>ON ");
    printf("<input type='radio' name='beacon' value='OFF' checked>OFF<br>");
}
printf('upTime: %s sec<br>',intVal($module->get_upTime()/1000));
printf('USB current: %sma<br>', $module->get_usbCurrent());
printf('logs:<br><pre>%s</pre>', $module->get_lastLogs());
YAPI::FreeAPI();
?>
<input type='submit' value='refresh'>
</FORM>
</BODY>
</HTML>

```

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

```

<HTML>
<HEAD>
<TITLE>save settings</TITLE>
<BODY>
<FORM method='get'>
<?php
    include('../..//php8/yocto_api.php');

    // Use explicit error handling rather than exceptions
    YAPI::DisableExceptions();

    // Setup the API to use the VirtualHub on local machine
    if(YAPI::RegisterHub('http://127.0.0.1:4444/', $errmsg) != YAPI::SUCCESS) {
        die("Cannot contact VirtualHub on 127.0.0.1");
    }

    @$serial = $_GET['serial'];
    if ($serial != '') {
        // Check if a specified module is available online
        $module = YModule::FindModule("$serial");
        if (!$module->isOnline()) {
            die("Module not connected (check serial and USB cable)");
        }
    } else {
        // or use any connected module suitable for the demo
        $module = YModule::FirstModule();
        if($module) { // skip VirtualHub
            $module = $module->nextModule();
        }
    }

```

```

        if(is_null($module)) {
            die("No module connected (check USB cable)");
        } else {
            $serial = $module->get_serialnumber();
        }
    }
    Print("Module to use: <input name='serial' value='$serial'><br>");

    if (isset($_GET['newname'])){
        $newname = $_GET['newname'];
        if (!yCheckLogicalName($newname))
            die('Invalid name');
        $module->set_logicalName($newname);
        $module->saveToFlash();
    }
    printf("Current name: %s<br>", $module->get_logicalName());
    print("New name: <input name='newname' value='' maxlength=19><br>");
    YAPI::FreeAPI();
?>
<input type='submit'>
</FORM>
</BODY>
</HTML>

```

Attention, le nombre de cycle d'écriture de la mémoire non volatile du module est limité. Passé cette limite plus rien ne garantit de que la sauvegarde des réglages se passera correctement. Cette limite, lié à la technologie employé par le micro-processeur du module se situe aux 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('../php8/yocto_api.php');
        YAPI::RegisterHub("http://127.0.0.1:4444/");
        $module = YModule::FirstModule();
        while (!is_null($module)) {
            printf("%s (%s)<br>\n", $module->get_serialNumber(),
                $module->get_productName());
            $module=$module->nextModule();
        }
        YAPI::FreeAPI();
    ?>
</TT>
</BODY>
</HTML>

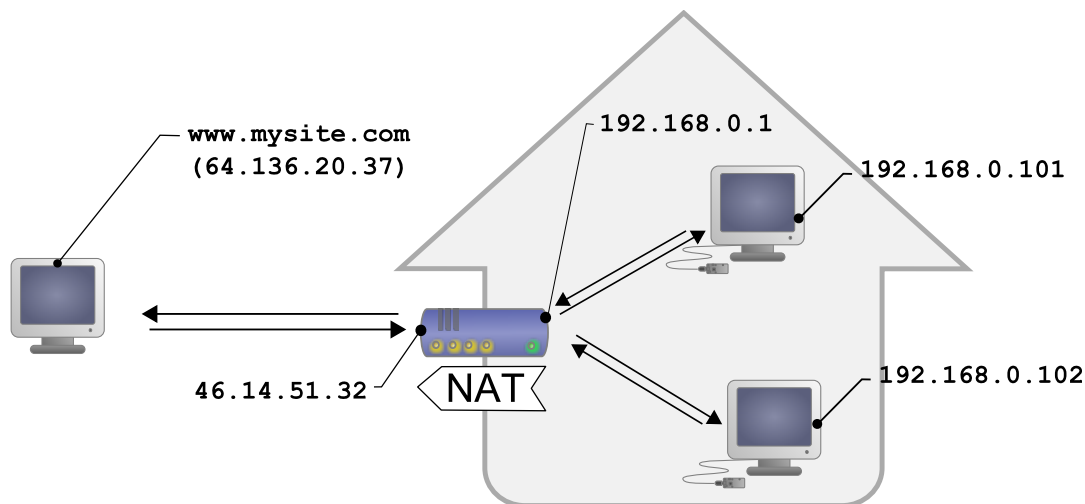
```

16.4. API par callback HTTP et filtres NAT

La librairie PHP est capable de fonctionner dans un mode spécial appelé *Yocto-API par callback HTTP*. Ce mode permet de contrôler des modules Yoctopuce installés derrière un filtre NAT tel qu'un routeur DSL par exemple, et ce sans avoir à ouvrir un port. L'application typique est le contrôle de modules Yoctopuce situés sur réseau privé depuis un site Web publique.

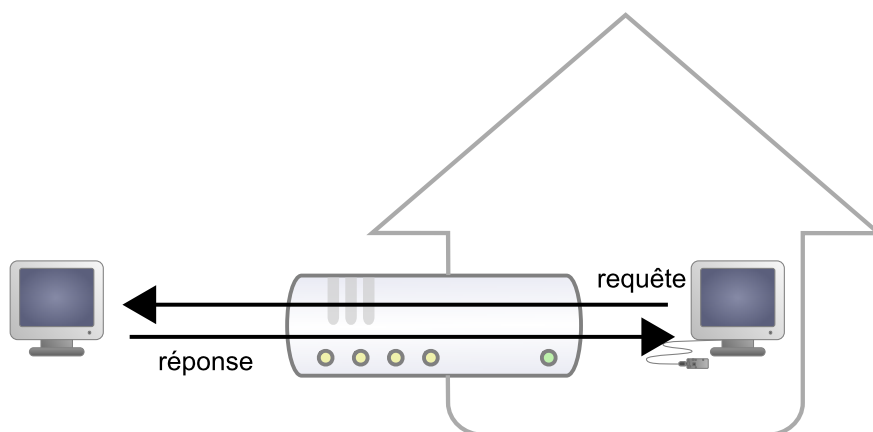
Le filtre NAT, avantages et inconvénients

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.

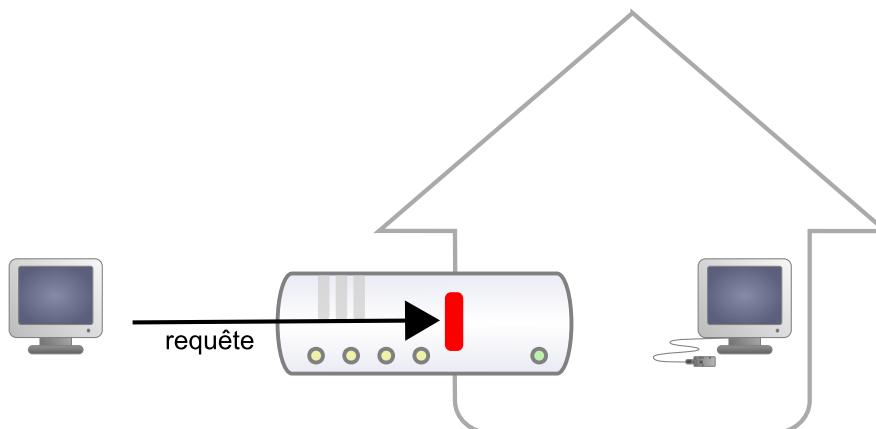


Configuration DSL typique, les machines du LAN sont isolées de l'extérieur par le router DSL

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.



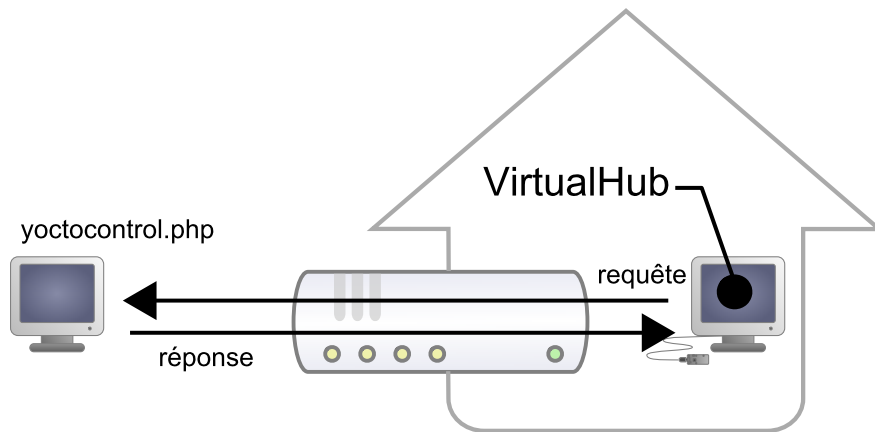
Les réponses aux requêtes venant des machines du LAN sont routées.



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
RELAYH11-00055		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

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

Et choisir "Yocto-API callback".

Il suffit alors de définir l'URL du script PHP et, si nécessaire, le nom d'utilisateur et le mot de passe pour accéder à cette URL. Les méthodes d'authentification supportées sont *basic* et *digest*. La seconde est plus sûre que la première car elle permet de ne pas transférer le mot de passe sur le réseau.

Utilisation

Du point de vue du programmeur, la seule différence se trouve au niveau de l'appel à la fonction `yRegisterHub`; au lieu d'utiliser une adresse IP, il faut utiliser la chaîne *callback* (ou `http://callback`, qui est équivalent).

```
include("yocto_api.php");
yRegisterHub("callback");
```

La suite du code reste strictement identique. Sur l'interface du *VirtualHub*, il y a en bas de la fenêtre de configuration de l'API par callback HTTP un bouton qui permet de tester l'appel au script PHP.

Il est à noter que le script PHP qui contrôle les modules à distance via l'API par callback HTTP ne peut être appelé que par le *VirtualHub*. En effet, il a besoin des informations postées par le *VirtualHub* pour fonctionner. Pour coder un site Web qui contrôle des modules Yoctopuce de manière interactive, il faudra créer une interface utilisateur qui stockera dans un fichier ou une base de données les actions à effectuer sur les modules Yoctopuce. Ces actions seront ensuite lues puis 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.

16.5. Gestion des erreurs

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

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

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

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

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `YAPI.DisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui

peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur `NomDeClasse.STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `NomDeClasse.CURRENTVALUE_INVALID`, etc. Dans tous les cas, la valeur retournée sera du type attendu, et ne sera pas un pointeur nul qui risquerait de faire crasher votre programme. Au pire, si vous affichez la valeur sans la tester, elle sera hors du cadre attendu pour la valeur retournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI.SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

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

17. Utilisation du Yocto-MaxiBridge-C en VisualBasic .NET

VisualBasic a longtemps été la porte d'entrée privilégiée vers le monde Microsoft. Nous nous devions donc d'offrir notre interface pour ce langage, même si la nouvelle tendance est le C#. Nous supportons Visual Studio 2017 et les versions plus récentes.

17.1. Installation

Téléchargez la librairie Yoctopuce pour Visual Basic depuis le site web de Yoctopuce¹. 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.

17.2. Utilisation l'API yoctopuce dans un projet Visual Basic

La librairie Yoctopuce pour Visual Basic .NET se présente sous la forme d'une DLL et de fichiers sources en Visual Basic. La DLL n'est pas une DLL .NET mais une DLL classique, écrite en C, qui gère les communications à bas niveau avec les modules². 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 www.yoctopuce.com/FR/libraries.php

² 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`³. Puis depuis la fenêtre **Explorateur de solutions**, effectuez un clic droit sur la DLL, choisissez **Propriété** et dans le panneau **Propriétés**, mettez l'option **Copier dans le répertoire de sortie à toujours copier**. Vous êtes maintenant prêt à utiliser vos modules Yoctopuce depuis votre environnement Visual Studio.

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

17.3. Contrôle de la fonction MultiCellWeighScale

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

```
[...]
' On active la détection des modules sur USB
Dim errmsg As String
YAPI.RegisterHub("usb", errmsg)
[...]

' On récupère l'objet permettant d'interagir avec le module
Dim multicellweighscale As YMultiCellWeighScale
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.multiCellWeighScale")

' Pour gérer le hot-plug, on vérifie que le module est là
If (multicellweighscale.IsOnline()) Then
    ' Utiliser multicellweighscale.GetCurrentValue()
    [...]
End If

[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

YAPI.RegisterHub

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

YMultiCellWeighScale.FindMultiCellWeighScale

La fonction `YMultiCellWeighScale.FindMultiCellWeighScale` permet de retrouver un capteur de poids multi-cellules 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-MaxiBridge-C avec le numéros de série `WMBRDG1C-123456` que vous auriez appelé `"MonModule"` et dont vous auriez nommé la fonction `multiCellWeighScale` `"MaFonction"`, les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("WMBRDG1C-123456.multiCellWeighScale")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
```

³ Pensez à changer le filtre de la fenêtre de sélection de fichiers, sinon la DLL n'apparaîtra pas

```

("WMBRDG1C-123456.MaFonction")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale
("MonModule.multiCellWeighScale")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MonModule.MaFonction")
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MaFonction")

```

`YMultiCellWeighScale.FindMultiCellWeighScale` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur de poids multi-cellules.

isOnline

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

set_excitation

La méthode `set_excitation()` de l'objet renvoyée par `yFindMultiCellWeighScale` permet d'activer sur le Yocto-MaxiBridge-C la source de tension permettant d'effectuer la mesure ratiométrique (mise sous tension de la cellule de charge et de force). L'utilisation d'une excitation alternative (AC) donne les meilleurs résultats pour une cellule de poids standard.

tare

La méthode `tare()` de l'objet renvoyée par `yFindMultiCellWeighScale` permet de calibrer la position au repos de la balance. Elle doit être appelée lorsque la balance n'est pas en charge.

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyée par `yFindMultiCellWeighScale` permet d'obtenir la valeur actuelle mesurée par le Yocto-MaxiBridge-C, calculée comme la somme de toutes les entrées actives du module. La valeur de poids dépend bien entendu de la configuration des paramètres des cellules de poids, qui devrait être faite précédemment pour obtenir une mesure correcte.

Un exemple réel

Lancez Microsoft VisualBasic et ouvrez le projet exemple correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-MaxiBridge-C** 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 sensor As YMultiCellWeighScale

        If argv.Length < 2 Then Usage()

        target = argv(1)
        REM Setup the API to use local USB devices
        If (YAPI.RegisterHub("usb", errmsg) <> YAPI_SUCCESS) Then
            Console.WriteLine("RegisterHub error: " + errmsg)
        End If
    End Sub
End Module

```

```

If target = "any" Then
    sensor = YMultiCellWeighScale.FirstMultiCellWeighScale()
    If sensor Is Nothing Then
        Console.WriteLine("No module connected (check USB cable) ")
    End If
    Console.WriteLine("using:" + sensor.get_module().get_serialNumber())
Else
    sensor = YMultiCellWeighScale.FindMultiCellWeighScale(target + ".multiCellWeighScale"
)
End If

If sensor.isOnline() Then
    REM On startup, enable excitation and tare weigh scale
    Console.WriteLine("Resetting tare weight...")
    sensor.set_excitation(YMultiCellWeighScale.EXCITATION_AC)
    YAPI.Sleep(3000, errmsg)
    sensor.tare()
End If

While sensor.isOnline()
    Console.WriteLine("Channel 1: " + Str(sensor.get_currentValue()) + sensor.get_unit())
    Console.WriteLine("    (press Ctrl-C to exit)")
    YAPI.Sleep(1000, errmsg)
End While
YAPI.FreeAPI()
Console.WriteLine("Module not connected (check identification and USB cable)")
End Sub

End Module

```

17.4. Contrôle de la partie module

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

```

Imports System.IO
Imports System.Environment

Module Module1

    Sub usage()
        Console.WriteLine("usage: demo <serial or logical name> [ON/OFF]")
    End Sub

    Sub Main()
        Dim argv() As String = System.Environment.GetCommandLineArgs()
        Dim errmsg As String = ""
        Dim m As ymodule

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

        If argv.Length < 2 Then usage()

        m = YModule.FindModule(argv(1)) REM use serial or logical name
        If (m.isOnline()) Then
            If argv.Length > 2 Then
                If argv(2) = "ON" Then m.set_beacon(Y_BEACON_ON)
                If argv(2) = "OFF" Then m.set_beacon(Y_BEACON_OFF)
            End If
            Console.WriteLine("serial:      " + m.get_serialNumber())
            Console.WriteLine("logical name: " + m.get_logicalName())
            Console.WriteLine("luminosity:   " + Str(m.get_luminosity()))
            Console.WriteLine("beacon:      ")
            If (m.get_beacon() = Y_BEACON_ON) Then
                Console.WriteLine("ON")
            End If
        End If
    End Sub
End Module

```

```

Else
    Console.WriteLine("OFF")
End If
Console.WriteLine("upTime:      " + Str(m.get_upTime() / 1000) + " sec")
Console.WriteLine("USB current:  " + Str(m.get_usbCurrent()) + " mA")
Console.WriteLine("Logs:")
Console.WriteLine(m.get_lastLogs())
Else
    Console.WriteLine(argv(1) + " not connected (check identification and USB cable)")
End If
YAPI.FreeAPI()
End Sub

End Module

```

Chaque propriété xxx du module peut être lue grâce à une méthode du type `get_xxxx()`, et les propriétés qui se sont pas en lecture seule peuvent être modifiées à l'aide de la méthode `set_xxx()`. Pour plus de détails concernant ces fonctions utilisées, reportez-vous 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 YAPI.RegisterHub("usb", errmsg) <> YAPI.SUCCESS Then
        Console.WriteLine("RegisterHub error: " + errmsg)
    End
    End If

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

End Sub

End Module

```

Attention, le nombre de cycles d'écriture de la mémoire non volatile du module est limité. Passé cette limite plus rien ne garantit que la sauvegarde des réglages se passera correctement. Cette limite, liée à la technologie employée par le micro-processeur du module se situe aux 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 modules connectés

```
Module Module1

Sub Main()
    Dim M As ymodule
    Dim errmsg As String = ""

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

    Console.WriteLine("Device list")
    M = YModule.FirstModule()
    While M IsNot Nothing
        Console.WriteLine(M.get_serialNumber() + " (" + M.get_productName() + ")")
        M = M.nextModule()
    End While
    YAPI.FreeAPI()
End Sub

End Module
```

17.5. Gestion des erreurs

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

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

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

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

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

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

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

18. Utilisation du Yocto-MaxiBridge-C en Delphi / Lazarus

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.

Lazarus² est un IDE gratuit basé sur Free-Pascal qui n'a pas grand chose à envier à Delphi. Il a aussi l'avantage d'exister pour Windows et Linux. La librairie Yoctopuce pour Delphi est compatible avec Lazarus tant sous Windows que Linux.

Les librairies pour Delphi / Lazarus 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 Delphi / Lazarus³.

18.1. Préparation

Connectez-vous sur le site de Yoctopuce et téléchargez la la librairie Yoctopuce pour Delphi⁴. Décompressez le tout dans le répertoire de votre choix.

- Avec Delphi ajoutez le sous-répertoire *sources* de l'archive dans la liste des répertoires des librairies de Delphi⁵.
- Avec Lazarus, ouvrez les options de votre projet et ajoutez le répertoire *sources* dans le champs "other unit files"⁶.

Windows

Sous Windows, la librairie Delphi / Lazarus utilise deux DLL: *yapi.dll* pour exécutables 32bits et *yapi64.dll* pour les exécutable 64bits. Toutes les applications que vous créez avec Delphi ou Lazarus devront avoir accès à ces DLL. Le plus simple est de faire en sorte qu'elles soient présentes dans le même répertoire que l'exécutable de votre application. Vous trouverez ces DLL dans le répertoire *sources/dll*.

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 www.lazarus-ide.org

³ Les librairies Delphi sont régulièrement testées avec Delphi 5 et Delphi XE2 et la dernière version de Lazarus

⁴ www.yoctopuce.com/FR/libraries.php

⁵ Utilisez le menu **outils / options d'environnement**

⁶ Utilisez le menu **Project / Project options/ Compiler options / Paths**

Linux

Sous Linux, la librairie Delphi / Lazarus utilise les librairies suivantes:

- libyapi-i386.so sur les systèmes Intel 32 bits
- libyapi-amd64.so sur les systèmes Intel 64 bits
- libyapi-armhf.so sur les systèmes ARM 32 bits
- libyapi-aarch64.so sur les systèmes ARM 64 bits

Vous trouverez ces fichiers lib dans le répertoire *sources/dll*. Vous devez faire en sorte que :

- Lazarus soit capable de localiser le bon fichier .so à la compilation
- L'exécutable soit capable de le localiser l'exécution

La solution la plus simple pour remplir ces conditions consiste à copier ces quatre fichiers dans le répertoire */usr/lib*. Une autre solution consiste à les copier dans le même répertoire que votre code source et à ajuster votre variable d'environnement *LD_LIBRARY_PATH* en conséquence.

A propos des exemples

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 fenêtrées.

Notez que la plupart de ces exemples utilisent des paramètres passés sur la ligne de commande⁷.

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.

18.2. Contrôle de la fonction MultiCellWeighScale

Il suffit de quelques lignes de code pour piloter un Yocto-MaxiBridge-C. Voici le squelette d'un fragment de code Delphi qui utilise la fonction MultiCellWeighScale.

```
uses yocto_api, yocto_multicellweighscale;

var errmsg: string;
    multicellweighscale: TYMultiCellWeighScale;

[...]
// On active la détection des modules sur USB
yRegisterHub('usb',errmsg)
[...]

// On récupère l'objet permettant d'interagir avec le module
multicellweighscale = yFindMultiCellWeighScale("WMBRDG1C-123456.multiCellWeighScale")

// Pour gérer le hot-plug, on vérifie que le module est là
if multicellweighscale.isOnline() then
begin
    // use multicellweighscale.get_currentValue()
    [...]
end;
[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api et yocto_multicellweighscale

Ces deux unités permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. *yocto_api* doit toujours être utilisé, *yocto_multicellweighscale* est nécessaire pour gérer les modules contenant un capteur de poids multi-cellules, comme le Yocto-MaxiBridge-C.

⁷ voir <http://www.yoctopuce.com/FR/article/a-propos-des-programmes-d-exemples>

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.

yFindMultiCellWeighScale

La fonction `yFindMultiCellWeighScale` permet de retrouver un capteur de poids multi-cellules 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-MaxiBridge-C avec le numéros de série *WMBRDG1C-123456* que vous auriez appelé *"MonModule"* et dont vous auriez nommé la fonction *multiCellWeighScale* *"MaFonction"*, les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
multicellweighscale := yFindMultiCellWeighScale("WMBRDG1C-123456.multiCellWeighScale");
multicellweighscale := yFindMultiCellWeighScale("WMBRDG1C-123456.MaFonction");
multicellweighscale := yFindMultiCellWeighScale("MonModule.multiCellWeighScale");
multicellweighscale := yFindMultiCellWeighScale("MonModule.MaFonction");
multicellweighscale := yFindMultiCellWeighScale("MaFonction");
```

`yFindMultiCellWeighScale` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur de poids multi-cellules.

isOnline

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

set_excitation

La méthode `set_excitation()` de l'objet renvoyée par `yFindMultiCellWeighScale` permet d'activer sur le Yocto-MaxiBridge-C la source de tension permettant d'effectuer la mesure ratiométrique (mise sous tension de la cellule de charge et de force). L'utilisation d'une excitation alternative (AC) donne les meilleurs résultats pour une cellule de poids standard.

tare

La méthode `tare()` de l'objet renvoyée par `yFindMultiCellWeighScale` permet de calibrer la position au repos de la balance. Elle doit être appelée lorsque la balance n'est pas en charge.

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyée par `yFindMultiCellWeighScale` permet d'obtenir la valeur actuelle mesurée par le Yocto-MaxiBridge-C, calculée comme la somme de toutes les entrées actives du module. La valeur de poids dépend bien entendu de la configuration des paramètres des cellules de poids, qui devrait être faite précédemment pour obtenir une mesure correcte.

Un exemple réel

Lancez votre environnement Delphi, copiez la DLL `yapi.dll` dans un répertoire et créez une nouvelle application console dans ce même répertoire, et copiez-coller le code ci dessous.

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

```
program helloworld;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  {$IFDEF UNIX}
  windows,
  {$ENDIF UNIX}
```

```

yocto_api,
yocto_multicellweighscale;

Procedure Usage();
var
  exe : string;

begin
  exe:= ExtractFileName(paramstr(0));
  WriteLn(exe+ ' <serial_number>');
  WriteLn(exe+ ' <logical_name>');
  WriteLn(exe+ ' any');
  sleep(3000);
  halt;
End;

var
  sensor : TYMultiCellWeighScale;
  errmsg : string;
  sensorUnit : string;

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);
    sleep(3000);
    halt;
  end;

  if paramstr(1)='any' then
  begin
    sensor := yFirstMultiCellWeighScale();
    if sensor=nil then
    begin
      writeln('No module connected (check USB cable)');
      sleep(3000);
      halt;
    end
  end
  else
    sensor:= yFindMultiCellWeighScale(paramstr(1)+'.multiCellWeighScale');

  if sensor.isOnline() then
  begin
    // On startup, enable excitation and tare weigh scale
    Writeln('Resetting tare weight...');
    sensor.set_excitation(Y_EXCITATION_AC);
    ySleep(3000,errmsg);
    sensor.tare();
    sensorUnit := sensor.get_unit();
  end;

  // Show measured weight continuously
  while sensor.isOnline() do
  begin
    Write('Weight : '+FloatToStr(sensor.get_currentValue())+sensorUnit);
    Writeln('      (press Ctrl-C to exit)');
    ySleep(1000,errmsg);
  end;
  yFreeAPI();
  Writeln('Module not connected (check identification and USB cable)');
end.

```

18.3. Contrôle de la partie module

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

```

program modulecontrol;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  yocto_api;

const
  serial = 'WMBRDG1C-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 = 'WMBRDG1C-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.

```

18.4. Gestion des erreurs

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

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

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

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

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

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

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

19. Utilisation du Yocto-MaxiBridge-C 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.

19.1. 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.

19.2. Installation

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

19.3. Utilisation l'API Yoctopuce dans un projet Visual Studio

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

Une fenêtre de sélection de fichiers apparaît: sélectionnez tous les fichiers du répertoire `Sources` de la librairie.

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

Le fichier `Package.appxmanifest`

Par défaut, une application Universal Windows n'a pas le droit 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-MaxiBridge-C voici ce qu'il faut ajouter dans le n u d "Capabilities":

```
<DeviceCapability Name="humaninterfacedevice">
  <!-- Yocto-MaxiBridge-C -->
  <Device Id="vidpid:24e0 00C2">
    <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.

19.4. Contrôle de la fonction MultiCellWeighScale

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

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

// On récupère l'objet permettant d'interagir avec le module
YMultiCellWeighScale multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale(
  "WMBRDG1C-123456.multiCellWeighScale");

// Pour gérer le hot-plug, on vérifie que le module est là
if (await multicellweighscale.IsOnline())
{
    // Use multicellweighscale.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.

YMultiCellWeighScale.FindMultiCellWeighScale

La fonction `YMultiCellWeighScale.FindMultiCellWeighScale` permet de retrouver un capteur de poids multi-cellules 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-MaxiBridge-C avec le numéros de série *WMBRDG1C-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *multiCellWeighScale* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale(
  "WMBRDG1C-123456.multiCellWeighScale");
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale(
  "WMBRDG1C-123456.MaFonction");
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale(
  "MonModule.multiCellWeighScale");
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MonModule.MaFonction");
multicellweighscale = YMultiCellWeighScale.FindMultiCellWeighScale("MaFonction");
```

`YMultiCellWeighScale.FindMultiCellWeighScale` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur de poids multi-cellules.

isOnline

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

set_excitation

La méthode `set_excitation()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'activer sur le Yocto-MaxiBridge-C la source de tension permettant d'effectuer la mesure ratiométrique (mise sous tension de la cellule de charge et de force). L'utilisation d'une excitation alternative (AC) donne les meilleurs résultats pour une cellule de poids standard.

tare

La méthode `tare()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet de calibrer la position au repos de la balance. Elle doit être appelée lorsque la balance n'est pas en charge.

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'obtenir la valeur actuelle mesurée par le Yocto-MaxiBridge-C, calculée comme la somme de toutes les entrées actives du module. La valeur de poids dépend bien entendu de la configuration des paramètres des cellules de poids, qui devrait être faite précédemment pour obtenir une mesure correcte.

19.5. Un exemple concret

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

Le projets Visual Studio contient de nombreux fichiers dont la plupart ne sont pas liés à l'utilisation de la librairie Yoctopuce. Pour simplifier la lecture du code nous avons regroupé tout le code qui utilise la librairie dans la classe `Demo` qui se trouve dans le fichier `demo.cs`. Les propriétés de cette classe correspondent aux 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);

                YMultiCellWeighScale sensor;

                if (Target.ToLower() == "any") {
                    sensor = YMultiCellWeighScale.FirstMultiCellWeighScale();
                }
            }
        }
    }
}
```

```

        if (sensor == null) {
            WriteLine("No module connected (check USB cable) ");
            return -1;
        }
        YModule ymod = await sensor.get_module();
        WriteLine("Using: " + await ymod.get_serialNumber());
    } else {
        sensor = YMultiCellWeighScale.FindMultiCellWeighScale(Target +
            ".multiCellWeighScale");
    }

    if (await sensor.isOnline()) {
        string unit = "";
        // On startup, enable excitation and tare weigh scale
        WriteLine("Resetting tare weight...");
        await sensor.set_excitation(YMultiCellWeighScale.EXCITATION_AC);
        await YAPI.Sleep(3000);
        await sensor.tare();
        unit = await sensor.get_unit();

        // Show measured weight continuously
        while (await sensor.isOnline()) {
            Write("Weight : " + await sensor.get_currentValue() + unit);
            await YAPI.Sleep(1000);
        }
    }
    WriteLine("Module not connected (check identification and USB cable)");
} catch (YAPI_Exception ex) {
    WriteLine("error: " + ex.Message);
}

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

```

19.6. Contrôle de la partie module

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

```

using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.UI.Xaml.Controls;
using com.yoctopuce.YoctoAPI;

namespace Demo
{
    public class Demo : DemoBase
    {
        public string HubURL { get; set; }
        public string Target { get; set; }
        public bool Beacon { get; set; }

        public override async Task<int> Run()
        {
            YModule m;
            string errmsg = "";

            if (await YAPI.RegisterHub(HubURL) != YAPI.SUCCESS) {
                WriteLine("RegisterHub error: " + errmsg);
                return -1;
            }
            m = YModule.FindModule(Target + ".module"); // use serial or logical name
            if (await m.isOnline()) {
                if (Beacon) {
                    await m.set_beacon(YModule.BEACON_ON);
                } else {
                    await m.set_beacon(YModule.BEACON_OFF);
                }
            }
        }
    }
}

```

```

        WriteLine("serial: " + await m.get_serialNumber());
        WriteLine("logical name: " + await m.get_logicalName());
        WriteLine("luminosity: " + await m.get_luminosity());
        Write("beacon: ");
        if (await m.get_beacon() == YModule.BEACON_ON)
            WriteLine("ON");
        else
            WriteLine("OFF");
        WriteLine("upTime: " + (await m.get_upTime() / 1000) + " sec");
        WriteLine("USB current: " + await m.get_usbCurrent() + " mA");
        WriteLine("Logs:\r\n" + await m.get_lastLogs());
    } else {
        WriteLine(Target + " not connected on" + HubURL +
            "(check identification and USB cable)");
    }
    await 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);
    }
    await 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);
            }
            await YAPI.FreeAPI();
            return 0;
        }
    }
}

```

19.7. Gestion des erreurs

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

La manière la plus simple de contourner le problème est celle que nous avons employé pour les petits exemples précédents de ce chapitre: avant d'accéder à un module, on vérifie qu'il est en ligne avec la méthode `isOnline()` et on suppose ensuite qu'il va y rester pendant la fraction de

seconde nécessaire à exécuter les lignes de code suivantes. Ce n'est pas parfait, mais ça peut suffire dans certains cas. Il faut toutefois être conscient qu'on ne peut pas totalement exclure une erreur se produisant après le `isOnline()`, qui pourrait faire planter le programme.

Dans la librairie Universal Windows Platform, le traitement d'erreur est implémenté au moyen d'exceptions. Vous devrez donc intercepter et traiter correctement ces exceptions si vous souhaitez avoir un projet fiable qui ne crashera pas 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);
}
```


20. Utilisation du Yocto-MaxiBridge-C en Objective-C

Objective-C est le langage de prédilection pour programmer sous macOS, en raison de son intégration avec le générateur d'interfaces Cocoa. Yoctopuce supporte les versions de XCode supportées par Apple. 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.

20.1. Contrôle de la fonction MultiCellWeighScale

Il suffit de quelques lignes de code pour piloter un Yocto-MaxiBridge-C. Voici le squelette d'un fragment de code Objective-C qui utilise la fonction MultiCellWeighScale.

```
#import "yocto_api.h"
#import "yocto_multicellweighscale.h"

...
NSError *error;
[YAPI RegisterHub:@"usb": &error]
...
// On récupère l'objet représentant le module (ici connecté en local sur USB)
multicellweighscale = [YMultiCellWeighScale
FindMultiCellWeighScale:@"WMBRDG1C-123456.multiCellWeighScale"];
```

¹ www.yoctopuce.com/FR/libraries.php

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

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.h et yocto_multicellweighscale.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_multicellweighscale.h` est nécessaire pour gérer les modules contenant un capteur de poids multi-cellules, comme le Yocto-MaxiBridge-C.

[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.

[MultiCellWeighScale FindMultiCellWeighScale]

La fonction `[MultiCellWeighScale FindMultiCellWeighScale]`, permet de retrouver un capteur de poids multi-cellules 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-MaxiBridge-C avec le numéros de série *WMBRDG1C-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *multiCellWeighScale* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YMultiCellWeighScale *multicellweighscale = [YMultiCellWeighScale
FindMultiCellWeighScale:@"WMBRDG1C-123456.multiCellWeighScale"];
YMultiCellWeighScale *multicellweighscale = [YMultiCellWeighScale
FindMultiCellWeighScale:@"WMBRDG1C-123456.MaFonction"];
YMultiCellWeighScale *multicellweighscale = [YMultiCellWeighScale
FindMultiCellWeighScale:@"MonModule.multiCellWeighScale"];
YMultiCellWeighScale *multicellweighscale = [YMultiCellWeighScale
FindMultiCellWeighScale:@"MonModule.MaFonction"];
YMultiCellWeighScale *multicellweighscale = [YMultiCellWeighScale
FindMultiCellWeighScale:@"MaFonction"];
```

`[YMultiCellWeighScale FindMultiCellWeighScale]` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur de poids multi-cellules.

isOnline

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

set_excitation

La méthode `set_excitation()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'activer sur le Yocto-MaxiBridge-C la source de tension permettant d'effectuer la mesure ratiométrique (mise sous tension de la cellule de charge et de force). L'utilisation d'une excitation alternative (AC) donne les meilleurs résultats pour une cellule de poids standard.

tare

La méthode `tare()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet de calibrer la position au repos de la balance. Elle doit être appelée lorsque la balance n'est pas en charge.

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyée par `YMultiCellWeighScale.FindMultiCellWeighScale` permet d'obtenir la valeur actuelle mesurée par le Yocto-MaxiBridge-C, calculée comme la somme de toutes les entrées actives du module. La valeur de poids dépend bien entendu de la configuration des paramètres des cellules de poids, qui devrait être faite précédemment pour obtenir une mesure correcte.

Un exemple réel

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

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

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

static void usage(void)
{
    NSLog(@"usage: demo <serial_number> ");
    NSLog(@"          demo <logical_name>");
    NSLog(@"          demo any          (use any discovered device)");
    exit(1);
}

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

    if (argc < 2) {
        usage();
    }

    @autoreleasepool {
        NSString *target = [NSString stringWithUTF8String:argv[1]];

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

        if ([target isEqualToString:@"any"]) {
            // retrieve any generic sensor
            sensor = [YMultiCellWeighScale FirstMultiCellWeighScale];
            if (sensor == NULL) {
                NSLog(@"No module connected (check USB cable)");
                return 1;
            }
        } else {
            sensor = [YMultiCellWeighScale FindMultiCellWeighScale:target];
        }

        // we need to retrieve both DC and AC current from the device.
        if (![sensor isOnline]) {
            NSLog(@"No module connected (check USB cable)");
            return 1;
        }

        // On startup, enable excitation and tare weigh scale
        NSLog(@"Resetting tare weight...");
        [sensor set_excitation: Y_EXCITATION_AC];
        [YAPI Sleep:3000:NULL];
        [sensor tare];
    }
}
```

```

while([sensor isOnline]) {
    NSLog(@"Weight : %f %@", [sensor currentValue], [sensor get_unit]);
    NSLog(@" (press Ctrl-C to exit)");
    [YAPI Sleep:1000:NULL];
}
[YAPI FreeAPI];
NSLog(@"Module not connected (check identification and USB cable)");
}
return 0;
}

```

20.2. Contrôle de la partie module

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

```

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

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

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

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

```

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

Modifications des réglages du module

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

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

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

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

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

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

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

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

Attention, le nombre de cycles d'écriture de la mémoire non volatile du module est limité. Passé cette limite plus rien ne garantit que la sauvegarde des réglages se passera correctement. Cette limite, liée à la technologie employée par le micro-processeur du module se situe aux 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;
}
```

20.3. Gestion des erreurs

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

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

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

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

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

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

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

21. Utilisation avec des langages non supportés

Les modules Yoctopuce peuvent être contrôlés depuis la plupart des langages de programmation courants. De nouveaux langages sont ajoutés régulièrement en fonction de l'intérêt exprimé par les utilisateurs de produits Yoctopuce. Cependant, certains langages ne sont pas et ne seront jamais supportés par Yoctopuce, les raisons peuvent être diverses: compilateurs plus disponibles, environnements inadaptés, etc...

Il existe cependant des méthodes alternatives pour accéder à des modules Yoctopuce depuis un langage de programmation non supporté.

21.1. Utilisation en ligne de commande

Le moyen le plus simple pour contrôler des modules Yoctopuce depuis un langage non supporté consiste à utiliser l'API en ligne de commande à travers des appels système. L'API en ligne de commande se présente en effet sous la forme d'un ensemble de petits exécutables qu'il est facile d'appeler et dont la sortie est facile à analyser. La plupart des langages de programmation permettant d'effectuer des appels système, cela permet de résoudre le problème en quelques lignes.

Cependant, si l'API en ligne de commande est la solution la plus facile, ce n'est pas la plus rapide ni la plus efficace. A chaque appel, l'exécutable devra initialiser sa propre API et faire l'inventaire des modules USB connectés. Il faut compter environ une seconde par appel.

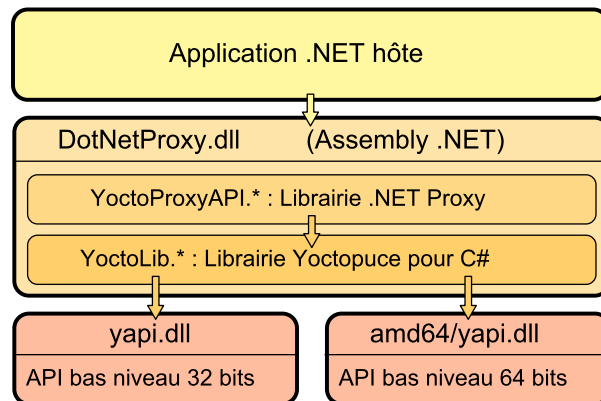
21.2. Assembly .NET

Un Assembly .NET permet de partager un ensemble de classes précompilées pour offrir un service, en annonçant des points d'entrées qui peuvent être utilisés par des applications tierces. Dans notre cas, c'est toute la librairie Yoctopuce qui est disponible dans l'Assembly .NET, de sorte à pouvoir être utilisée dans n'importe quel environnement qui supporte le chargement dynamique d'Assembly .NET.

La librairie Yoctopuce sous forme d'Assembly .NET ne contient pas uniquement la librairie Yoctopuce standard pour C#, car cela n'aurait pas permis une utilisation optimale dans tous les environnements. En effet, on ne peut pas attendre forcément des applications hôtes d'offrir un système de threads ou de callbacks, pourtant très utiles pour la gestion du plug-and-play et des capteurs à taux de rafraîchissements élevé. De même, on ne peut pas attendre des applications externes un comportement transparent dans le cas où un appel de fonction dans l'Assembly cause un délai en raison de communication réseau.

Nous y avons donc ajouté une surcouche, appelée librairie *.NET Proxy*. Cette surcouche offre une interface très similaire à la librairie standard mais un peu simplifiée, car elle gère en interne tous les

mécanismes de callbacks. A la place, cette librairie offre des objets miroirs, appelés *Proxys*, qui publient par le biais de *Propriétés* les principaux attributs des fonctions Yoctopuce tels que la mesure courante, les paramètres de configuration, l'état, etc.



Architecture de l'Assembly .NET

Les propriétés des objets *Proxys* sont automatiquement mises à jour en tâche de fond par le mécanisme de callbacks, sans que l'application hôte n'ait à s'en soucier. Celle-ci peut donc à tout moment et sans aucun risque de latence afficher la valeur de toutes les propriétés des objets *Proxys* Yoctopuce.

Notez bien que la librairie de communication de bas niveau `yapi.dll` n'est **pas** incluse dans l'Assembly .NET. Il faut donc bien penser à la garder toujours avec `DotNetProxyLibrary.dll`. La version 32 bits doit être dans le même répertoire que `DotNetProxyLibrary.dll`, tandis que la version 64 bits doit être dans un sous-répertoire nommé `amd64`.

Exemple d'utilisation avec MATLAB

Voici comment charger notre Assembly .NET Proxy dans MATLAB et lire la valeur du premier capteur branché par USB trouvé sur la machine :

```

NET.addAssembly("C:/Yoctopuce/DotNetProxyLibrary.dll");
import YoctoProxyAPI.*

errmsg = YAPIProxy.RegisterHub("usb");
sensor = YSensorProxy.FindSensor("");
measure = sprintf('%0.3f %s', sensor.CurrentValue, sensor.Unit);
  
```

Exemple d'utilisation en PowerShell

Les commandes en PowerShell sont un peu plus étranges, mais on reconnaît le même schéma :

```

Add-Type -Path "C:/Yoctopuce/DotNetProxyLibrary.dll"

$errmsg = [YoctoProxyAPI.YAPIProxy]::RegisterHub("usb")
$sensor = [YoctoProxyAPI.YSensorProxy]::FindSensor("")
$measure = "{0:n3} {1}" -f $sensor.CurrentValue, $sensor.Unit
  
```

Particularités de la librairie .NET Proxy

Par rapport aux librairies Yoctopuce classiques, on notera en particulier les différences suivantes.

Pas de méthode FirstModule/nextModule

Pour obtenir un objet se référant au premier module trouvé, on appelle un `YModuleProxy.FindModule("")`. Si aucun module n'est connecté, cette méthode retournera un objet avec la propriété `module.IsOnline` à `False`. Dès le branchement d'un module, la propriété passera à `True` et l'identifiant matériel du module sera mis à jour.

Pour énumérer les modules, on peut appeler la méthode `module.GetSimilarFunctions()` qui retourne un tableau de chaînes de caractères contenant les identifiants de tous les module trouvés.

Pas de fonctions de callback

Les fonctions de callback sont implémentées en interne et mettent à jour les propriétés des objets. Vous pouvez donc simplement faire du polling sur les propriétés, sans pénalité significative de performance. Prenez garde au fait que si vous utilisez l'une des méthodes qui désactive les callbacks, le rafraîchissement automatique des propriétés des objets en sera altéré.

Une nouvelle méthode `YAPIProxy.GetLog` permet de récupérer les logs de diagnostics de bas niveau sans recourir à l'utilisation de callbacks.

Types énumérés

Pour maximiser la compatibilité avec les applications hôte, la librairie .NET Proxy n'utilise pas de véritables types énumérés .NET, mais des simples entiers. Pour chaque type énuméré, la librairie publie des constantes entières nommées correspondant aux valeurs possibles. Contrairement aux librairies Yoctopuce classiques, les valeurs utiles commencent toujours à 1, la valeur 0 étant réservée pour signifier une valeur invalide, par exemple lorsque le module est débranché.

Valeurs numériques invalides

Pour toutes les grandeurs numériques, plutôt qu'une constante arbitraire, la valeur invalide retournée en cas d'erreur est *NaN*. Il faut donc utiliser la fonction `isNaN()` pour détecter cette valeur.

Utilisation de l'Assembly .NET sans la librairie Proxy

Si pour une raison ou une autre vous ne désirez pas utiliser la librairie Proxy, et que votre environnement le permet, vous pouvez utiliser l'API C# standard puisqu'elle se trouve dans l'Assembly, sous le namespace `YoctoLib`. Attention toutefois à ne pas mélanger les deux utilisations: soit vous passez par la librairie Proxy, soit vous utilisez directement la version `YoctoLib`, mais pas les deux !

Compatibilité

Pour que la librairie .NET Proxy fonctionne correctement avec vos modules Yoctopuce, ces derniers doivent avoir au moins le firmware 37120.

Afin d'être compatible avec un maximum de version de Windows, y compris Windows XP, la librairie *DotNetProxyLibrary.dll* est compilée en .NET 3.5, qui est disponible par défaut sur toutes les versions de Windows depuis XP. A ce jour nous n'avons pas trouvé d'environnement hormis Windows qui supporte le chargement d'Assemblies, donc seules les dll de bas niveau pour Windows sont distribuées avec l'Assembly.

21.3. Virtual Hub et HTTP GET

Le *Virtual Hub* est disponible pour presque toutes les plateformes actuelles, il sert généralement de passerelle pour permettre l'accès aux modules Yoctopuce depuis des langages qui interdisent l'accès direct aux couches matérielles d'un ordinateur (Javascript, PHP, Java...).

Il se trouve que le *Virtual Hub* est en fait un petit serveur Web qui est capable de router des requêtes HTTP vers les modules Yoctopuce. Ce qui signifie que si vous pouvez faire une requête HTTP depuis votre langage de programmation, vous pouvez contrôler des modules Yoctopuce, même si ce langage n'est pas officiellement supporté.

Interface REST

A bas niveau, les modules sont pilotés à l'aide d'une API REST. Ainsi pour contrôler un module, il suffit de faire les requêtes HTTP appropriées sur le *Virtual Hub*. Par défaut le port HTTP du *Virtual Hub* est 4444.

Un des gros avantages de cette technique est que les tests préliminaires sont très faciles à mettre en 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-MaxiBridge-C avec le numéro de de série *WMBRDG1C-12345* et le nom logique *monModule*. l'URL suivante permettra de connaître l'état du module.

```
http://127.0.0.1:4444/bySerial/WMBRDG1C-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/WMBRDG1C-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/WMBRDG1C-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 *multiCellWeighScale*, il suffit de construire l'URL suivante.

```
http://127.0.0.1:4444/bySerial/WMBRDG1C-12345/api/multiCellWeighScale.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/WMBRDG1C-12345/api/multiCellWeighScale/logicalName
```

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

```
http://127.0.0.1:4444/bySerial/WMBRDG1C-12345/api/multiCellWeighScale?
logicalName=maFonction
```

Vous trouverez la liste des attributs disponibles pour votre Yocto-MaxiBridge-C 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/WMBRDG1C-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/WMBRDG1C-12345/dataLogger.json?
id=multiCellWeighScale&utc=1389801080
```

21.4. 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 (32 bits)
libyapi-armhf.so	Linux ARM HL (32 bits)
libyapi-aarch64.so	Linux ARM (64 bits)
libyapi-i386.so	Linux Intel (32 bits)
yapi64.dll	Windows (64 bits)
yapi.dll	Windows (32 bits)

Ces 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 bufsize, 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 = 'WMBRDG1C-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;
```

```
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 *neededsized, 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. *neededsized* contiendra au retour la taille nécessaire pour stocker tous les handles. Cela permet d'en déduire le nombre de module connectés, ou si le buffer passé en entrée est trop petit. Le paramètre *errmsg* devra pointer sur un buffer de 255 caractères pour récupérer un éventuel message d'erreur. Ce pointeur peut aussi être à *null*. Cette fonction retourne un entier négatif en cas d'erreur, ou zéro dans le cas contraire.

La fonction *yapiGetDeviceInfo* permet de récupérer les informations relatives à un module à partir de son handle. *devdesc* est un entier 32bit qui représente le module, et qui a été obtenu grâce à *yapiGetAllDevices*. *infos* pointe sur une structure de données dans laquelle sera stocké le résultat. Le format de cette structure est le suivant:

Nom	Type	Taille (bytes)	Description
vendorid	int	4	ID USB de Yoctopuce
deviceid	int	4	ID USB du module
devrelease	int	4	Version du module
nbinbterfaces	int	4	Nombre d'interfaces USB utilisée par le module
manufacturer	char[]	20	Yoctopuce (null terminé)
productname	char[]	28	Modèle (null terminé)
serial	char[]	20	Numéro de série (null terminé)
logicalname	char[]	20	Nom logique (null terminé)
firmware	char[]	22	Version du firmware (null terminé)
beacon	byte	1	Etat de la balise de localisation (0/1)

Le paramètre *errmsg* devra pointer sur un buffer de 255 caractères pour récupérer un éventuel message d'erreur.

Voici un programme d'exemple écrit en pascal qui utilise la DLL *yapi.dll* pour lister les modules connectés.

```
// device description structure
type yDeviceSt = packed record
  vendorid      : word;
  deviceid      : word;
  devrelease    : word;
  nbinbterfaces : word;
  manufacturer  : array [0..19] of ansichar;
  productname   : array [0..27] of ansichar;
  serial        : array [0..19] of ansichar;
  logicalname   : array [0..19] of ansichar;
  firmware      : array [0..21] of ansichar;
  beacon        : byte;
end;

// Dll function import
function yapiInitAPI(mode:integer;
  errmsg : pansichar):integer;cdecl;
  external 'yapi.dll' name 'yapiInitAPI';

function yapiUpdateDeviceList(force:integer;errmsg : pansichar):integer;cdecl;
  external 'yapi.dll' name 'yapiUpdateDeviceList';
```

```

function yapiGetAllDevices( buffer:pointer;
                           maxsize:integer;
                           var neededsize:integer;
                           errmsg : pansichar):integer; cdecl;
external 'yapi.dll' name 'yapiGetAllDevices';

function apiGetDeviceInfo(d:integer; var infos:yDeviceSt;
                          errmsg : pansichar):integer; cdecl;
external 'yapi.dll' name 'yapiGetDeviceInfo';

var
errmsgBuffer : array [0..256] of ansichar;
dataBuffer   : array [0..127] of integer; // max of 128 USB devices
errmsg,data   : pansichar;
neededsize,i  : integer;
devinfos      : yDeviceSt;

begin
    errmsg := @errmsgBuffer;

    // API initialisation
    if(yapiInitAPI(1,errmsg)<0) then
    begin
        writeln(errmsg);
        halt;
    end;

    // forces a device inventory
    if( yapiUpdateDeviceList(1,errmsg)<0) then
    begin
        writeln(errmsg);
        halt;
    end;

    // loads all device handles into dataBuffer
    if yapiGetAllDevices(@dataBuffer,sizeof(dataBuffer),neededsize,errmsg)<0 then
    begin
        writeln(errmsg);
        halt;
    end;

    // gets device info from each handle
    for i:=0 to neededsize div sizeof(integer)-1 do
    begin
        if (apiGetDeviceInfo(dataBuffer[i], devinfos, errmsg)<0) then
        begin
            writeln(errmsg);
            halt;
        end;
        writeln(pansichar(@devinfos.serial)+' ('+pansichar(@devinfos.productname)+' '));
    end;

end.

```

VB6 et yapi.dll

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

21.5. Port de la librairie haut niveau

Toutes les sources de l'API Yoctopuce étant fournies dans leur intégralité, vous pouvez parfaitement entreprendre le port complet de l'API dans le langage de votre choix. Sachez cependant qu'une grande partie du code source de l'API est généré automatiquement.

Ainsi, il n'est pas nécessaire de porter la totalité de l'API, il suffit de porter le fichier `yocto_api` et un de ceux correspondant à une fonctionnalité, par exemple `yocto_relay`. Moyennant un peu de travail supplémentaire, Yoctopuce sera alors en mesure de générer tous les autres fichiers. C'est pourquoi il est fortement recommandé de contacter le support Yoctopuce avant d'entreprendre le port de la librairie Yoctopuce dans un autre langage. Un travail collaboratif sera profitable aux deux parties.

22. Programmation avancée

Les chapitres précédents vous ont présenté dans chaque langage disponible les fonctions de programmation de base utilisables avec votre module Yocto-MaxiBridge-C. 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 librairies 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.

22.1. Programmation par événements

Les méthodes de gestion des modules Yoctopuce qui vous ont été présentées dans les chapitres précédents sont des fonctions de polling, qui consistent à demander en permanence à l'API si quelque chose a changé. Facile à appréhender, cette technique de programmation 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 capteur

L'API Yoctopuce fournit aussi un système de callback permettant d'être prévenu automatiquement de la valeur d'un capteur, 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 capteur de poids multi-cellules 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(YMultiCellWeighScale 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 YMultiCellWeighScale 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 MultiCellWeighScale donnée à l'aide de la méthode `registerValueCallback`. L'exemple suivant met en place un callback pour la première fonction MultiCellWeighScale disponible.

```
YMultiCellWeighScale f = YMultiCellWeighScale.FirstMultiCellWeighScale();
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(YMultiCellWeighScale 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 MultiCellWeighScale 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 `MultiCellWeighScale` disponible.

```
YMultiCellWeighScale f = YMultiCellWeighScale.FirstMultiCellWeighScale();
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 `YMultiCellWeighScale`. Ainsi, la même fonction callback pourra être utilisée avec toutes les sous-classes de `YSensor` (et en particulier avec `YMultiCellWeighScale`). 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 *Examples/Prog-EventBased*.

22.2. L'enregistreur de données

Votre Yocto-MaxiBridge-C 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.

Notez qu'il est inutile et contre-productive de programmer une fréquence d'enregistrement plus élevée que la fréquence native d'échantillonnage du capteur concerné.

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 connexion USB active pour fonctionner: ils commencent à fonctionner dès qu'ils sont alimentés. Le Yocto-MaxiBridge-C 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-MaxiBridge-C 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-MaxiBridge-C, 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-MaxiBridge-C 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-MaxiBridge-C 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-MaxiBridge-C 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-MaxiBridge-C 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-MaxiBridge-C 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-MaxiBridge-C 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-MaxiBridge-C 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-MaxiBridge-C "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-MaxiBridge-C peut dériver petit à petit (en principe jusqu'à 2%).

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

22.3. Calibration des senseurs

Votre module Yocto-MaxiBridge-C 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-MaxiBridge-C. 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-MaxiBridge-C, 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-MaxiBridge-C. D'autre part l'API supporte une calibration arbitraire externe, implémentée à l'aide de callbacks.

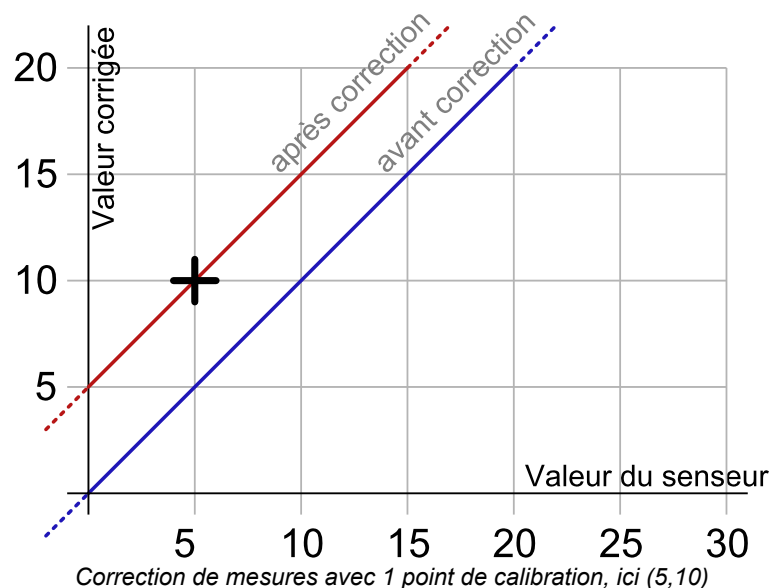
Interpolation linéaire 1 à 5 points

Ces transformations sont effectuées directement dans le Yocto-MaxiBridge-C 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 `multiCellWeighScale` retournera b .



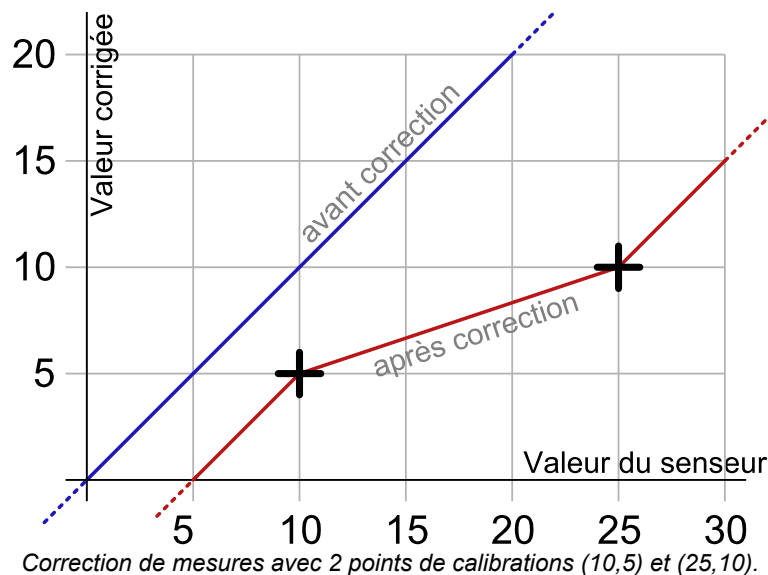
La mise en pratique est des plus simples: il suffit d'appeler la méthode `calibrateFromPoints()` de la fonction que l'on désire corriger. Le code suivant applique la correction illustrée sur le graphique ci-dessus à la première fonction `multiCellWeighScale` trouvée. Notez l'appel à la méthode `saveToFlash`

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};
YMultiCellWeighScale f = YMultiCellWeighScale.FirstMultiCellWeighScale();
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 multiCellWeighScale 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.



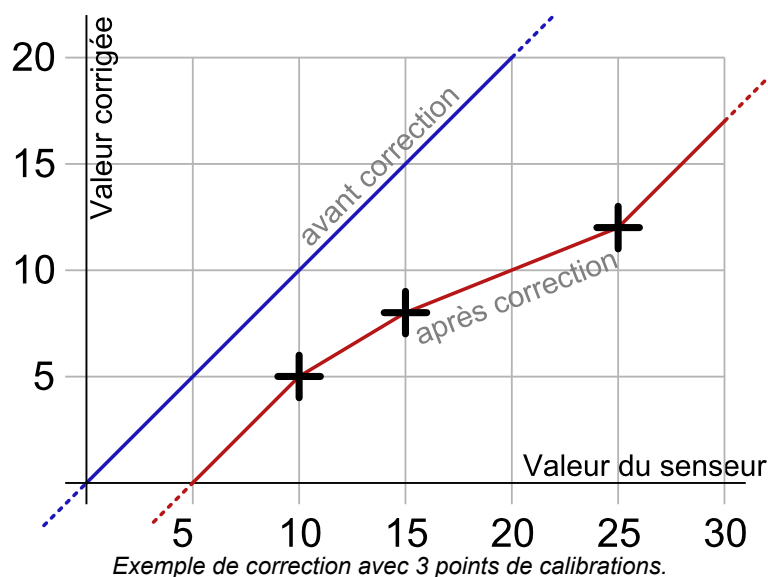
Le code permettant de programmer cette calibration est très similaire au code précédent

```
Double[] ValuesBefore = {10,25};
Double[] ValuesAfter = {5,10};
YMultiCellWeighScale f = YMultiCellWeighScale.FirstMultiCellWeighScale();
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 = {};
YMultiCellWeighScale f = YMultiCellWeighScale.FirstMultiCellWeighScale();
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.

23. Mise à jour du firmware

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

23.1. Le VirtualHub ou le YoctoHub

Il est possible de mettre à jour un module directement depuis l'interface web du 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 de fonctionnait plus, débranchez puis rebranchez le module en maintenant sur le Yocto-bouton appuyé. Le module va démarrer en mode "mise à jour" et sera listé en dessous des modules connectés.

23.2. La librairie ligne de commandes

Tous les outils en lignes de commandes ont la possibilité de mettre à jour les modules Yoctopuce grâce à la commande `downloadAndUpdate`. Le mécanisme de sélection des modules fonctionne comme pour une commande traditionnelle. La [cible] est le nom du module qui va être mis à jour. Vous pouvez aussi utiliser les alias "any" ou "all", ou encore une liste de noms, séparés par des virgules, sans espace.

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

L'exemple suivant met à jour tous les modules Yoctopuce connectés en USB.

```
C:\>YModule all downloadAndUpdate
ok: Yocto-PowerRelay RELAYHI1-266C8(rev=15430) is up to date.
ok: 0 / 0 hubs in 0.000000s.
ok: 0 / 0 shields in 0.000000s.
ok: 1 / 1 devices in 0.130000s 0.130000s per device.
ok: All devices are now up to date.
C:\>
```

23.3. L'application Android Yocto-Firmware

Il est possible de mettre à jour le firmware de vos modules depuis votre téléphone ou tablette Android avec l'application [Yocto-Firmware](#). Cette application liste tous les modules Yoctopuce

branchés en USB et vérifie si un firmware plus récent est disponible sur www.yoctopuce.com. Si un firmware plus récent est disponible, il est possible de mettre à jour le module. L'application se charge de télécharger et d'installer le nouveau firmware en préservant les paramètres du module.

Attention, pendant la mise à jour du firmware, le module redémarre plusieurs fois. Android interprète le reboot d'un périphérique USB comme une déconnexion et reconnexion du périphérique USB, et demande à nouveau l'autorisation d'utiliser le port USB. L'utilisateur est obligé de cliquer sur **OK** pour que la procédure de mise à jour se termine correctement.

23.4. La librairie de programmation

Si vous avez besoin d'intégrer la mise à jour de firmware dans votre application, les librairies proposent une API pour mettre à jour vos modules.

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");
}
...
```

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.**

23.5. Le mode "mise à jour"

Si vous désirez effacer tous les paramètres du module ou que votre module ne démarre plus correctement, il est possible d'installer un firmware depuis le mode "mise à jour".

Pour forcer le module à fonctionner dans le mode "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();
.....

```

24. Référence de l'API de haut niveau

Ce chapitre résume les fonctions de l'API de haut niveau pour commander votre Yocto-MaxiBridge-C. 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.

24.1. La classe YAPI

Fonctions générales

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

java	<code>import com.yoctopuce.YoctoAPI.YAPI;</code>
dnf	<code>import YoctoProxyAPI.YAPIProxy</code>
cp	<code>#include "yocto_api_proxy.h"</code>
cpp	<code>#include "yocto_api.h"</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
py	<code>from yocto_api import *</code>
php	<code>require_once('yocto_api.php');</code>
ts	<code>in HTML: import { YAPI, YErrorMsg, YModule, YSensor } from '../dist/esm/yocto_api_browser.js'; in Node.js: import { YAPI, YErrorMsg, YModule, YSensor } from 'yoctolib-cjs/yocto_api_nodejs.js';</code>
ppy	<code>from yoctolib.yocto_api import *</code>

Fonction globales

YAPI.AddTrustedCertificates(**certificate**)

Ajoute un certificat TLS/SSL à la liste des certificats de confiance.

cpp vb cs java py php ts ppy

YAPI.AddUdevRule(**force**)

Ajoute une règle UDEV qui autorise tous les utilisateurs à accéder aux modules Yoctopuce connectés sur les ports USB.

cpp vb cs java py php ts

YAPI.CheckLogicalName(**name**)

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

cpp vb cs java py php ts ppy

YAPI.ClearHTTPCallbackCacheDir(**removeFiles**)

Désactive le cache de callback HTTP.

java php

YAPI.DisableExceptions()

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

cpp vb cs py php ts

YAPI.DownloadHostCertificate(**url**, **mstimeout**)

Télécharge le certificat TLS/SSL du hub.

cpp vb cs java py php ts ppy

YAPI.EnableExceptions()

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

cpp vb cs py php ts

YAPI.EnableUSBHost(osContext)

Cette fonction est utilisée uniquement sous Android.

java

YAPI.FreeAPI()

Attends que toutes les communications en cours avec les modules Yoctopuce soient terminées puis libère les ressources utilisées par la librairie Yoctopuce.

cpp vb cs java py php ts dnp ppy

YAPI.GetAPIVersion()

Retourne la version de la librairie Yoctopuce utilisée.

cpp vb cs java py php ts dnp ppy

YAPI.GetCacheValidity()

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

cpp vb cs java py php ts ppy

YAPI.GetDeviceListValidity()

Retourne le délai entre chaque énumération forcée des YoctoHubs utilisés.

cpp vb cs java py php ts ppy

YAPI.GetDllArchitecture()

Retourne l'architecture du binaire de la librairie de communication Yoctopuce utilisée.

dnp

YAPI.GetDllPath()

Retourne l'emplacement sur le disque des librairies Yoctopuce utilisées.

dnp

YAPI.GetLog(lastLogLine)

Récupère les messages de logs de la librairie de communication Yoctopuce.

dnp

YAPI.GetNetworkTimeout()

Retourne le délai de connexion réseau pour yRegisterHub() et yUpdateDeviceList().

cpp vb cs java py php ts dnp ppy

YAPI.GetTickCount()

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

cpp vb cs java py php ts ppy

YAPI.GetYAPISharedLibraryPath()

Retourne le chemin de la librairie dynamique YAPI.

cpp vb cs java py php ts

YAPI.HandleEvents(errmsg)

Maintient la communication de la librairie avec les modules Yoctopuce.

cpp vb cs java py php ts ppy

YAPI.InitAPI(mode, errmsg)

Initialise la librairie de programmation de Yoctopuce explicitement.

cpp vb cs java py php ts ppy

YAPI.PreregisterHub(url, errmsg)

Alternative plus tolérante à `yRegisterHub()`.

`cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

YAPI.RegisterDeviceArrivalCallback(arrivalCallback)

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

`cpp` `vb` `cs` `java` `py` `php` `ts` `ppy`

YAPI.RegisterDeviceRemovalCallback(removalCallback)

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

`cpp` `vb` `cs` `java` `py` `php` `ts` `ppy`

YAPI.RegisterHub(url, errmsg)

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

`cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

YAPI.RegisterHubDiscoveryCallback(hubDiscoveryCallback)

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

`cpp` `vb` `cs` `java` `py` `ts` `ppy`

YAPI.RegisterHubWebsocketCallback(ws, errmsg, authpwd)

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

YAPI.RegisterLogFunction(logfun)

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

`cpp` `vb` `cs` `java` `py` `ts` `ppy`

YAPI.SelectArchitecture(arch)

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

`py`

YAPI.SetCacheValidity(cacheValidityMs)

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

`cpp` `vb` `cs` `java` `py` `php` `ts` `ppy`

YAPI.SetDelegate(object)

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

YAPI.SetDeviceListValidity(deviceListValidity)

Change le délai entre chaque énumération forcée des YoctoHub utilisés.

`cpp` `vb` `cs` `java` `py` `php` `ts` `ppy`

YAPI.SetHTTPCallbackCacheDir(directory)

Active le cache du callback HTTP.

`java` `php`

YAPI.SetNetworkSecurityOptions(opts)

Permet d'activer ou désactiver certains checks des certificats TLS/SSL.

`cpp` `vb` `cs` `java` `py` `php` `ts` `ppy`

YAPI.SetNetworkTimeout(networkMsTimeout)

Change le délai de connexion réseau pour `yRegisterHub()` et `yUpdateDeviceList()`.

`cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

YAPI.SetTimeout(callback, ms_timeout, args)

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

ts

YAPI.SetTrustedCertificatesList(certificatePath)

Définit le chemin d'accès au fichier de l'autorité de certification.

cpp vb cs java py php ts ppy

YAPI.SetUSBPacketAckMs(pktAckDelay)

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

java

YAPI.Sleep(ms_duration, errmsg)

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

cpp vb cs java py php ts ppy

YAPI.TestHub(url, mtimeout, errmsg)

Test si un hub est joignable.

cpp vb cs java py php ts dnp ppy

YAPI.TriggerHubDiscovery(errmsg)

Relance une détection des hubs réseau.

cpp vb cs java py ts ppy

YAPI.UnregisterHub(url)

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

cpp vb cs java py php ts ppy

YAPI.UpdateDeviceList(errmsg)

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

cpp vb cs java py php ts ppy

YAPI.UpdateDeviceList_async(callback, context)

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

24.2. La classe YModule

Interface de contrôle des paramètres généraux des modules Yoctopuce

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

cpp	<code>#include "yocto_api.h"</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>
php	<code>require_once('yocto_api.php');</code>
ts	<code>in HTML: import { YAPI, YErrorMsg, YModule, YSensor } from '../dist/esm/yocto_api_browser.js'; in Node.js: import { YAPI, YErrorMsg, YModule, YSensor } from 'yoctolib-cjs/yocto_api_nodejs.js';</code>
dnf	<code>import YoctoProxyAPI.YModuleProxy</code>
cp	<code>#include "yocto_module_proxy.h"</code>
ppy	<code>from yoctolib.yocto_api import *</code>

Fonction globales

YModule.FindModule(func)

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

cpp vb cs java py php ts dnf ppy

YModule.FindModuleInContext(yctx, func)

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

java ts ppy

YModule.FirstModule()

Commence l'énumération des modules accessibles par la librairie.

cpp vb cs java py php ts ppy

Propriétés des objets YModuleProxy

module→Beacon [modifiable]

état de la balise de localisation.

dnf

module→FirmwareRelease [lecture seule]

Version du logiciel embarqué du module.

dnf

module→FunctionId [lecture seule]

Identifiant matériel de la *nième* fonction du module.

dnf

module→HardwareId [lecture seule]

Identifiant unique du module.

dnf

module→IsOnline [lecture seule]

Vérifie si le module est joignable.

dnp

module→LogicalName [modifiable]

Nom logique du module.

dnp

module→Luminosity [modifiable]

Luminosité des leds informatives du module (valeur entre 0 et 100).

dnp

module→ProductId [lecture seule]

Identifiant USB du module, préprogrammé en usine.

dnp

module→ProductName [lecture seule]

Nom commercial du module, préprogrammé en usine.

dnp

module→ProductRelease [lecture seule]

Numéro uméro de révision du module hardware, préprogrammé en usine.

dnp

module→SerialNumber [lecture seule]

Numéro de série du module, préprogrammé en usine.

dnp

Méthodes des objets YModule**module→addFileToHTTPCallback(filename)**

Ajoute un fichier aux données uploadées lors du prochain callback HTTP.

cmd cpp vb cs java py php ts dnp

module→checkFirmware(path, onlynew)

Teste si le fichier byn est valide pour le module.

cmd cpp vb cs java py php ts dnp ppy

module→clearCache()

Invalide le cache.

cpp vb cs java py php ts

module→describe()

Retourne un court texte décrivant le module.

cpp vb cs java py php ts

module→download(pathname)

Télécharge le fichier choisi du module et retourne son contenu.

cmd cpp vb cs java py php ts dnp ppy

module→functionBaseType(functionIndex)

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

cpp vb cs java py php ts

module→functionCount()

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

cpp vb cs java py php ts

module→functionId(functionIndex)

Retourne l'identifiant matériel de la *nième* fonction du module.

cpp vb cs java py php ts

module→functionName(functionIndex)

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

cpp vb cs java py php ts

module→functionType(functionIndex)

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

cpp vb cs java py php ts

module→functionValue(functionIndex)

Retourne la valeur publiée par la *nième* fonction du module.

cpp vb cs java py php ts

module→get_allSettings()

Retourne tous les paramètres de configuration du module.

cmd cpp vb cs java py php ts dnp ppy

module→get_beacon()

Retourne l'état de la balise de localisation.

cmd cpp vb cs java py php ts dnp ppy

module→get_errorMessage()

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

cpp vb cs java py php ts

module→get_errorType()

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

cpp vb cs java py php ts

module→get_firmwareRelease()

Retourne la version du logiciel embarqué du module.

cmd cpp vb cs java py php ts dnp ppy

module→get_functionIds(funType)

Retourne les identifiants matériels des fonctions correspondant au type passé en argument.

cmd cpp vb cs java py php ts dnp

module→get_hardwareId()

Retourne l'identifiant unique du module.

cpp vb cs java py php ts dnp ppy cmd

module→get_icon2d()

Retourne l'icône du module.

cmd cpp vb cs java py php ts dnp ppy

module→get_lastLogs()

Retourne une chaîne de caractère contenant les derniers logs du module.

cmd cpp vb cs java py php ts dnp ppy

module→get_logicalName()

Retourne le nom logique du module.

cpp vb cs java py php ts dnp ppy cmd

module→get_luminosity()

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

cmd cpp vb cs java py php ts dnp ppy

module→get_parentHub()

Retourne le numéro de série du YoctoHub sur lequel est connecté le module.

cmd cpp vb cs java py php ts dnp

module→get_persistentSettings()

Retourne l'état courant des réglages persistents du module.

cmd cpp vb cs java py php ts dnp ppy

module→get_productId()

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

cmd cpp vb cs java py php ts dnp ppy

module→get_productName()

Retourne le nom commercial du module, préprogrammé en usine.

cmd cpp vb cs java py php ts dnp ppy

module→get_productRelease()

Retourne le numéro uméro de révision du module hardware, préprogrammé en usine.

cmd cpp vb cs java py php ts dnp ppy

module→get_rebootCountdown()

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

cmd cpp vb cs java py php ts dnp ppy

module→get_serialNumber()

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

cpp vb cs java py php ts dnp ppy cmd

module→get_subDevices()

Retourne la liste des modules branchés au module courant.

cmd cpp vb cs java py php ts dnp

module→get_upTime()

Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module

cmd cpp vb cs java py php ts dnp ppy

module→get_url()

Retourne l'URL utilisée pour accéder au module.

cmd cpp vb cs java py php ts dnp

module→get_usbCurrent()

Retourne le courant consommé par le module sur le bus USB, en milliampères.

cmd cpp vb cs java py php ts dnp ppy

module→get_userData()

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

cpp vb cs java py php ts

module→get_userVar()

Retourne la valeur entière précédemment stockée dans cet attribut.

cmd cpp vb cs java py php ts dnp ppy

module→hasFunction(funcId)

Teste la présence d'une fonction pour le module courant.

cmd cpp vb cs java py php ts dnp ppy

module→isOnline()

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

cpp vb cs java py php ts dnp

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.

cpp vb cs java py php ts

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.

cmd cpp vb cs java py php ts dnp ppy

module→nextModule()

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

cpp vb cs java py php ts ppy

module→reboot(secBeforeReboot)

Agende un simple redémarrage du module dans un nombre donné de secondes.

cmd cpp vb cs java py php ts dnp ppy

module→registerBeaconCallback(callback)

Enregistre une fonction de callback qui sera appelée à chaque changement d'état de la balise de localisation du module.

cpp vb cs java py php ts ppy

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.)

cpp vb cs java py php ts ppy

module→registerLogCallback(callback)

Enregistre une fonction de callback qui sera appelée à chaque fois le module émet un message de log.

cpp vb cs java py php ts ppy

module→revertFromFlash()

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

cmd cpp vb cs java py php ts dnp ppy

module→saveToFlash()

Sauve les réglages courants dans la mémoire non volatile du module.

cmd cpp vb cs java py php ts dnp ppy

module→set_allSettings(settings)

Rétablit tous les paramètres du module.

cmd cpp vb cs java py php ts dnp ppy

module→set_allSettingsAndFiles(settings)

Rétablit tous les paramètres de configuration et fichiers sur un module.

cmd cpp vb cs java py php ts dnp ppy

module→set_beacon(newval)

Allume ou éteint la balise de localisation du module.

cmd cpp vb cs java py php ts dnp ppy

module→set_logicalName(newval)

Change le nom logique du module.

cpp vb cs java py php ts dnp ppy cmd

module→set_luminosity(newval)

Modifie la luminosité des leds informatives du module.

cmd cpp vb cs java py php ts dnp ppy

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.

cpp vb cs java py php ts

module→set_userVar(newval)

Stocke une valeur 32 bits dans la mémoire volatile du module.

cmd cpp vb cs java py php ts dnp ppy

module→triggerConfigChangeCallback()

Force le déclenchement d'un callback de changement de configuration, afin de vérifier s'ils sont disponibles ou pas.

cmd cpp vb cs java py php ts dnp ppy

module→triggerFirmwareUpdate(secBeforeReboot)

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

cmd cpp vb cs java py php ts dnp ppy

module→updateFirmware(path)

Prepares une mise à jour de firmware du module.

cmd cpp vb cs java py php ts dnp ppy

module→updateFirmwareEx(path, force)

Prepares une mise à jour de firmware du module.

cmd cpp vb cs java py php ts dnp ppy

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.

ts

24.3. La classe YMultiCellWeighScale

Interface pour interagir avec les capteurs de poids multi-cellules, disponibles par exemple dans le Yocto-MaxiBridge

La classe YMultiCellWeighScale permet d'obtenir une mesure de poids à partir d'un groupe de capteurs ratiométrique. Elle permet de contrôler globalement le mode d'excitation des cellules, pour éviter les dérives liées aux changements de température de l'électronique, et d'appliquer automatiquement une correction supplémentaire en fonction de la température pour compenser la dérive des cellules de poids elles-mêmes.

Pour utiliser les fonctions décrites ici, vous devez inclure:

es	in HTML: <code><script src="../../lib/yocto_multicellweighscale.js"></script></code> in node.js: <code>require('yoctolib-es2017/yocto_multicellweighscale.js');</code>
cpp	<code>#include "yocto_multicellweighscale.h"</code>
vb	<code>yocto_multicellweighscale.vb</code>
cs	<code>yocto_multicellweighscale.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YMultiCellWeighScale;</code>
py	<code>from yocto_multicellweighscale import *</code>
php	<code>require_once('yocto_multicellweighscale.php');</code>
ts	in HTML: <code>import { YMultiCellWeighScale } from '../../dist/esm/yocto_multicellweighscale.js';</code> in Node.js: <code>import { YMultiCellWeighScale } from 'yoctolib-cjs/yocto_multicellweighscale.js';</code>
dnp	<code>import YoctoProxyAPI.YMultiCellWeighScaleProxy</code>
cp	<code>#include "yocto_multicellweighscale_proxy.h"</code>
ppy	<code>from yoctolib.yocto_multicellweighscale import *</code>

Fonction globales

YMultiCellWeighScale.FindMultiCellWeighScale(func)

Permet de retrouver un capteur de poids multi-cellules d'après un identifiant donné.

cpp vb cs java py php ts dnp ppy

YMultiCellWeighScale.FindMultiCellWeighScaleInContext(yctx, func)

Permet de retrouver un capteur de poids multi-cellules d'après un identifiant donné dans un Context YAPI.

java ts ppy

YMultiCellWeighScale.FirstMultiCellWeighScale()

Commence l'énumération des capteurs de poids multi-cellules accessibles par la librairie.

cpp vb cs java py php ts ppy

YMultiCellWeighScale.FirstMultiCellWeighScaleInContext(yctx)

Commence l'énumération des capteurs de poids multi-cellules accessibles par la librairie.

java ts ppy

YMultiCellWeighScale.GetSimilarFunctions()

Enumère toutes les fonctions de type MultiCellWeighScale disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

dnp

Propriétés des objets YMultiCellWeighScaleProxy

multicellweighscale→AdvMode [modifiable]

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

	dnp
multicellweighscale→AdvertisedValue [lecture seule] Courte chaîne de caractères représentant l'état courant de la fonction.	dnp
multicellweighscale→CellCount [modifiable] Nombre de cellules de poids utilisées.	dnp
multicellweighscale→Excitation [modifiable] Méthode d'excitation de la cellule de poids.	dnp
multicellweighscale→ExternalSense [modifiable] Vrai si l'entrée 4 est utilisée comme référence externe pour les cellules à 6 fils.	dnp
multicellweighscale→FriendlyName [lecture seule] Identifiant global de la fonction au format NOM_MODULE . NOM_FONCTION.	dnp
multicellweighscale→FunctionId [lecture seule] Identifiant matériel du senseur, sans référence au module.	dnp
multicellweighscale→HardwareId [lecture seule] Identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.	dnp
multicellweighscale→IsOnline [lecture seule] Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.	dnp
multicellweighscale→LogFrequency [modifiable] Fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.	dnp
multicellweighscale→LogicalName [modifiable] Nom logique de la fonction.	dnp
multicellweighscale→ReportFrequency [modifiable] Fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.	dnp
multicellweighscale→Resolution [modifiable] Résolution des valeurs mesurées.	dnp
multicellweighscale→SerialNumber [lecture seule] Numéro de série du module, préprogrammé en usine.	

dnf

multicellweighscale→TempAvgAdaptRatio [modifiable]

Taux d'adaptation pour le calcul de la température moyenne, en pour mille.

dnf

multicellweighscale→TempChgAdaptRatio [modifiable]

Taux d'adaptation pour le calcul de la variation de température, en pour mille.

dnf

multicellweighscale→ZeroTracking [modifiable]

Valeur seuil pour le suivi continu du zéro.

dnf

Méthodes des objets YMultiCellWeighScale**multicellweighscale→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.

cmd cpp vb cs java py php ts dnf ppy

multicellweighscale→clearCache()

Invalide le cache.

cpp vb cs java py php ts

multicellweighscale→describe()

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

cpp vb cs java py php ts

multicellweighscale→get_advMode()

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

cmd cpp vb cs java py php ts dnf ppy

multicellweighscale→get_advertisedValue()

Retourne la valeur courante du capteur de poids multi-cellules (pas plus de 6 caractères).

cpp vb cs java py php ts dnf ppy cmd

multicellweighscale→get_cellCount()

Retourne le nombre de cellules de poids utilisées.

cmd cpp vb cs java py php ts dnf ppy

multicellweighscale→get_compTempAvg()

Retourne la température moyennée utilisée actuellement pour la compensation thermique.

cmd cpp vb cs java py php ts dnf ppy

multicellweighscale→get_compTempChg()

Retourne la variation actuelle de la température, utilisée pour la compensation thermique.

cmd cpp vb cs java py php ts dnf ppy

multicellweighscale→get_compensation()

Retourne la valeur de la compensation thermique courante.

cmd cpp vb cs java py php ts dnf ppy

multicellweighscale→get_currentRawValue()

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

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→get_currentValue()

Retourne la valeur actuelle de la mesure, en l'unité spécifiée, sous forme de nombre à virgule.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→get_dataLogger()

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

cpp vb cs java py php ts dnp ppy

multicellweighscale→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du capteur de poids multi-cellules.

cpp vb cs java py php ts

multicellweighscale→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du capteur de poids multi-cellules.

cpp vb cs java py php ts

multicellweighscale→get_excitation()

Retourne la méthode d'excitation de la cellule de poids.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→get_externalSense()

Retourne vrai si l'entrée 4 est utilisée comme référence externe pour les cellules à 6 fils.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→get_friendlyName()

Retourne un identifiant global du capteur de poids multi-cellules au format NOM_MODULE.NOM_FONCTION.

cpp cs java py php ts dnp

multicellweighscale→get_functionDescriptor()

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

cpp vb cs java py php ts

multicellweighscale→get_functionId()

Retourne l'identifiant matériel du capteur de poids multi-cellules, sans référence au module.

cpp vb cs java py php ts dnp ppy

multicellweighscale→get_hardwareId()

Retourne l'identifiant matériel unique du capteur de poids multi-cellules au format SERIAL.FUNCTIONID.

cpp vb cs java py php ts dnp ppy

multicellweighscale→get_highestValue()

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→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.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→get_logicalName()

Retourne le nom logique du capteur de poids multi-cellules.

cpp vb cs java py php ts dnp ppy cmd

multicellweighscale→get_lowestValue()

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→get_module()

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

cpp vb cs java py php ts dnp

multicellweighscale→get_module_async(callback, context)

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

multicellweighscale→get_recordedData(startTime, endTime)

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

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→get_reportFrequency()

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

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→get_resolution()

Retourne la résolution des valeurs mesurées.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→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.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→get_serialNumber()

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

cpp vb cs java py php ts dnp ppy cmd

multicellweighscale→get_tempAvgAdaptRatio()

Retourne le taux d'adaptation pour le calcul de la température moyenne, en pour mille.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→get_tempChgAdaptRatio()

Retourne le taux d'adaptation pour le calcul de la variation de température, en pour mille.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→get_unit()

Retourne l'unité dans laquelle la mesure est exprimée.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→get_userData()

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

cpp vb cs java py php ts

multicellweighscale→get_zeroTracking()

Retourne la valeur seuil pour le suivi continu du zéro.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→isOnline()

Vérifie si le module hébergeant le capteur de poids multi-cellules est joignable, sans déclencher d'erreur.

cpp vb cs java py php ts dnp

multicellweighscale→isOnline_async(callback, context)

Vérifie si le module hébergeant le capteur de poids multi-cellules est joignable, sans déclencher d'erreur.

multicellweighscale→isReadOnly()

Indique si la fonction est en lecture seule.

cpp vb cs java py php ts dnp cmd

multicellweighscale→isSensorReady()

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

cmd

multicellweighscale→load(msValidity)

Met en cache les valeurs courantes du capteur de poids multi-cellules, avec une durée de validité spécifiée.

cpp vb cs java py php ts

multicellweighscale→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.

cpp vb cs java py php ts dnp ppy

multicellweighscale→loadCalibrationPoints(rawValues, refValues)

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

cmd cpp vb cs java py php ts ppy

multicellweighscale→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du capteur de poids multi-cellules, avec une durée de validité spécifiée.

multicellweighscale→muteValueCallbacks()

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

cpp vb cs java py php ts dnp ppy cmd

multicellweighscale→nextMultiCellWeighScale()

Continue l'énumération des capteurs de poids multi-cellules commencée à l'aide de `yFirstMultiCellWeighScale()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les capteurs de poids multi-cellules sont retournés.

cpp vb cs java py php ts ppy

multicellweighscale→registerTimedReportCallback(callback)

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

cpp vb cs java py php ts ppy

multicellweighscale→registerValueCallback(callback)

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

cpp vb cs java py php ts ppy

multicellweighscale→set_advMode(newval)

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

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→set_cellCount(newval)

Modifie le nombre de cellules de poids utilisées.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→set_excitation(newval)

Modifie la méthode d'excitation de la cellule de poids.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→set_externalSense(newval)

Modifie la configuration indiquant si l'entrée 4 est utilisée comme référence externe pour les cellules à 6 fils ou pas.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→set_logFrequency(newval)

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

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→set_logicalName(newval)

Modifie le nom logique du capteur de poids multi-cellules.

cpp vb cs java py php ts dnp ppy cmd

multicellweighscale→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→set_tempAvgAdaptRatio(newval)

Modifie le taux d'adaptation pour le calcul de la température moyenne, en pour mille.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→set_tempChgAdaptRatio(newval)

Modifie le taux d'adaptation pour le calcul de la variation de température, en pour mille.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→set_unit(newval)

Modifie l'unité dans laquelle le poids est exprimé.

cmd cpp vb cs java py php ts dnp ppy

multicellweighscale→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`.

cpp vb cs java py php ts

`multicellweighscale→set_zeroTracking(newval)`

Modifie la valeur seuil pour le suivi continu du zéro.

cmd cpp vb cs java py php ts dnp ppy

`multicellweighscale→setupSpan(currWeight, maxWeight)`

Configure le facteur de poids et l'amplitude maximale des cellules de poids (stockés dans les `genericSensors` correspondants) de sorte à ce que le signal actuel corresponde au poids de référence spécifié.

cmd cpp vb cs java py php ts dnp ppy

`multicellweighscale→startDataLogger()`

Démarre l'enregistreur de données du module.

cmd cpp vb cs java py php ts dnp ppy

`multicellweighscale→stopDataLogger()`

Arrête l'enregistreur de données du module.

cmd cpp vb cs java py php ts dnp ppy

`multicellweighscale→tare()`

Adapte le biais de mesure des cellules de poids (stocké dans les `genericSensors` correspondants) de sorte à ce que le signal actuel corresponde à un poids nul.

cmd cpp vb cs java py php ts dnp ppy

`multicellweighscale→unmuteValueCallbacks()`

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

cpp vb cs java py php ts dnp ppy cmd

`multicellweighscale→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.

ts

24.4. La classe YWeighScale

Interface pour interagir avec les capteurs de poids, disponibles par exemple dans le Yocto-Bridge et le Yocto-MaxiBridge

La classe YWeighScale permet d'obtenir une mesure de poids à partir d'un capteur ratiométrique. Elle permet de contrôler le mode d'excitation de la cellule, pour éviter les dérives liées aux changements de température de l'électronique, et d'appliquer automatiquement une correction supplémentaire en fonction de la température pour compenser la dérive de la cellule de poids elle-même.

Pour utiliser les fonctions décrites ici, vous devez inclure:

es	in HTML: <code><script src="../../lib/yocto_weighscale.js"></script></code> in node.js: <code>require('yoctolib-es2017/yocto_weighscale.js');</code>
cpp	<code>#include "yocto_weighscale.h"</code>
vb	<code>yocto_weighscale.vb</code>
cs	<code>yocto_weighscale.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YWeighScale;</code>
py	<code>from yocto_weighscale import *</code>
php	<code>require_once('yocto_weighscale.php');</code>
ts	in HTML: <code>import { YWeighScale } from '../../dist/esm/yocto_weighscale.js';</code> in Node.js: <code>import { YWeighScale } from 'yoctolib-cjs/yocto_weighscale.js';</code>
dnp	<code>import YoctoProxyAPI.YWeighScaleProxy</code>
cp	<code>#include "yocto_weighscale_proxy.h"</code>
ppy	<code>from yoctolib.yocto_weighscale import *</code>

Fonction globales

YWeighScale.FindWeighScale(func)

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

cpp vb cs java py php ts dnp ppy

YWeighScale.FindWeighScaleInContext(yctx, func)

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

java ts ppy

YWeighScale.FirstWeighScale()

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

cpp vb cs java py php ts ppy

YWeighScale.FirstWeighScaleInContext(yctx)

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

java ts ppy

YWeighScale.GetSimilarFunctions()

Enumère toutes les fonctions de type WeighScale disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

dnp

Propriétés des objets YWeighScaleProxy

weighscale→AdvMode [modifiable]

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

	dnp
weighscale→AdvertisedValue [lecture seule] Courte chaîne de caractères représentant l'état courant de la fonction.	dnp
weighscale→Excitation [modifiable] Méthode d'excitation de la cellule de poids.	dnp
weighscale→FriendlyName [lecture seule] Identifiant global de la fonction au format NOM_MODULE . NOM_FONCTION.	dnp
weighscale→FunctionId [lecture seule] Identifiant matériel du senseur, sans référence au module.	dnp
weighscale→HardwareId [lecture seule] Identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.	dnp
weighscale→IsOnline [lecture seule] Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.	dnp
weighscale→LogFrequency [modifiable] Fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.	dnp
weighscale→LogicalName [modifiable] Nom logique de la fonction.	dnp
weighscale→ReportFrequency [modifiable] Fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.	dnp
weighscale→Resolution [modifiable] Résolution des valeurs mesurées.	dnp
weighscale→SerialNumber [lecture seule] Numéro de série du module, préprogrammé en usine.	dnp
weighscale→TempAvgAdaptRatio [modifiable] Taux d'adaptation pour le calcul de la température moyenne, en pour mille.	dnp
weighscale→TempChgAdaptRatio [modifiable] Taux d'adaptation pour le calcul de la variation de température, en pour mille.	

dnp

weighscale→ZeroTracking [modifiable]

Valeur seuil pour le suivi continu du zéro.

dnp

Méthodes des objets YWeighScale**weighscale→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.

cmd cpp vb cs java py php ts dnp ppy

weighscale→clearCache()

Invalide le cache.

cpp vb cs java py php ts

weighscale→describe()

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

cpp vb cs java py php ts

weighscale→get_advMode()

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

cmd cpp vb cs java py php ts dnp ppy

weighscale→get_advertisedValue()

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

cpp vb cs java py php ts dnp ppy cmd

weighscale→get_compTempAvg()

Retourne la température moyennée utilisée actuellement pour la compensation thermique.

cmd cpp vb cs java py php ts dnp ppy

weighscale→get_compTempChg()

Retourne la variation actuelle de la température, utilisée pour la compensation thermique.

cmd cpp vb cs java py php ts dnp ppy

weighscale→get_compensation()

Retourne la valeur de la compensation thermique courante.

cmd cpp vb cs java py php ts dnp ppy

weighscale→get_currentRawValue()

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

cmd cpp vb cs java py php ts dnp ppy

weighscale→get_currentValue()

Retourne la valeur actuelle de la mesure, en l'unité spécifiée, sous forme de nombre à virgule.

cmd cpp vb cs java py php ts dnp ppy

weighscale→get_dataLogger()

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

cpp vb cs java py php ts dnp ppy

weighscale→get_errorMessage()

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

`cpp` `vb` `cs` `java` `py` `php` `ts`

`weighscale→get_errorType()`

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

`cpp` `vb` `cs` `java` `py` `php` `ts`

`weighscale→get_excitation()`

Retourne la méthode d'excitation de la cellule de poids.

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

`weighscale→get_friendlyName()`

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

`cpp` `cs` `java` `py` `php` `ts` `dnp`

`weighscale→get_functionDescriptor()`

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

`cpp` `vb` `cs` `java` `py` `php` `ts`

`weighscale→get_functionId()`

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

`cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

`weighscale→get_hardwareId()`

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

`cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

`weighscale→get_highestValue()`

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

`weighscale→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.

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

`weighscale→get_logicalName()`

Retourne le nom logique du capteur de poids.

`cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy` `cmd`

`weighscale→get_lowestValue()`

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

`weighscale→get_module()`

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

`cpp` `vb` `cs` `java` `py` `php` `ts` `dnp`

`weighscale→get_module_async(callback, context)`

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

`weighscale→get_recordedData(startTime, endTime)`

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

cmd cpp vb cs java py php ts dnp ppy

weighscale→get_reportFrequency()

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

cmd cpp vb cs java py php ts dnp ppy

weighscale→get_resolution()

Retourne la résolution des valeurs mesurées.

cmd cpp vb cs java py php ts dnp ppy

weighscale→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.

cmd cpp vb cs java py php ts dnp ppy

weighscale→get_serialNumber()

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

cpp vb cs java py php ts dnp ppy cmd

weighscale→get_tempAvgAdaptRatio()

Retourne le taux d'adaptation pour le calcul de la température moyenne, en pour mille.

cmd cpp vb cs java py php ts dnp ppy

weighscale→get_tempChgAdaptRatio()

Retourne le taux d'adaptation pour le calcul de la variation de température, en pour mille.

cmd cpp vb cs java py php ts dnp ppy

weighscale→get_unit()

Retourne l'unité dans laquelle la mesure est exprimée.

cmd cpp vb cs java py php ts dnp ppy

weighscale→get_userData()

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

cpp vb cs java py php ts

weighscale→get_zeroTracking()

Retourne la valeur seuil pour le suivi continu du zéro.

cmd cpp vb cs java py php ts dnp ppy

weighscale→isOnline()

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

cpp vb cs java py php ts dnp

weighscale→isOnline_async(callback, context)

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

weighscale→isReadOnly()

Indique si la fonction est en lecture seule.

cpp vb cs java py php ts dnp cmd

weighscale→isSensorReady()

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

cmd

weighscale→load(msValidity)

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

cpp vb cs java py php ts

weighscale→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.

cpp vb cs java py php ts dnp ppy

weighscale→loadCalibrationPoints(rawValues, refValues)

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

cmd cpp vb cs java py php ts ppy

weighscale→loadOffsetAvgCompensationTable(tempValues, compValues)

Récupère la table de compensation de l'offset de poids en fonction de la température moyennée précédemment enregistrée à l'aide de la fonction `set_offsetAvgCompensationTable`.

cmd cpp vb cs java py php ts ppy

weighscale→loadOffsetChgCompensationTable(tempValues, compValues)

Récupère la table de compensation en température de l'offset de poids en fonction de la variation de température précédemment enregistrée à l'aide de la fonction `set_offsetChgCompensationTable`.

cmd cpp vb cs java py php ts ppy

weighscale→loadSpanAvgCompensationTable(tempValues, compValues)

Récupère la table de compensation en température de l'amplitude de poids en fonction de la température moyennée précédemment enregistrée à l'aide de la fonction `set_spanAvgCompensationTable`.

cmd cpp vb cs java py php ts ppy

weighscale→loadSpanChgCompensationTable(tempValues, compValues)

Récupère la table de compensation de l'amplitude de poids en fonction de la variation de température, précédemment enregistrée à l'aide de la fonction `set_spanChgCompensationTable`.

cmd cpp vb cs java py php ts ppy

weighscale→load_async(msValidity, callback, context)

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

weighscale→muteValueCallbacks()

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

cpp vb cs java py php ts dnp ppy cmd

weighscale→nextWeighScale()

Continue l'énumération des capteurs de poids commencée à l'aide de `yFirstWeighScale()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les capteurs de poids sont retournés.

cpp vb cs java py php ts ppy

weighscale→registerTimedReportCallback(callback)

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

cpp vb cs java py php ts ppy

weighscale→registerValueCallback(callback)

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

cpp vb cs java py php ts ppy

weighscale→set_advMode(newval)

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

cmd cpp vb cs java py php ts dnp ppy

weighscale→set_excitation(newval)

Modifie la méthode d'excitation de la cellule de poids.

cmd cpp vb cs java py php ts dnp ppy

weighscale→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

cmd cpp vb cs java py php ts dnp ppy

weighscale→set_logFrequency(newval)

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

cmd cpp vb cs java py php ts dnp ppy

weighscale→set_logicalName(newval)

Modifie le nom logique du capteur de poids.

cpp vb cs java py php ts dnp ppy cmd

weighscale→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

cmd cpp vb cs java py php ts dnp ppy

weighscale→set_offsetAvgCompensationTable(tempValues, compValues)

Enregistre la table de compensation de l'offset de poids en fonction de la température moyennée, afin de pouvoir corriger automatiquement le poids mesuré sur la base de la température moyennée.

cmd cpp vb cs java py php ts dnp ppy

weighscale→set_offsetChgCompensationTable(tempValues, compValues)

Enregistre la table de compensation de l'offset de poids en fonction de la variation de température, afin de pouvoir corriger automatiquement le poids mesuré sur la base de la variation de la température.

cmd cpp vb cs java py php ts dnp ppy

weighscale→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

cmd cpp vb cs java py php ts dnp ppy

weighscale→set_resolution(newval)

Modifie la résolution des valeurs physique mesurées.

cmd cpp vb cs java py php ts dnp ppy

weighscale→set_spanAvgCompensationTable(tempValues, compValues)

Enregistre la table de compensation en température de l'amplitude du poids en fonction de la température moyennée, afin de pouvoir corriger automatiquement le poids mesuré sur la base de la température moyennée de compensation.

cmd cpp vb cs java py php ts dnp ppy

weighscale→set_spanChgCompensationTable(tempValues, compValues)

Enregistre la table de compensation de l'amplitude du poids en fonction de la variation de température, afin de pouvoir corriger automatiquement le poids mesuré sur la base de la variation de température.

cmd cpp vb cs java py php ts dnp ppy

weighscale→set_tempAvgAdaptRatio(newval)

Modifie le taux d'adaptation pour le calcul de la température moyenne, en pour mille.

cmd cpp vb cs java py php ts dnp ppy

weighscale→set_tempChgAdaptRatio(newval)

Modifie le taux d'adaptation pour le calcul de la variation de température, en pour mille.

cmd cpp vb cs java py php ts dnp ppy

weighscale→set_unit(newval)

Modifie l'unité dans laquelle le poids est exprimé.

cmd cpp vb cs java py php ts dnp ppy

weighscale→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.

cpp vb cs java py php ts

weighscale→set_zeroTracking(newval)

Modifie la valeur seuil pour le suivi continu du zéro.

cmd cpp vb cs java py php ts dnp ppy

weighscale→setupSpan(currWeight, maxWeight)

Configure le facteur de poids et l'amplitude maximale de la cellule de poids (stockés dans le genericSensor correspondant) de sorte à ce que le signal actuel corresponde au poids de référence spécifié.

cmd cpp vb cs java py php ts dnp ppy

weighscale→startDataLogger()

Démarre l'enregistreur de données du module.

cmd cpp vb cs java py php ts dnp ppy

weighscale→stopDataLogger()

Arrête l'enregistreur de données du module.

cmd cpp vb cs java py php ts dnp ppy

weighscale→tare()

Adapte le biais de mesure de la cellule de poids (stocké dans le genericSensor correspondant) de sorte à ce que le signal actuel corresponde à un poids nul.

cmd cpp vb cs java py php ts dnp ppy

weighscale→unmuteValueCallbacks()

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

cpp vb cs java py php ts dnp ppy cmd

weighscale→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.

ts

24.5. La classe YGenericSensor

Interface pour interagir avec les capteurs de type GenericSensor, disponibles par exemple dans le Yocto-0-10V-Rx, le Yocto-4-20mA-Rx, le Yocto-Bridge et le Yocto-milliVolt-Rx

La classe YGenericSensor permet de lire et de configurer les transducteurs de signaux Yoctopuce. Elle hérite de la classe YSensor toutes les fonctions de base des capteurs Yoctopuce: lecture de mesures, callbacks, enregistreur de données. De plus, elle permet de configurer une conversion automatique entre le signal mesuré et la grandeur physique représentée.

Pour utiliser les fonctions décrites ici, vous devez inclure:

cpp	<code>#include "yocto_genericsensor.h"</code>
vb	<code>yocto_genericsensor.vb</code>
cs	<code>yocto_genericsensor.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YGenericSensor;</code>
py	<code>from yocto_genericsensor import *</code>
php	<code>require_once('yocto_genericsensor.php');</code>
ts	<code>in HTML: import { YGenericSensor } from '../dist/esm/yocto_genericsensor.js'; in Node.js: import { YGenericSensor } from 'yoctolib-cjs/yocto_genericsensor.js';</code>
dnp	<code>import YoctoProxyAPI.YGenericSensorProxy</code>
cp	<code>#include "yocto_genericsensor_proxy.h"</code>
ppy	<code>from yoctolib.yocto_genericsensor import *</code>

Fonction globales

YGenericSensor.FindGenericSensor(func)

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

cpp vb cs java py php ts dnp ppy

YGenericSensor.FindGenericSensorInContext(yctx, func)

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

java ts ppy

YGenericSensor.FirstGenericSensor()

Commence l'énumération des capteurs génériques accessibles par la librairie.

cpp vb cs java py php ts ppy

YGenericSensor.FirstGenericSensorInContext(yctx)

Commence l'énumération des capteurs génériques accessibles par la librairie.

java ts ppy

YGenericSensor.GetSimilarFunctions()

Enumère toutes les fonctions de type GenericSensor disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

dnp

Propriétés des objets YGenericSensorProxy

genericsensor→AdvMode [modifiable]

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

dnp

genericsensor→AdvertisedValue [lecture seule]

Courte chaîne de caractères représentant l'état courant de la fonction.

	dnp
genericsensor→Enabled [modifiable] état d'activation de cette mesure.	dnp
genericsensor→FriendlyName [lecture seule] Identifiant global de la fonction au format NOM_MODULE . NOM_FONCTION.	dnp
genericsensor→FunctionId [lecture seule] Identifiant matériel du senseur, sans référence au module.	dnp
genericsensor→HardwareId [lecture seule] Identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.	dnp
genericsensor→IsOnline [lecture seule] Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.	dnp
genericsensor→LogFrequency [modifiable] Fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.	dnp
genericsensor→LogicalName [modifiable] Nom logique de la fonction.	dnp
genericsensor→ReportFrequency [modifiable] Fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.	dnp
genericsensor→Resolution [modifiable] Résolution des valeurs mesurées.	dnp
genericsensor→SerialNumber [lecture seule] Numéro de série du module, préprogrammé en usine.	dnp
genericsensor→SignalBias [modifiable] Biais du signal électrique pour la correction du point zéro.	dnp
genericsensor→SignalRange [modifiable] Plage de signal d'entrée utilisé par le capteur.	dnp
genericsensor→SignalSampling [modifiable] Méthode d'échantillonnage du signal utilisée.	

[dnf](#)
genericsensor→SignalUnit [lecture seule]

Unité du signal électrique utilisé par le capteur.

[dnf](#)
genericsensor→ValueRange [modifiable]

Plage de valeurs physiques mesurés par le capteur.

[dnf](#)
Méthodes des objets YGenericSensor**genericsensor→calibrateFromPoints(rawValues, refValues)**

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

[cmd](#) [cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnf](#) [ppy](#)
genericsensor→clearCache()

Invalide le cache.

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#)
genericsensor→describe()

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

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#)
genericsensor→get_advMode()

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

[cmd](#) [cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnf](#) [ppy](#)
genericsensor→get_advertisedValue()

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

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnf](#) [ppy](#) [cmd](#)
genericsensor→get_currentRawValue()

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

[cmd](#) [cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnf](#) [ppy](#)
genericsensor→get_currentValue()

Retourne la valeur mesurée actuelle.

[cmd](#) [cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnf](#) [ppy](#)
genericsensor→get_dataLogger()

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

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnf](#) [ppy](#)
genericsensor→get_enabled()

Retourne l'état d'activation de cette mesure.

[cmd](#) [cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnf](#) [ppy](#)
genericsensor→get_errorMessage()

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

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#)
genericsensor→get_errorType()

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

`cpp` `vb` `cs` `java` `py` `php` `ts`

genericsensor→get_friendlyName()

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

`cpp` `cs` `java` `py` `php` `ts` `dnp`

genericsensor→get_functionDescriptor()

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

`cpp` `vb` `cs` `java` `py` `php` `ts`

genericsensor→get_functionId()

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

`cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

genericsensor→get_hardwareId()

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

`cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

genericsensor→get_highestValue()

Retourne la valeur maximale observée pour la mesure depuis le démarrage du module.

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

genericsensor→get_logFrequency()

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

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

genericsensor→get_logicalName()

Retourne le nom logique du capteur générique.

`cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy` `cmd`

genericsensor→get_lowestValue()

Retourne la valeur minimale observée pour la mesure depuis le démarrage du module.

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

genericsensor→get_module()

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

`cpp` `vb` `cs` `java` `py` `php` `ts` `dnp`

genericsensor→get_module_async(callback, context)

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

genericsensor→get_recordedData(startTime, endTime)

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

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

genericsensor→get_reportFrequency()

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

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

genericsensor→get_resolution()

Retourne la résolution des valeurs mesurées.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→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.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→get_serialNumber()

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

cpp vb cs java py php ts dnp ppy cmd

genericsensor→get_signalBias()

Retourne le biais du signal électrique pour la correction du point zéro.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→get_signalRange()

Retourne la plage de signal d'entrée utilisé par le capteur.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→get_signalSampling()

Retourne la méthode d'échantillonnage du signal utilisée.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→get_signalUnit()

Retourne l'unité du signal électrique utilisé par le capteur.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→get_signalValue()

Retourne la valeur actuelle du signal électrique mesuré par le capteur.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→get_unit()

Retourne l'unité dans laquelle la mesure est exprimée.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→get_userData()

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

cpp vb cs java py php ts

genericsensor→get_valueRange()

Retourne la plage de valeurs physiques mesurés par le capteur.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→isOnline()

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

cpp vb cs java py php ts dnp

genericsensor→isOnline_async(callback, context)

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

genericsensor→isReadOnly()

Indique si la fonction est en lecture seule.

cpp vb cs java py php ts dnp cmd

genericSensor→isSensorReady()

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

cmd

genericSensor→load(msValidity)

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

cpp vb cs java py php ts

genericSensor→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.

cpp vb cs java py php ts dnp ppy

genericSensor→loadCalibrationPoints(rawValues, refValues)

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

cmd cpp vb cs java py php ts ppy

genericSensor→load_async(msValidity, callback, context)

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

genericSensor→muteValueCallbacks()

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

cpp vb cs java py php ts dnp ppy cmd

genericSensor→nextGenericSensor()

Continue l'énumération des capteurs génériques commencée à l'aide de `yFirstGenericSensor()`. Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les capteurs génériques sont retournés.

cpp vb cs java py php ts ppy

genericSensor→registerTimedReportCallback(callback)

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

cpp vb cs java py php ts ppy

genericSensor→registerValueCallback(callback)

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

cpp vb cs java py php ts ppy

genericSensor→set_advMode(newval)

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

cmd cpp vb cs java py php ts dnp ppy

genericSensor→set_enabled(newval)

Modifie l'état d'activation de cette mesure.

cmd cpp vb cs java py php ts dnp ppy

genericSensor→set_highestValue(newval)

Modifie la mémoire de valeur maximale observée.

cmd cpp vb cs java py php ts dnp ppy

genericSensor→set_logFrequency(newval)

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

cmd cpp vb cs java py php ts dnp ppy

genericSensor→set_logicalName(newval)

Modifie le nom logique du capteur générique.

cpp vb cs java py php ts dnp ppy cmd

genericsensor→set_lowestValue(newval)

Modifie la mémoire de valeur minimale observée.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→set_reportFrequency(newval)

Modifie la fréquence de notification périodique des valeurs mesurées.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→set_resolution(newval)

Change la résolution des valeurs physique mesurées.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→set_signalBias(newval)

Modifie le biais du signal électrique pour la correction du point zéro.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→set_signalRange(newval)

Modifie la plage du signal d'entrée utilisé par le capteur.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→set_signalSampling(newval)

Modifie la méthode d'échantillonnage du signal à utiliser.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→set_unit(newval)

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

cmd cpp vb cs java py php ts dnp ppy

genericsensor→set_userData(data)

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

cpp vb cs java py php ts

genericsensor→set_valueRange(newval)

Modifie la plage de valeur de sortie, correspondant à la grandeur physique mesurée par le capteur.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→startDataLogger()

Démarre l'enregistreur de données du module.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→stopDataLogger()

Arrête l'enregistreur de données du module.

cmd cpp vb cs java py php ts dnp ppy

genericsensor→unmuteValueCallbacks()

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

cpp vb cs java py php ts dnp ppy cmd

genericsensor→wait_async(callback, context)

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

ts

genericsensor→zeroAdjust()

Ajuste le biais du signal de sorte à ce que la valeur actuelle du signal soit interprétée comme zéro (tare).

cmd

cpp

vb

cs

java

py

php

ts

dnp

ppy

24.6. La classe YTemperature

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

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

cpp	<code>#include "yocto_temperature.h"</code>
vb	<code>yocto_temperature.vb</code>
cs	<code>yocto_temperature.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YTemperature;</code>
py	<code>from yocto_temperature import *</code>
php	<code>require_once('yocto_temperature.php');</code>
ts	<code>in HTML: import { YTemperature } from '../dist/esm/yocto_temperature.js'; in Node.js: import { YTemperature } from 'yoctolib-cjs/yocto_temperature.js';</code>
dnp	<code>import YoctoProxyAPI.YTemperatureProxy</code>
cp	<code>#include "yocto_temperature_proxy.h"</code>
ppy	<code>from yoctolib.yocto_temperature import *</code>

Fonction globales

YTemperature.FindTemperature(func)

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

cpp vb cs java py php ts dnp ppy

YTemperature.FindTemperatureInContext(yctx, func)

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

java ts ppy

YTemperature.FirstTemperature()

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

cpp vb cs java py php ts ppy

YTemperature.FirstTemperatureInContext(yctx)

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

java ts ppy

YTemperature.GetSimilarFunctions()

Enumère toutes les fonctions de type Temperature disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

dnp

Propriétés des objets YTemperatureProxy

temperature→AdvMode [modifiable]

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

dnp

temperature→AdvertisedValue [lecture seule]

Courte chaîne de caractères représentant l'état courant de la fonction.

	dnp
temperature→FriendlyName <i>[lecture seule]</i>	
Identifiant global de la fonction au format NOM_MODULE . NOM_FONCTION.	dnp
temperature→FunctionId <i>[lecture seule]</i>	
Identifiant matériel du senseur, sans référence au module.	dnp
temperature→HardwareId <i>[lecture seule]</i>	
Identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.	dnp
temperature→IsOnline <i>[lecture seule]</i>	
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.	dnp
temperature→LogFrequency <i>[modifiable]</i>	
Fréquence d'enregistrement des mesures dans le datalogger, ou "OFF" si les mesures ne sont pas stockées dans la mémoire de l'enregistreur de données.	dnp
temperature→LogicalName <i>[modifiable]</i>	
Nom logique de la fonction.	dnp
temperature→ReportFrequency <i>[modifiable]</i>	
Fréquence de notification périodique des valeurs mesurées, ou "OFF" si les notifications périodiques sont désactivées pour cette fonction.	dnp
temperature→Resolution <i>[modifiable]</i>	
Résolution des valeurs mesurées.	dnp
temperature→SensorType <i>[modifiable]</i>	
Type de capteur de température utilisé par le module Modifiable .	dnp
temperature→SerialNumber <i>[lecture seule]</i>	
Numéro de série du module, préprogrammé en usine.	dnp
temperature→SignalUnit <i>[lecture seule]</i>	
Unité du signal électrique utilisé par le capteur.	dnp
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.	
	cmd cpp vb cs java py php ts dnp ppy
temperature→clearCache()	

Invalide le cache.

cpp vb cs java py php ts

temperature→describe()

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

cpp vb cs java py php ts

temperature→get_advMode()

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

cmd cpp vb cs java py php ts dnp ppy

temperature→get_advertisedValue()

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

cpp vb cs java py php ts dnp ppy cmd

temperature→get_currentRawValue()

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

cmd cpp vb cs java py php ts dnp ppy

temperature→get_currentValue()

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

cmd cpp vb cs java py php ts dnp ppy

temperature→get_dataLogger()

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

cpp vb cs java py php ts dnp ppy

temperature→get_errorMessage()

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

cpp vb cs java py php ts

temperature→get_errorType()

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

cpp vb cs java py php ts

temperature→get_friendlyName()

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

cpp cs java py php ts dnp

temperature→get_functionDescriptor()

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

cpp vb cs java py php ts

temperature→get_functionId()

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

cpp vb cs java py php ts dnp ppy

temperature→get_hardwareId()

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

cpp vb cs java py php ts dnp ppy

temperature→get_highestValue()

Retourne la valeur maximale observée pour la température depuis le démarrage du module.

cmd cpp vb cs java py php ts dnp ppy

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.

cmd cpp vb cs java py php ts dnp ppy

temperature→get_logicalName()

Retourne le nom logique du capteur de température.

cpp vb cs java py php ts dnp ppy cmd

temperature→get_lowestValue()

Retourne la valeur minimale observée pour la température depuis le démarrage du module.

cmd cpp vb cs java py php ts dnp ppy

temperature→get_module()

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

cpp vb cs java py php ts dnp

temperature→get_module_async(callback, context)

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

temperature→get_recordedData(startTime, endTime)

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

cmd cpp vb cs java py php ts dnp ppy

temperature→get_reportFrequency()

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

cmd cpp vb cs java py php ts dnp ppy

temperature→get_resolution()

Retourne la résolution des valeurs mesurées.

cmd cpp vb cs java py php ts dnp ppy

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.

cmd cpp vb cs java py php ts dnp ppy

temperature→get_sensorType()

Retourne le type de capteur de température utilisé par le module

cmd cpp vb cs java py php ts dnp ppy

temperature→get_serialNumber()

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

cpp vb cs java py php ts dnp ppy cmd

temperature→get_signalUnit()

Retourne l'unité du signal électrique utilisé par le capteur.

cmd cpp vb cs java py php ts dnp ppy

temperature→get_signalValue()

Retourne la valeur actuelle du signal électrique mesuré par le capteur.

cmd cpp vb cs java py php ts dnp ppy

temperature→get_unit()

Retourne l'unité dans laquelle la température est exprimée.

cmd cpp vb cs java py php ts dnp ppy

temperature→get_userData()

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

cpp vb cs java py php ts

temperature→isOnline()

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

cpp vb cs java py php ts dnp

temperature→isOnline_async(callback, context)

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

temperature→isReadOnly()

Indique si la fonction est en lecture seule.

cpp vb cs java py php ts dnp cmd

temperature→isSensorReady()

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

cmd

temperature→load(msValidity)

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

cpp vb cs java py php ts

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.

cpp vb cs java py php ts dnp ppy

temperature→loadCalibrationPoints(rawValues, refValues)

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

cmd cpp vb cs java py php ts ppy

temperature→loadThermistorResponseTable(tempValues, resValues)

Récupère la table de réponse d'un thermistor précédemment enregistrée à l'aide de la fonction set_thermistorResponseTable.

cmd cpp vb cs java py php ts ppy

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.

cpp vb cs java py php ts dnp ppy cmd

temperature→nextTemperature()

Continue l'énumération des capteurs de température commencée à l'aide de `yFirstTemperature()`. Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les capteurs de température sont retournés.

`cpp` `vb` `cs` `java` `py` `php` `ts` `ppy`

`temperature→registerTimedReportCallback(callback)`

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

`cpp` `vb` `cs` `java` `py` `php` `ts` `ppy`

`temperature→registerValueCallback(callback)`

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

`cpp` `vb` `cs` `java` `py` `php` `ts` `ppy`

`temperature→set_advMode(newval)`

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

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

`temperature→set_highestValue(newval)`

Modifie la mémoire de valeur maximale observée.

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

`temperature→set_logFrequency(newval)`

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

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

`temperature→set_logicalName(newval)`

Modifie le nom logique du capteur de température.

`cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy` `cmd`

`temperature→set_lowestValue(newval)`

Modifie la mémoire de valeur minimale observée.

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

`temperature→set_ntcParameters(res25, beta)`

Configure les paramètres d'un thermistor NTC pour calculer correctement la température sur la base de la résistance mesurée.

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

`temperature→set_reportFrequency(newval)`

Modifie la fréquence de notification périodique des valeurs mesurées.

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

`temperature→set_resolution(newval)`

Modifie la résolution des valeurs physique mesurées.

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

`temperature→set_sensorType(newval)`

Modifie le type de capteur utilisé par le module.

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

`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.

`cmd` `cpp` `vb` `cs` `java` `py` `php` `ts` `dnp` `ppy`

temperature→set_unit(newval)

Modifie l'unité dans laquelle la température mesurée est exprimée.

cmd cpp vb cs java py php ts dnp ppy

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.

cpp vb cs java py php ts

temperature→startDataLogger()

Démarre l'enregistreur de données du module.

cmd cpp vb cs java py php ts dnp ppy

temperature→stopDataLogger()

Arrête l'enregistreur de données du module.

cmd cpp vb cs java py php ts dnp ppy

temperature→unmuteValueCallbacks()

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

cpp vb cs java py php ts dnp ppy cmd

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.

ts

24.7. La classe YDataLogger

Interface de contrôle de l'enregistreur de données, présent sur la plupart des capteurs Yoctopuce.

La plupart des capteurs Yoctopuce sont équipés d'une mémoire non-volatile. Elle permet de mémoriser les données mesurées d'une manière autonome, sans nécessiter le suivi permanent d'un ordinateur. La classe `YDataLogger` contrôle les paramètres globaux de cet enregistreur de données. Le contrôle de l'enregistrement (start / stop) et la récupération des données se fait au niveau des objets qui gèrent les senseurs.

Pour utiliser les fonctions décrites ici, vous devez inclure:

cpp	<code>#include "yocto_module.h"</code>
vb	<code>yocto_module.vb</code>
cs	<code>yocto_module.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YDataLogger;</code>
py	<code>from yocto_module import *</code>
php	<code>require_once('yocto_module.php');</code>
ts	<code>in HTML: import { YDataLogger } from '../dist/esm/yocto_module.js'; in Node.js: import { YDataLogger } from 'yoctolib-cjs/yocto_module.js';</code>
dnp	<code>import YoctoProxyAPI.YDataLoggerProxy</code>
cp	<code>#include "yocto_module_proxy.h"</code>
ppy	<code>from yoctolib.yocto_module import *</code>

Fonction globales

`YDataLogger.FindDataLogger(func)`

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

cpp vb cs java py php ts dnp ppy

`YDataLogger.FindDataLoggerInContext(yctx, func)`

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

java ts ppy

`YDataLogger.FirstDataLogger()`

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

cpp vb cs java py php ts ppy

`YDataLogger.FirstDataLoggerInContext(yctx)`

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

java ts ppy

`YDataLogger.GetSimilarFunctions()`

Enumère toutes les fonctions de type `DataLogger` disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (`hardwareId`).

dnp

Propriétés des objets `YDataLoggerProxy`

`datalogger→AdvertisedValue [lecture seule]`

Courte chaîne de caractères représentant l'état courant de la fonction.

dnp

`datalogger→AutoStart [modifiable]`

Mode d'activation automatique de l'enregistreur de données à la mise sous tension.

dnf

datalogger→BeaconDriven [modifiable]

Vrai si l'enregistreur de données est synchronisé avec la balise de localisation.

dnf

datalogger→FriendlyName [lecture seule]

Identifiant global de la fonction au format NOM_MODULE . NOM_FONCTION.

dnf

datalogger→FunctionId [lecture seule]

Identifiant matériel de l'enregistreur de données, sans référence au module.

dnf

datalogger→HardwareId [lecture seule]

Identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.

dnf

datalogger→IsOnline [lecture seule]

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

dnf

datalogger→LogicalName [modifiable]

Nom logique de la fonction.

dnf

datalogger→Recording [modifiable]

état d'activation de l'enregistreur de données.

dnf

datalogger→SerialNumber [lecture seule]

Numéro de série du module, préprogrammé en usine.

dnf

Méthodes des objets YDataLogger**datalogger→clearCache()**

Invalide le cache.

cpp vb cs java py php ts

datalogger→describe()

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

cpp vb cs java py php ts

datalogger→forgetAllDataStreams()

Efface tout l'historique des mesures de l'enregistreur de données.

cmd cpp vb cs java py php ts dnf ppy

datalogger→get_advertisedValue()

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

cpp vb cs java py php ts dnf ppy cmd

datalogger→get_autoStart()

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

cmd cpp vb cs java py php ts dnp ppy

datalogger→get_beaconDriven()

Retourne vrai si l'enregistreur de données est synchronisé avec la balise de localisation.

cmd cpp vb cs java py php ts dnp ppy

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.

cmd cpp vb cs java py php ts dnp ppy

datalogger→get_dataSets()

Retourne une liste d'objets `YDataSet` permettant de récupérer toutes les mesures stockées par l'enregistreur de données.

cmd cpp vb cs java py php ts dnp

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.

cpp vb cs java py php ts

datalogger→get_errorType()

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

cpp vb cs java py php ts

datalogger→get_friendlyName()

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

cpp cs java py php ts dnp

datalogger→get_functionDescriptor()

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

cpp vb cs java py php ts

datalogger→get_functionId()

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

cpp vb cs java py php ts dnp ppy

datalogger→get_hardwareId()

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

cpp vb cs java py php ts dnp ppy

datalogger→get_logicalName()

Retourne le nom logique de l'enregistreur de données.

cpp vb cs java py php ts dnp ppy cmd

datalogger→get_module()

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

cpp vb cs java py php ts dnp

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.

cmd cpp vb cs java py php ts dnp ppy

datalogger→get_serialNumber()

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

cpp vb cs java py php ts dnp ppy cmd

datalogger→get_timeUTC()

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

cmd cpp vb cs java py php ts dnp ppy

datalogger→get_usage()

Retourne le pourcentage d'utilisation de la mémoire d'enregistrement.

cmd cpp vb cs java py php ts dnp ppy

datalogger→get_userData()

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

cpp vb cs java py php ts

datalogger→isOnline()

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

cpp vb cs java py php ts dnp

datalogger→isOnline_async(callback, context)

Vérifie si le module hébergeant l'enregistreur de données est joignable, sans déclencher d'erreur.

datalogger→isReadOnly()

Indique si la fonction est en lecture seule.

cpp vb cs java py php ts dnp cmd

datalogger→load(msValidity)

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

cpp vb cs java py php ts

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.

cpp vb cs java py php ts dnp ppy

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.

cpp vb cs java py php ts dnp ppy cmd

datalogger→nextDataLogger()

Continue l'énumération des enregistreurs de données commencée à l'aide de `yFirstDataLogger()`. Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les enregistreurs de données sont retournés.

cpp vb cs java py php ts ppy

datalogger→registerValueCallback(callback)

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

cpp vb cs java py php ts ppy

datalogger→set_autoStart(newval)

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

cmd cpp vb cs java py php ts dnp ppy

datalogger→set_beaconDriven(newval)

Modifie le mode de synchronisation de l'enregistreur de données .

cmd cpp vb cs java py php ts dnp ppy

datalogger→set_logicalName(newval)

Modifie le nom logique de l'enregistreur de données.

cpp vb cs java py php ts dnp ppy cmd

datalogger→set_recording(newval)

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

cmd cpp vb cs java py php ts dnp ppy

datalogger→set_timeUTC(newval)

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

cmd cpp vb cs java py php ts dnp ppy

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.

cpp vb cs java py php ts

datalogger→unmuteValueCallbacks()

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

cpp vb cs java py php ts dnp ppy cmd

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.

ts

24.8. La classe YDataSet

Séquence de données enregistrées par le datalogger, obtenue par la méthode `sensor.get_recordedData()`

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 méthode `sensor.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 relativement 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:

cpp	<code>#include "yocto_module.h"</code>
vb	<code>yocto_module.vb</code>
cs	<code>yocto_module.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YDataSet;</code>
py	<code>from yocto_module import *</code>
php	<code>require_once('yocto_module.php');</code>
ts	<code>in HTML: import { YDataSet } from '../dist/esm/yocto_module.js'; in Node.js: import { YDataSet } from 'yoctolib-cjs/yocto_module.js';</code>
dnf	<code>import YoctoProxyAPI.YDataSetProxy</code>
cp	<code>#include "yocto_module_proxy.h"</code>
ppy	<code>from yoctolib.yocto_module import *</code>

Fonction globales

`YDataSet.Init(sensorName, startTime, endTime)`

Retourne un objet `YDataSet` permettant de charger les mesures d'un capteur donné par son nom ou identifiant matériel, pour un intervalle de temps spécifié.

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).

cpp vb cs java py php ts dnf ppy

`dataset→get_functionId()`

Retourne l'identifiant matériel de la fonction qui a effectué les mesures, sans référence au module.

cpp vb cs java py php ts dnf ppy

`dataset→get_hardwareId()`

Retourne l'identifiant matériel unique de la fonction qui a effectué les mesures, au format `SERIAL.FUNCTIONID`.

cpp vb cs java py php ts dnf ppy

`dataset→get_measures()`

Retourne toutes les mesures déjà disponibles pour le DataSet, sous forme d'une liste d'objets YMeasure.

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnp](#)

dataset→get_measuresAt(measure)

Retourne les mesures détaillées pour une mesure résumée précédemment retournée par get_preview().

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnp](#)

dataset→get_measuresAvgAt(index)

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par l'entrée spécifiée du résumé des mesures.

dataset→get_measuresEndTimeAt(index)

Retourne l'heure de fin de l'entrée spécifiée du résumé des mesures, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

dataset→get_measuresMaxAt(index)

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par l'entrée spécifiée du résumé des mesures.

dataset→get_measuresMinAt(index)

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par l'entrée spécifiée du résumé des mesures.

dataset→get_measuresRecordCount()

Retourne le nombre de mesures déjà chargées pour ce DataSet.

dataset→get_measuresStartTimeAt(index)

Retourne l'heure absolue de l'entrée spécifiée du résumé des mesures, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

dataset→get_preview()

Retourne une version résumée des mesures qui pourront être obtenues de ce YDataSet, sous forme d'une liste d'objets YMeasure.

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnp](#)

dataset→get_previewAvgAt(index)

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par l'entrée spécifiée du résumé des mesures.

dataset→get_previewEndTimeAt(index)

Retourne l'heure de fin de l'entrée spécifiée du résumé des mesures, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

dataset→get_previewMaxAt(index)

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par l'entrée spécifiée du résumé des mesures.

dataset→get_previewMinAt(index)

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par l'entrée spécifiée du résumé des mesures.

dataset→get_previewRecordCount()

Retourne le nombre d'entrées dans la version résumée des mesures qui pourront être obtenues dans ce YDataSet.

dataset→get_previewStartTimeAt(index)

Retourne l'heure absolue de l'entrée spécifiée du résumé des mesures, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

dataset→get_progress()

Retourne l'état d'avancement du chargement des données, sur une échelle de 0 à 100.

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnp](#) [ppy](#)**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).

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnp](#) [ppy](#)**dataset→get_summary()**

Retourne un objet YMeasure résumant tout le YDataSet.

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnp](#) [ppy](#)**dataset→get_summaryAvg()**

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par ce DataSet.

dataset→get_summaryEndTime()

Retourne l'heure de fin de la dernière mesure du data set, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

dataset→get_summaryMax()

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par ce DataSet.

dataset→get_summaryMin()

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par ce DataSet.

dataset→get_summaryStartTime()

Retourne l'heure absolue de la première mesure du data set, sous forme du nombre de secondes depuis le 1er janvier 1970 UTC (date/heure au format Unix).

dataset→get_unit()

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

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnp](#) [ppy](#)**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.

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [dnp](#) [ppy](#)**dataset→loadMore_async(callback, context)**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

24.9. La classe YMeasure

Valeur mesurée, retournée en particulier par les méthodes de la classe YDataSet.

Les objets YMeasure sont utilisés dans l'interface de programmation Yoctopuce pour représenter une valeur observée à un moment donnée. Ces objets sont utilisés en particulier en conjonction avec la classe YDataSet, mais aussi par les notification périodique des capteurs configurées (voir `sensor.registerTimedReportCallback`).

Pour utiliser les fonctions décrites ici, vous devez inclure:

cpp	<code>#include "yocto_module.h"</code>
vb	<code>yocto_module.vb</code>
cs	<code>yocto_module.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YMeasure;</code>
py	<code>from yocto_module import *</code>
php	<code>require_once('yocto_module.php');</code>
ts	<code>in HTML: import { YMeasure } from '../dist/esm/yocto_module.js'; in Node.js: import { YMeasure } from 'yoctolib-cjs/yocto_module.js';</code>
ppy	<code>from yoctolib.yocto_module import *</code>

Méthodes des objets YMeasure

`measure→get_averageValue()`

Retourne la valeur moyenne observée durant l'intervalle de temps couvert par la mesure.

cpp vb cs java py php ts ppy

`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).

cpp vb cs java py php ts ppy

`measure→get_maxValue()`

Retourne la plus grande valeur observée durant l'intervalle de temps couvert par la mesure.

cpp vb cs java py php ts ppy

`measure→get_minValue()`

Retourne la plus petite valeur observée durant l'intervalle de temps couvert par la mesure.

cpp vb cs java py php ts ppy

`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).

cpp vb cs java py php ts ppy

24.10. La classe YDataStream

Séquence de données enregistrées brute

Les objets `DataStream` correspondent aux séquences de mesures enregistrées brutes, directement telles qu'obtenues par l'enregistreur de données présent dans les senseurs de Yoctopuce.

Dans la plupart des cas, il n'est pas nécessaire d'utiliser les objets `DataStream`, car les objets `YDataSet` (retournés par la méthode `get_recordedData()` des senseurs et la méthode `get_dataSets()` du `DataLogger`) fournissent une interface plus pratique.

Pour utiliser les fonctions décrites ici, vous devez inclure:

cpp	<code>#include "yocto_module.h"</code>
vb	<code>yocto_module.vb</code>
cs	<code>yocto_module.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YDataStream;</code>
py	<code>from yocto_module import *</code>
php	<code>require_once('yocto_module.php');</code>
ts	<code>in HTML: import { YDataStream } from '../dist/esm/yocto_module.js'; in Node.js: import { YDataStream } from 'yoctolib-cjs/yocto_module.js';</code>
ppy	<code>from yoctolib.yocto_module import *</code>

Méthodes des objets YDataStream

`datastream→get_averageValue()`

Retourne la moyenne des valeurs observées durant cette séquence.

cpp vb cs java py php ts ppy

`datastream→get_columnCount()`

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

cpp vb cs java py php ts ppy

`datastream→get_columnNames()`

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

cpp vb cs java py php ts

`datastream→get_data(row, col)`

Retourne une mesure unique de la séquence, spécifiée par l'index de l'enregistrement (ligne) et de la mesure (colonne).

cpp vb cs java py php ts ppy

`datastream→get_dataRows()`

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

cpp vb cs java py php ts

`datastream→get_dataSamplesIntervalMs()`

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

cpp vb cs java py php ts ppy

`datastream→get_maxValue()`

Retourne la plus grande valeur observée durant cette séquence.

cpp vb cs java py php ts ppy

datastream→get_minValue()

Retourne la plus petite valeur observée durant cette séquence.

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [ppy](#)

datastream→get_realStartTimeUTC()

Retourne l'heure absolue du début de la séquence de données, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [ppy](#)

datastream→get_rowCount()

Retourne le nombre d'enregistrement contenus dans la séquence.

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [ppy](#)

datastream→get_runIndex()

Retourne le numéro de Run de la séquence de données.

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [ppy](#)

datastream→get_startTime()

Retourne le temps de départ relatif de la séquence (en secondes).

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [ppy](#)

datastream→get_startTimeUTC()

Retourne l'heure absolue du début de la séquence de données, sous forme du nombre de secondes depuis le 1er janvier 1970 (date/heure au format Unix).

[cpp](#) [vb](#) [cs](#) [java](#) [py](#) [php](#) [ts](#) [ppy](#)

25. Problèmes courants

25.1. Par où commencer ?

Si c'est la première fois que vous utilisez un module Yoctopuce et ne savez pas trop par où commencer, allez donc jeter un coup d'œil sur le blog de Yoctopuce. Il y a une section dédiée aux débutants ¹.

25.2. Linux et USB

Pour fonctionner correctement sous Linux, la librairie a besoin d'avoir accès en écriture à tous les périphériques USB Yoctopuce. Or, par défaut, sous Linux les droits d'accès des utilisateurs non-root à USB sont limités à la lecture. Afin d'éviter de devoir lancer les 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 de VirtualHub² 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"
```

25.3. 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.

25.4. Les exemples de programmation n'ont pas l'air de marcher

La plupart des exemples de programmation de l'API Yoctopuce sont des programmes en ligne de commande et ont besoin de quelques paramètres pour fonctionner. Vous devez les lancer depuis l'invite de commande de votre système d'exploitation ou configurer votre IDE pour qu'il passe les paramètres corrects au programme ³.

25.5. Module alimenté mais invisible pour l'OS

Si votre Yocto-MaxiBridge-C est branché par USB et que sa LED bleue s'allume, mais que le module n'est pas vu par le système d'exploitation, vérifiez que vous utilisez bien un vrai câble USB avec les fils pour les données, et non pas un câble de charge. Les câbles de charge n'ont que les fils d'alimentation.

25.6. Another process named xxx is already using yAPI

Si lors de l'initialisation de l'API Yoctopuce, vous obtenez le message d'erreur "*Another process named xxx is already using yAPI*", cela signifie qu'une autre application est déjà en train d'utiliser les modules Yoctopuce USB. Sur une même machine, un seul processus à la fois peut accéder aux modules Yoctopuce par USB. Cette limitation peut facilement être contournée en utilisant un VirtualHub et le mode réseau ⁴.

³ voir: <http://www.yoctopuce.com/FR/article/a-propos-des-programmes-d-exemples>

⁴ voir: <http://www.yoctopuce.com/FR/article/message-d-erreur-another-process-is-already-using-yapi>

25.7. Déconnexions, comportement erratique

Si votre Yocto-MaxiBridge-C 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}.

25.8. Le module ne marche plus après une mise à jour ratée

Si une mise à jour du firmware de votre Yocto-MaxiBridge-C échoue, il est possible que le module ne soit plus en état de fonctionner. Si c'est le cas, branchez votre module en maintenant le bouton Yocto-Bouton pressé. La Yocto-LED devrait s'allumer en haute intensité et rester fixe. Relâchez le bouton. Votre Yocto-MaxiBridge-C devrait alors apparaître dans le bas de l'interface du virtualHub comme un module attendant une mise à jour de firmware. Cette mise à jour aura aussi pour effet de réinitialiser le module à sa configuration d'usine.

25.9. RegisterHub d'une instance de VirtualHub déconnecte la précédente

Si lorsque vous faites un `YAPI.RegisterHub` de VirtualHub et que la connexion avec un autre VirtualHub précédemment enregistré tombe, vérifiez que les machines qui hébergent ces VirtualHubs ont bien un *hostname* différent. Ce cas de figure est très courant avec les machines dont le système d'exploitation est installé avec une image monolithique, comme les Raspberry Pi par exemple. L'API Yoctopuce utilise les numéros de série Yoctopuce pour communiquer et le numéro de série d'un VirtualHub est créé à la volée à partir du *hostname* de la machine qui l'héberge.

25.10. Commandes ignorées

Si vous avez l'impression que des commandes envoyées à un module Yoctopuce sont ignorées, typiquement lorsque vous avez écrit un programme qui sert à configurer ce module Yoctopuce et qui envoie donc beaucoup de commandes, vérifiez que vous avez bien mis un `YAPI.FreeAPI()` à la fin du programme. Les commandes sont envoyées aux modules de manière asynchrone grâce à un processus qui tourne en arrière plan. Lorsque le programme se termine, ce processus est tué, même s'il n'a pas eu le temps de tout envoyer. En revanche `API.FreeAPI()` attend que la file d'attente des commandes à envoyer soit vide avant de libérer les ressources utilisées par l'API et rendre la main.

25.11. Module endommagé

Yoctopuce s'efforce de réduire la production de déchets électroniques. Si vous avez l'impression que votre Yocto-MaxiBridge-C 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 vous voulez vraiment vous débarrasser de votre Yocto-MaxiBridge-C, ne le jetez pas à la poubelle, mais ramenez-le à l'un des points de collecte proposé dans votre région afin qu'il soit envoyé à un centre de recyclage ou de traitement spécialisé.

⁵ voir: <http://www.yoctopuce.com/FR/article/cables-usb-la-taille-compte>

⁶ voir: <http://www.yoctopuce.com/FR/article/combien-de-capteurs-usb-peut-on-connecter>

26. Caractéristiques

Vous trouverez résumées ci-dessous les principales caractéristiques techniques de votre module Yocto-MaxiBridge-C

Identifiant produit	WMBRDG1C
Révision matérielle [†]	
Connecteur USB	USB-C
Largeur	58.3 mm
Longueur	50 mm
Poids	15 g
Canaux	4
Chipset	Texas ADS1263
Fréquence de rafraîchissement	2 ou 7 Hz
Précision (typ.)	10 nV/V
Tension d'excitation	4.7 V AC/DC
Classe de protection selon IEC 61140	classe III
Temp. de fonctionnement normale	5...40 °C
Temp. de fonctionnement étendue [‡]	-30...85 °C
Conformité RoHS	RoHS III (2011/65/UE+2015/863)
USB Vendor ID	0x24E0
USB Device ID	0x00C2
Boîtier recommandé	YoctoBox-MaxiIO-Transp
Code tarifaire harmonisé	9032.9000
Câbles et boîtiers	disponibles séparément
Fabriqué en	Suisse

[†] Ces spécifications correspondent à la révision matérielle actuelle du produit. Les spécifications des versions antérieures peuvent être inférieures.

[‡] La plage de température étendue est définie d'après les spécifications des composants et testée sur une durée limitée (1h). En cas d'utilisation prolongée hors de la plage de température standard, il est recommandé procéder à des tests extensifs avant la mise en production.



A4

Scale
2:1
Echelle