

Yocto-MaxiDisplay-G

Mode d'emploi

Table des matières

1. Introduction	1
1.1. Informations de sécurité	2
1.2. Conditions environnementales	3
2. Présentation	5
2.1. Les éléments communs	5
2.2. Les éléments spécifiques	6
2.3. Accessoires optionnels	7
3. Principes de fonctionnement	9
3.1. Processeur et mémoire embarqués	9
3.2. Orientation	9
3.3. Système de couche	9
3.4. Routines graphiques	10
3.5. Affichage de texte	11
3.6. Format des fichiers de fontes	12
3.7. Séquences et animations	13
3.8. Optimisations	14
4. Le système de fichiers embarqué	15
4.1. Utilisation	15
4.2. Limitations	16
5. Premiers pas	17
5.1. Prérequis	17
5.2. Test de la connectivité USB	18
5.3. Localisation	19
5.4. Test du module	19
5.5. Configuration	20
6. Montage et connectique	23
6.1. Fixation	23
6.2. Contraintes d'alimentation par USB	24

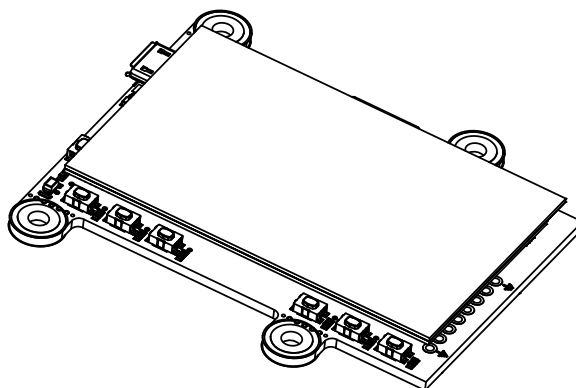
7. Programmation, concepts généraux	27
7.1. Paradigme de programmation	27
7.2. Le module Yocto-MaxiDisplay-G	29
7.3. Module	30
7.4. Display	31
7.5. AnButton	32
7.6. Files	34
7.7. Quelle interface: Native, DLL ou Service?	34
7.8. Programmation, par où commencer?	37
8. Utilisation du Yocto-MaxiDisplay-G en ligne de commande	39
8.1. Installation	39
8.2. Utilisation: description générale	39
8.3. Contrôle de la fonction Display	40
8.4. Contrôle de la partie module	41
8.5. Limitations	41
9. Utilisation du Yocto-MaxiDisplay-G en Python	43
9.1. Fichiers sources	43
9.2. Librairie dynamique	43
9.3. Contrôle de la fonction Display	43
9.4. Contrôle de la partie module	45
9.5. Gestion des erreurs	46
10. Utilisation du Yocto-MaxiDisplay-G en C++	49
10.1. Contrôle de la fonction Display	49
10.2. Contrôle de la partie module	51
10.3. Gestion des erreurs	53
10.4. Intégration de la librairie Yoctopuce en C++	54
11. Utilisation du Yocto-MaxiDisplay-G en C#	57
11.1. Installation	57
11.2. Utilisation l'API yoctopuce dans un projet Visual C#	57
11.3. Contrôle de la fonction Display	58
11.4. Contrôle de la partie module	59
11.5. Gestion des erreurs	62
12. Utilisation du Yocto-MaxiDisplay-G avec LabVIEW	63
12.1. Architecture	63
12.2. Compatibilité	64
12.3. Installation	64
12.4. Présentation des VIs Yoctopuce	69
12.5. Fonctionnement et utilisation des VIs	72
12.6. Utilisation des objets	74
12.7. Gestion du datalogger	76
12.8. Énumération de fonctions	77
12.9. Un mot sur les performances	78
12.10. Un exemple complet de programme LabVIEW	78
12.11. Différences avec les autres API Yoctopuce	79
13. Utilisation du Yocto-MaxiDisplay-G en Java	81

13.1. Préparation	81
13.2. Contrôle de la fonction <i>Display</i>	81
13.3. Contrôle de la partie module	82
13.4. Gestion des erreurs	85
14. Utilisation du Yocto-MaxiDisplay-G avec Android	87
14.1. Accès Natif et Virtual Hub.	87
14.2. Préparation	87
14.3. Compatibilité	87
14.4. Activer le port USB sous Android	88
14.5. Contrôle de la fonction <i>Display</i>	90
14.6. Contrôle de la partie module	91
14.7. Gestion des erreurs	95
15. Utilisation du Yocto-MaxiDisplay-G en TypeScript	97
15.1. Utiliser la librairie Yoctopuce pour TypeScript	98
15.2. Petit rappel sur les fonctions asynchrones en JavaScript	98
15.3. Contrôle de la fonction <i>Display</i>	99
15.4. Contrôle de la partie module	102
15.5. Gestion des erreurs	104
16. Utilisation du Yocto-MaxiDisplay-G en JavaScript / EcmaScript	105
16.1. Fonctions bloquantes et fonctions asynchrones en JavaScript	106
16.2. Utiliser la librairie Yoctopuce pour JavaScript / EcmaScript 2017	107
16.3. Contrôle de la fonction <i>Display</i>	109
16.4. Contrôle de la partie module	111
16.5. Gestion des erreurs	113
17. Utilisation du Yocto-MaxiDisplay-G en PHP	115
17.1. Préparation	115
17.2. Contrôle de la fonction <i>Display</i>	115
17.3. Contrôle de la partie module	117
17.4. API par callback HTTP et filtres NAT	119
17.5. Gestion des erreurs	123
18. Utilisation du Yocto-MaxiDisplay-G en VisualBasic .NET	125
18.1. Installation	125
18.2. Utilisation l'API yoctopuce dans un projet Visual Basic	125
18.3. Contrôle de la fonction <i>Display</i>	126
18.4. Contrôle de la partie module	127
18.5. Gestion des erreurs	129
19. Utilisation du Yocto-MaxiDisplay-G en Delphi	131
19.1. Préparation	131
19.2. Contrôle de la fonction <i>Display</i>	131
19.3. Contrôle de la partie module	133
19.4. Gestion des erreurs	135
20. Utilisation du Yocto-MaxiDisplay-G avec Universal Windows Platform	137
20.1. Fonctions bloquantes et fonctions asynchrones	137
20.2. Installation	138

20.3. Utilisation l'API Yoctopuce dans un projet Visual Studio	138
20.4. Contrôle de la fonction Display	139
20.5. Un exemple concret	140
20.6. Contrôle de la partie module	140
20.7. Gestion des erreurs	142
21. Utilisation du Yocto-MaxiDisplay-G en Objective-	145
21.1. Contrôle de la fonction Display	145
21.2. Contrôle de la partie module	147
21.3. Gestion des erreurs	149
22. Utilisation avec des langages non supportés	151
22.1. Utilisation en ligne de commande	151
22.2. Assembly .NET	151
22.3. Virtual Hub et HTTP GET	153
22.4. Utilisation des librairies dynamiques	155
22.5. Port de la librairie haut niveau	158
23. Programmation avancée	159
23.1. Programmation par événements	159
24. Référence de l'API de haut niveau	161
24.1. La classe YAPI	162
24.2. La classe YModule	203
24.3. La classe YDisplay	280
24.4. La classe YDisplayLayer	358
24.5. La classe YAnButton	392
24.6. La classe YFiles	460
25. Problèmes courants	513
25.1. Par où commencer ?	513
25.2. Linux et USB	513
25.3. Plateformes ARM: HF et EL	514
25.4. Les exemples de programmation n'ont pas l'air de marcher	514
25.5. Module alimenté mais invisible pour l'OS	514
25.6. Another process named xxx is already using yAPI	514
25.7. Déconnexions, comportement erratique	515
25.8. RegisterHub d'un VirtualHub déconnecte le précédent	515
25.9. Commandes ignorées	515
25.10. Module endommagé	515
26. Caractéristiques	517

1. Introduction

Le Yocto-MaxiDisplay-G est un module électronique de 66x58mm qui permet de piloter un écran OLED monochrome de 128x64px ainsi que 6 canaux qui permettant de mesurer l'état d'interrupteurs, boutons poussoir, ou encore de potentiomètres. Cet écran vous permettra d'afficher facilement quelques informations très lisibles depuis une machine qui n'est normalement pas munie d'un moniteur.



Le module Yocto-MaxiDisplay-G

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

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

1.1. Informations de sécurité

Le Yocto-MaxiDisplay-G 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

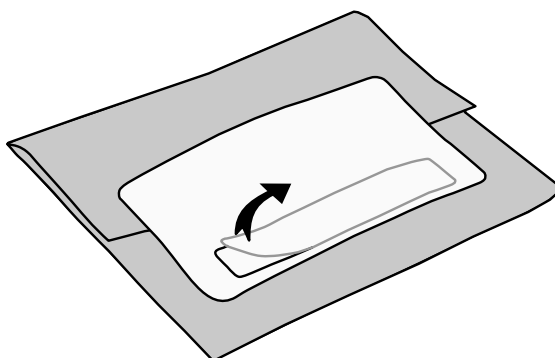
Le Yocto-MaxiDisplay-G ne doit pas être utilisé sans boîtier de protection, en raison des composants électriques à nu. Pour une sécurité optimale, il devrait être mis dans un boîtier non métallique, non-inflammable, résistant à un choc de 5 J, par exemple en polycarbonate (LEXAN ou autre) d'indice de 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-MaxiDisplay-G, 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 extérieure du boîtier. L'étiquette est résistante à l'humidité et au frottement usuel qui peut survenir durant un entretien normal.



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

Applications

La norme de sécurité vérifiée correspond aux instruments de laboratoire, pour le contrôle de procédés industriels, ou pour des applications similaires en milieu résidentiel ou commercial. Si vous comptez l'utiliser le Yocto-MaxiDisplay-G 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-MaxiDisplay-G 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-MaxiDisplay-G n'est *pas* certifié pour utilisation dans les zones dangereuses, ni pour les environnements explosifs. Les conditions environnementales assignées sont décrites ci-dessous.

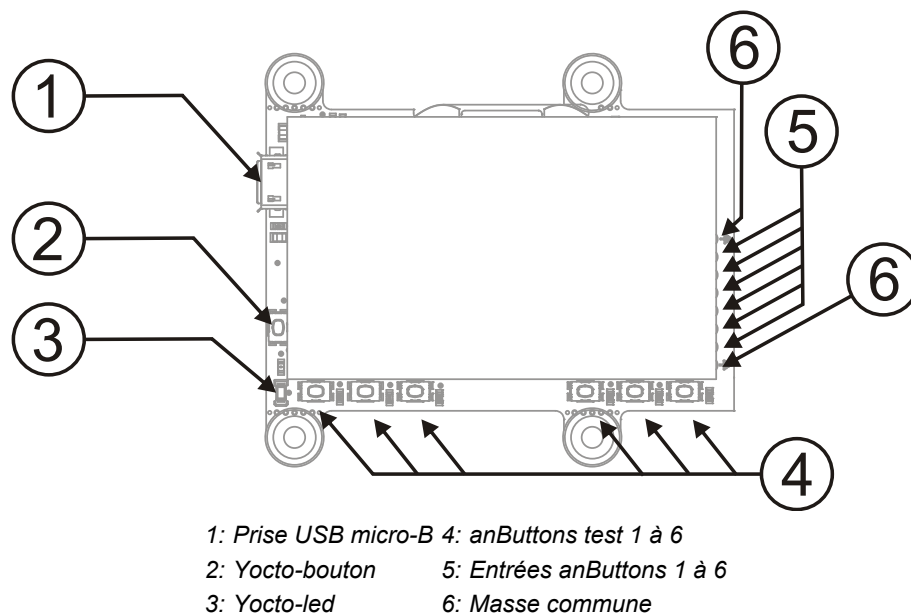
1.2. Conditions environnementales

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

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

La plage de température de fonctionnement étendue du Yocto-MaxiDisplay-G est -25...70°C. Cette plage de température a été déterminée en fonction des recommandations officielles des fabricants des composants utilisés dans le Yocto-MaxiDisplay-G, et par des tests de durée limitée (1h) dans les conditions extrêmes, en environnement contrôlé. Si vous envisagez d'utiliser le Yocto-MaxiDisplay-G 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



2.1. Les éléments communs

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

Le connecteur USB

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

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

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

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

Le Yocto-bouton

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

La Yocto-Led

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

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

La Yocto-Led a une troisième fonctionnalité moins plaisante: lorsque ce logiciel interne qui contrôle le module rencontre une erreur fatale, elle se met à flasher SOS en 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-MaxiDisplay-G ce numéro commence par YD128G64. Le module peut être piloté par logiciel en utilisant ce numéro de série. Ce numéro de série ne peut pas être changé.

Le nom logique

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

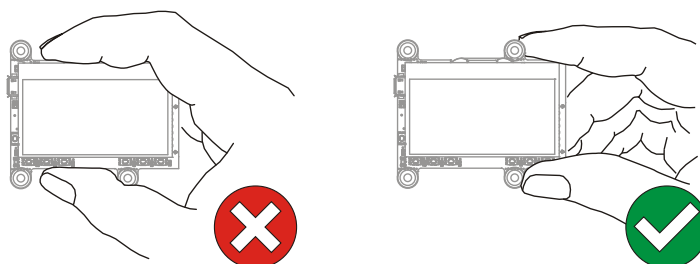
L'écran

L'écran est un écran OLED fabriqué par WiseChip sous la référence UG-2864ASGGG14. Etant en verre, il est assez fragile, ne laissez pas tomber votre Yocto-MaxiDisplay-G. Lorsque vous installez votre Yocto-MaxiDisplay-G veillez à ce que l'écran ne soit soumis à aucune contrainte mécanique. De plus la nappe qui sort de l'écran est particulièrement fragile au niveau de la jonction à l'écran, veillez

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

² support@yoctopuce.com

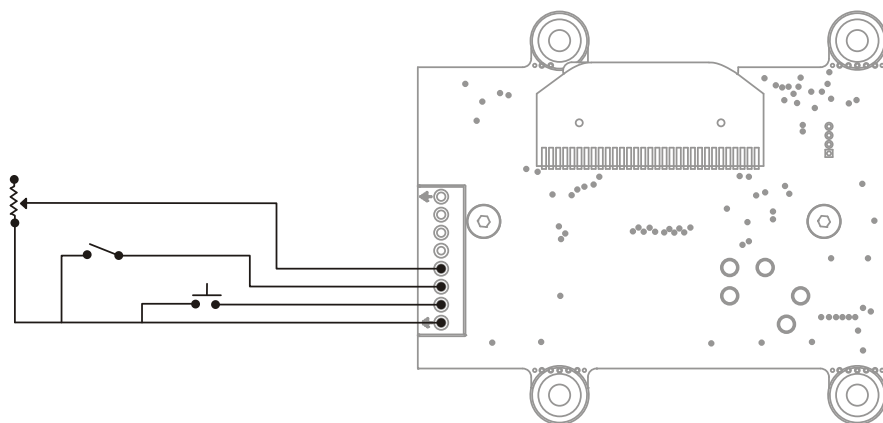
à ce qu'elle ne soit soumise à aucune contrainte mécanique, lorsque vous manipulez le Yocto-MaxiDisplay-G, veillez à ne pas appuyer sur la nappe.



N'appuyez pas sur la nappe quand vous manipulez le Yocto-MaxiDisplay-G

Les entrées 1 à 6

Le module Yocto-MaxiDisplay-G dispose de 6 entrées permettant de mesurer l'état de composants résistifs (interrupteurs, boutons poussoir, potentiomètres etc.). Ces entrées sont en masse commune, cela signifie que chaque interrupteur / bouton poussoir / potentiomètre doit être relié à la fois à l'entrée correspondante et à la masse. Vous pouvez utiliser n'importe quelle valeur de potentiomètre entre 1K Ω et 200 K Ω .



Câblage d'un potentiomètre, d'un interrupteur et d'un bouton poussoir, en masse commune.

Vous voudrez probablement souder un connecteur à l'endroit prévu à cet effet. Pour accéder au pads, enlevez les vis situées au dos de votre Yocto-MaxiDisplay-G, écarterez délicatement l'écran et soudez votre connecteur. Remplacez l'écran ensuite.

Potentiomètres et calibration

Ce module vous permet d'utiliser une grande plage de valeurs de potentiomètres. Mais pour qu'il soit capable de vous donner des mesures cohérentes pour le modèle que vous utiliserez, vous devrez calibrer les canaux correspondant. Cela peut être fait très simplement grâce à l'interface de configuration. Il n'est pas nécessaire de faire une calibration si vous utilisez de simple interrupteurs ou encore des boutons poussoirs.

Les boutons poussoirs de test

Chaque canal dispose d'un petit bouton poussoir qui permet de fermer artificiellement le circuit correspondant. Ce qui vous aidera probablement à déboguer vos projets.

2.3. Accessoires optionnels

Les accessoires ci-dessous ne sont pas nécessaires à l'utilisation du module Yocto-MaxiDisplay-G, 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-MaxiDisplay-G à un support, vous pouvez placer des petites vis de 3mm avec une tête de 8mm au maximum dans les trous prévus ad-hoc. Il est conseillé de les visser dans des entretoises filetées, que vous pourrez fixer sur le support. Vous trouverez plus de détail à ce sujet dans le chapitre concernant le montage et la connectique.

Micro-hub USB

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

YoctoHub-Ethernet, YoctoHub-Wireless and YoctoHub-GSM

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

Connecteurs 1.27mm (ou 1.25mm)

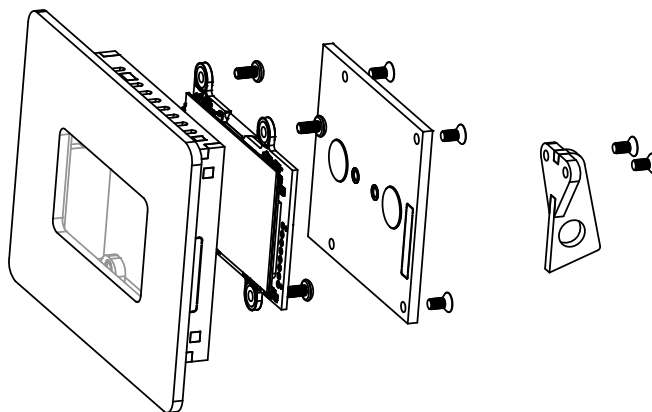
Si vous désirez raccorder le module Yocto-MaxiDisplay-G à un Micro-hub USB ou à un YoctoHub en évitant l'encombrement d'un vrai câble USB, vous pouvez utiliser les 4 pads au pas 1.27mm juste derrière le connecteur USB. Vous avez alors deux possibilités.

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-MaxiDisplay-G.

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-MaxiDisplay-G a été conçu pour pouvoir être installé tel quel dans votre projet. Néanmoins, Yoctopuce commercialise des boîtiers spécialement conçus pour les modules Yoctopuce. Le boîtier recommandé pour votre Yocto-MaxiDisplay-G est le modèle YoctoBox-MaxiDisplay. Il a un petit pied amovible qui lui permet de tenir debout, il dispose aussi d'aimants intégrés puissants qui lui permettent de tenir sur des surfaces ferromagnétiques. Vous trouverez plus d'informations à propos de ce boîtier sur le site de Yoctopuce³.



Vous pouvez installer votre Yocto-MaxiDisplay-G dans un boîtier optionnel.

³ <http://www.yoctopuce.com/EN/products/category/enclosures>

3. Principes de fonctionnement

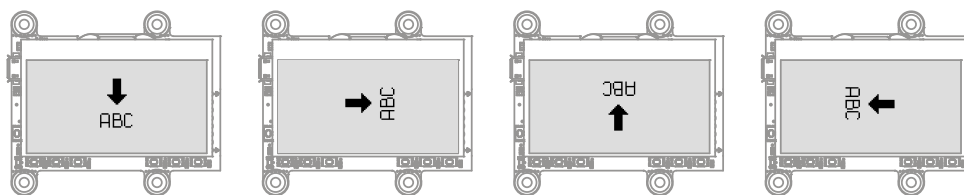
3.1. Processeur et mémoire embarqués

Comme tous les modules Yoctopuce, votre Yocto-MaxiDisplay-G dispose d'un processeur embarqué qui lui permet d'effectuer des opérations relativement complexes en toute transparence. Ainsi, pour tracer une ligne, l'ordinateur hôte n'a qu'à envoyer la commande *tracer une ligne* au Yocto-MaxiDisplay-G, et ne plus s'occuper de rien, le reste est géré par le processeur du Yocto-MaxiDisplay-G. A ce titre le Yocto-MaxiDisplay-G fonctionne un peu comme un accélérateur graphique où les tâches graphiques sont effectuées par un processeur dédié afin de laisser le processeur principal vaquer à d'autres tâches.

Votre Yocto-MaxiDisplay-G dispose aussi d'un petit système de fichiers qui vous permettra de stocker quelques graphismes, fontes et autres animations.

3.2. Orientation

Afin de faciliter son installation matérielle, votre Yocto-MaxiDisplay-G peut fonctionner selon quatre orientations différentes. Il suffit de configurer un paramètre pour indiquer la position du connecteur USB (*left, up, right, down*) par rapport à l'affichage, et l'écran tournera le contenu pour qu'il apparaisse du bon sens.

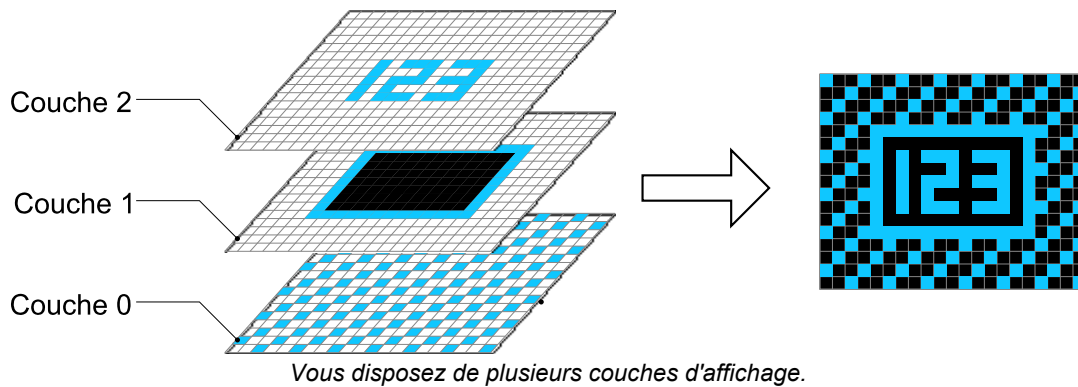


Effet des paramètres LEFT, UP, RIGHT et DOWN sur l'orientation de l'affichage.

Ce paramètre est persistant et peut être sauvé dans la mémoire flash du Yocto-MaxiDisplay-G

3.3. Système de couche

Votre Yocto-MaxiDisplay-G fonctionne selon un principe de couche superposées et indépendantes. Vous pouvez écrire et dessiner indépendamment dans chacune des 5 couches. Cela vous permet de simplifier et d'optimiser votre code d'affichage.



Vous pouvez cacher et rendre visible n'importe quelle couche. Vous pouvez déplacer latéralement ces couches qui sont légèrement plus grandes que la surface affichable (128x128), créant ainsi un effet de scrolling. Vous pouvez mettre à profit ce système de couches pour implémenter un système de *double buffering* ¹.

Chaque couche dispose de son propre contexte graphique: position du curseur, police courante, couleur courante etc... cela signifie que vous devrez régler ces paramètres pour chacune des couches avec lesquelles vous travaillerez. Mais cela signifie aussi que plusieurs processus différents peuvent interagir avec votre Yocto-MaxiDisplay-G sans risquer de problèmes de concurrence: il suffit qu'ils écrivent dans des couches différentes.

Les primitives permettant de manipuler directement les couches sont:

- clear
- hide
- unhide
- setLayerPosition
- reset
- swapLayerContent
- copyLayerContent

3.4. Routines graphiques

Votre Yocto-MaxiDisplay-G dispose des routines graphiques de base: lignes, rectangles, cercles, disques, affichage de texte etc. Toutes ces routines supportent le clipping: vous pouvez écrire à cheval sur la bordure d'une couche, la partie située dans la zone gérée sera prise en compte, et la partie située en dehors sera ignorée.

Pour effectuer des opérations plus complexes, ou simplement si vous le préférez, il est aussi possible d'utiliser la librairie graphique de votre choix pour dessiner sur un bitmap en mémoire dans l'ordinateur pilotant le Yocto-MaxiDisplay-G, et d'afficher ensuite ce bitmap d'un coup directement sur la couche graphique de votre choix. Cette opération est suffisamment rapide pour fabriquer des animations graphiques arbitraires en temps réel.

Les primitives graphiques de base sont:

- moveTo
- lineTo
- drawPixel
- drawRect
- drawBar
- drawCircle
- drawDisc
- drawBitmap
- drawImage

¹ http://en.wikipedia.org/wiki/Multiple_buffering

Couleurs

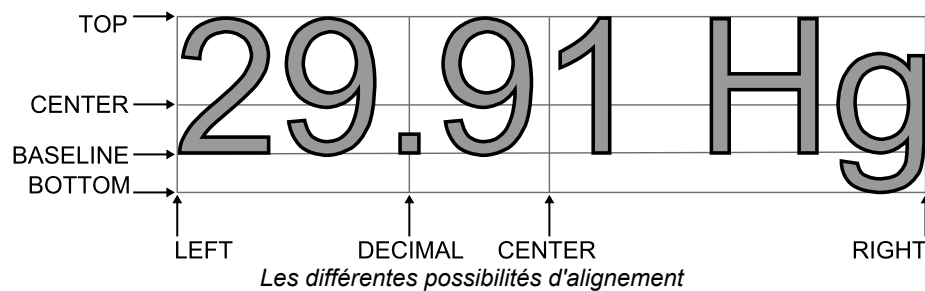
L'écran du Yocto-MaxiDisplay-G est purement monochrome. Vous ne pouvez donc pas afficher de niveaux de gris, ni bénéficier d'anti-aliasing. Vous pouvez dessiner en trois "couleurs": la couleur d'affichage de l'écran (que nous appellerons "blanc" dans cette documentation, même si il s'agit de bleu clair par exemple), en noir ou en transparent (gomme). Lorsque vous écrivez en transparent, les couches inférieures deviennent visibles. Notez que la couche 0 n'a pas de transparence. Écrire en transparent sur cette couche revient à écrire en noir. Cela a son importance lorsque vous intervertissez le contenu de deux couches.

Les primitives permettant de changer de couleur de dessin sont:

- selectGrayPen
- selectEraser

3.5. Affichage de texte

Vous pouvez afficher n'importe quel texte à une position arbitraire de l'écran. Le Yocto-MaxiDisplay-G dispose de quelques polices de caractères embarquées, mais vous pouvez créer les vôtres relativement facilement. Il n'est pas possible de connaître à l'avance la taille d'un texte, mais pour compenser de nombreux mode d'alignement de texte vous sont proposés. Vous pouvez aligner du texte à gauche à droite, centre, en fonction du point décimal, de la base line etc.



Les primitives de base permettant d'afficher un texte sont:

- selectFont
- drawText

Les polices de caractères préchargées sur le module sont:

- Small.yfm, qui fait 8 points de haut
- Medium.yfm qui fait 16 points de haut
- Large.yfm qui fait 32 points de haut
- 8x8.yfm à chasse fixe

Mode console

Il existe une autre méthode pour afficher du texte sur votre Yocto-MaxiDisplay-G: le mode console. La console est une zone rectangulaire dont la position est paramétrable. Les textes affichés dans cette console sont affichés à la manière d'un terminal, les retours à la ligne et le défilement sont gérés automatiquement. Chaque couche a une console intégrée. La taille par défaut de la console de chaque couche est initialisée à la taille de l'écran.

Les primitives permettant de gérer la console sont:

- clearConsole
- consoleOut
- setConsoleMargins
- setConsoleBackground
- setConsoleWordWrap

Internationalisation et caractères régionaux

Les fonctions affichant du texte supportent les caractères internationaux (lettres accentuées, alphabets spéciaux) pour les langues qui remplissent les caractéristiques suivantes:

- jeu de caractères sur 8 bits
- mots écrits de gauche à droite

Pour les jeux de caractères nécessitant plus de 8 bits (par exemple les polices chinoises) et les langages s'écrivant de droite à gauche, la seule solution consiste à construire une image bitmap dans l'ordinateur hôte à l'aide des fonctions du système, et de l'afficher avec la primitive `drawBitmap`.

Pour tous les autres langages, il suffit d'utiliser la même page de localisation pour la police de caractères et dans le programme pour que les caractères accentués s'affichent correctement. Les polices de caractères préchargées sur les modules correspondent à la locale `iso-8859-1` (aussi appelée `iso-latin-1` ou `Windows-1252`) permettant l'affichage des pays de l'ouest de l'Europe, mais vous pouvez facilement générer des polices équivalentes utilisant la locale de votre ordinateur à l'aide du petit utilitaire fourni dans la librairie Delphi, dans le répertoire `Examples\Display-font-generator`.

En pratique, pour les langages qui supportent intrinsèquement un encodage Unicode (16bit, UTF8 ou autre) comme Python, C#, VB ou Java, vous configurez dans l'API la page de localisation qui doit être utilisée pour ramener les chaînes de caractères à 8 bits. La valeur par défaut est `iso-8859-1`, puisque c'est celle qui correspond aux polices préchargées dans le module.

Pour les autres langages comme Delphi, C++ où PHP où c'est la page de localisation du fichier source qui détermine implicitement l'interprétation des chaînes de caractères immédiates, et où les conversions sont sous le contrôle direct du programmeur, vous n'avez qu'à vous assurer de passer à l'API de chaînes compatibles avec la police de caractères que vous utilisez sur votre écran. L'erreur la plus fréquente consiste à utiliser un format UTF-8 sans s'en rendre compte (parce que c'est le format utilisé par de nombreux éditeurs modernes) et d'omettre de convertir la chaîne en `iso-8859-*` avant de la passer en argument à la fonction `drawText` ou `consoleOut`. Si vous voyez s'afficher deux lettres bizarres à la place de chaque caractère accentué, c'est que vous êtes dans ce cas de figure.

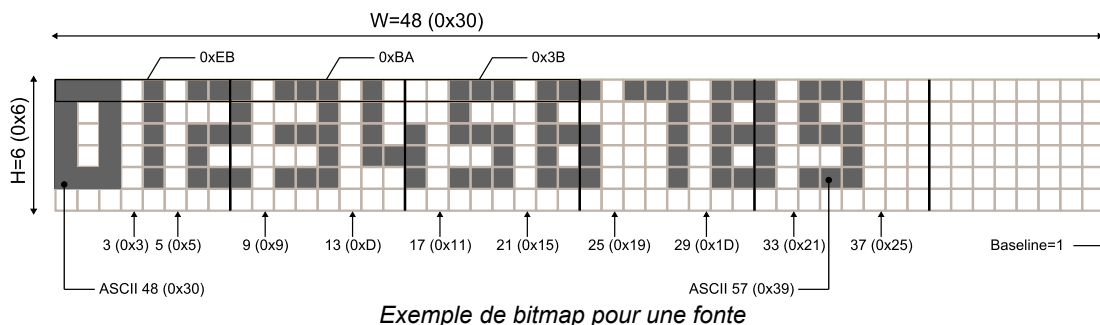
3.6. Format des fichiers de fontes

Votre Yocto-MaxiDisplay-G contient quelques fontes intégrées, mais il a été conçu pour que vous puissiez créer vos propres fontes aussi facilement que possible. Un fichier de fonte pour votre Yocto-MaxiDisplay-G est essentiellement un gros bitmap où tous les caractères sont dessinés dans l'ordre de leur code ASCII, les un derrière les autres. En plus du bitmap, ces fichiers incluent un header contenant quelques informations annexe ainsi qu'une liste des positions de la dernière colonne de chaque caractère. Le format est le suivant:

offset	type	Taille (bytes)	signification
0x00	U16	2	Signature ("YF" 0x4659, little endian)
0x02	U8	1	Version = 1
0x03	U8	1	Bits par pixel =1
0x04	U16	2	<i>W</i> la largeur du bitmap, little endian, (doit être un multiple de 16)
0x06	U8	1	<i>H</i> la hauteur du bitmap
0x09	U8	1	Base line (en partant du bas)
0x07	U8	1	Premier caractère défini
0x08	U8	1	Dernier caractère défini
0x0A	U16[]	2*N	coordonnées de la dernière colonne de chaque caractère (little endian)
0x0A +2*N	U8[]	H* W / 16	Données du bitmap

Exemple pratique

Imaginons que l'on souhaite définir une fonte minuscule 3x5 pixels pour les chiffres 0 à 9, le bitmap correspondrait à ceci.



Le fichier de fonte contiendra alors les données suivantes:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	59	46	01	01	30	00	06	01	30	39	03	00	05	00	09	00
00000010	0D	00	11	00	15	00	19	00	1D	00	21	00	25	00	EB	BA
00000020	3B	BB	B8	00	A8	8A	22	0A	A8	00	AB	BA	BB	8B	B8	00
00000030	AA	0B	8A	8A	88	00	EB	B8	BB	8B	B8	00	00	00	00	00
00000040	00	00														

Notez que la largeur du bitmap doit être un multiple de 16 pixels, et que la hauteur ne peut pas être plus grande que 255 pixels. Par ailleurs l'espacement entre deux caractères est encodé directement dans l'image. Vous n'êtes pas obligé de laisser un espace vide sous les caractères si vous ne comptez pas utiliser votre fonte en mode console.

Vous trouverez dans la librairie Delphi un petit utilitaire Windows² vous permettant de générer des fichiers de fontes à partir des fontes système.

3.7. Séquences et animations

Il est possible de pré-programmer des animations pour ensuite les rejouer en tâche de fond. Pour cela il suffit d'appeler la méthode *newSequence* de l'objet *Display*, puis de faire appel aux méthodes graphiques disponibles, il est possible s'insérer des temps d'attente avec *pauseSequence*. Une fois l'enregistrement de éléments de la séquence terminé, appeler *saveSequence*. La séquence sera alors sauvée dans le file system du Yocto-MaxiDisplay-G. Elle pourra être rejouée à volonté grâce à *playSequence*. Il est possible de créer des boucles en appelant *playSequence* à l'intérieur d'une séquence.

Vous trouverez dans les librairies un exemple de code³ illustrant le fonctionnement des séquences. Dès que cet exemple sera exécuté, l'écran se mettra à jouer la séquence indéfiniment.

Attention, les séquences modifient les paramètres (point courant, couleur courante etc..) des couches avec lesquelles elles travaillent. Si vous utilisez une séquence comme une animation en tâche de fond veillez à ne pas travailler avec les mêmes couches que celles utilisées par votre séquence.

Séquence de démarrage

A l'allumage votre Yocto-MaxiDisplay-G joue la séquence *yocto.seq* qui est préchargée dans le module, mais vous pouvez configurer votre module pour qu'il exécute une séquence animée de votre choix.

² Examples\Display-font-generator

³ Prog-Display-Sequences

3.8. Optimisations

Le Yocto-MaxiDisplay-G a beau disposer de son propre processeur et proposer de nombreuses routines graphiques, il reste néanmoins un système relativement lent comparé à un système d'affichage classique. Cette lenteur est essentiellement due au protocole HID utilisé pour que le Yocto-MaxiDisplay-G puisse être piloté sans drivers. Le taux de transfert entre l'écran et l'ordinateur est limité à 64Ko par seconde. Chaque requête prends environ 3 ms. Il existe cependant des techniques pour optimiser l'affichage.

Écriture dans les couche cachées

Les actions à effectuer sur couches visibles sont envoyées immédiatement à l'écran afin d'être exécutée au plus vite, en revanche les actions à effectuer sur les couches cachées sont bufferisées, permettant ainsi d'envoyer plusieurs commandes d'un coup. Il est donc beaucoup plus efficace d'écrire dans une couche cachée et de la rendre visible après coup, ce qui rend le *double buffering* très intéressant.

Double buffering

La technique appelée *double buffering* permet d'afficher des animations sans que les artefacts lié à la construction des graphismes ne soient visibles. Elle consiste à travailler sur deux couches, une visible et une seconde cachée. Les images sont construites dans la couche invisible et à la fin de chaque construction, le contenu des deux couches est permuté. Vous trouverez dans les librairies un exemple⁴ utilisant cette technique pour animer un flocon de Von Koch.

Utilisation de bitmap

A partir d'une certaine complexité graphique, il sera plus probablement plus efficace de calculer un bitmap directement sur la machine hôte et de l'envoyer à l'écran. Pour cela il suffit d'utiliser la fonction *drawBitmap*. Les données d'un bitmap sont encodées dans un tableau de bytes, lignes par lignes en partant du coin supérieur gauche. Dans chaque byte, le bit de point le plus fort représente le pixel le plus à gauche. Vous trouverez dans les librairies un exemple⁵ calculant un ensemble de Mandelbrot basé sur ce principe.

Voici le code en C permettant de dessiner un pixel aux coordonnées (x,y) dans un tableau de bytes représentant un bitmap de *w x h*

```
void putpixel(unsigned char *data, int x, int y)
{
    int bytesPerLine = (w + 7) >> 3;
    data[ (x >> 3) + (y * bytesPerLine) ] |= 128 >> (x & 7);
}
```

⁴ Prog-Display-DoubleBuffering

⁵ Prog-Display-DrawBitmap

4. Le système de fichiers embarqué

Votre Yocto-MaxiDisplay-G dispose d'un petit système de fichiers embarqué, qui permet de stocker des fichiers personnalisés utilisables par le Yocto-MaxiDisplay-G. Le système de fichier se manipule grâce à la librairie *yocto_files*.

4.1. Utilisation

Utilisation interactive avec le virtual hub

Le *Virtual Hub* fourni une interface sommaire pour manipuler le contenu du système de fichiers: cliquez simplement le bouton *configuration* correspondant à votre module dans l'interface du *Virtual Hub*, puis sur le bouton *manage files*. Les fichiers présents sont listés, et vous pouvez les visualiser, les effacer ou en ajouter (téléchargement).

En raison de sa petite taille, le système de fichiers ne possède pas de notion explicite de répertoire. Vous pouvez toutefois utiliser la barre oblique "/" à l'intérieur des noms de fichiers pour les *classer* comme si ils étaient dans des répertoires.

Utilisation programmée

Le système de fichiers s'utilise avec la librairie *yocto_files*. Les fonctions de bases sont disponibles:

- *upload* vous permet de créer un nouveau fichier sur le module, dont vous fournissez le contenu;
- *get_list* vous permet de connaître la liste de fichier présents sur le module, y compris la taille et le CRC32 du contenu;
- *download* vous permet de le récupérer dans une variable le contenu d'un fichier présent sur le module;
- *remove* permet d'effacer un fichier du module.
- *format* permet de réinitialiser le système de fichiers à un état vide, non fragmenté.

Un programme utilisant le système de fichier bien conçu devrait toujours commencer par s'assurer que les fichiers nécessaires à son fonctionnement sont présents sur le module, et si nécessaire les charger sur le module. Cela permet de gérer de manière transparente les mises à jour logicielles et le déploiement de l'application sur des nouveaux modules. Pour faciliter la détection des versions de fichiers présents sur le module, la méthode *get_list* retourne pour chaque fichier une signature sur 32 bit appelée CRC (Cyclic Redundancy Check), qui identifie de manière fiable le contenu du fichier. Ainsi, si le CRC du fichier correspond, il y a moins d'une chance sur 4 milliards que son contenu ne soit pas le bon. Vous pouvez même calculer dans votre programme par avance le CRC du contenu que vous désirez, et ainsi le vérifier sans avoir à transférer le fichier. La fonction CRC utilisée par le

système de fichier Yoctopuce est la même que celle d'Ethernet, Gzip, PNG, etc. Sa valeur caractéristique pour la chaîne de neuf caractères "123456789" est 0xCB43926.

Utilisation par HTTP

Les fichiers que vous avez chargés sur votre Yocto-MaxiDisplay-G sont accessibles par HTTP, à la racine du module (au même niveau que l'API REST). Cela permet de charger par exemple des pages d'interface HTML et Javascript personnalisées. Vous ne pouvez toutefois pas remplacer le contenu d'un fichier préchargé sur le module, mais seulement en ajouter des nouveaux.

4.2. Limitations

Le filesystem embarqué sur votre Yocto-MaxiDisplay-G a quelques limitations techniques:

- Son espace de stockage maximal est 3.5Mo, répartis en blocs permettant de stocker jusqu'à environ 800 fichiers
- L'effacement d'un fichier ne récupère pas nécessairement immédiatement la totalité de la place utilisée par le fichier. L'espace non libéré sera entièrement réutilisé si l'on recrée un fichier du même nom, mais pas forcément si l'on crée des fichiers utilisant chaque fois des noms différents. Pour cette raison, il n'est pas recommandé de générer automatiquement des fichiers avec des noms toujours différents.
- L'espace non libéré peut être entièrement récupéré avec la commande *format*, qui libère la totalité des fichiers.
- Chaque mise à jour du firmware provoque implicitement un formatage complet du filesystem.
- Comme toutes les mémoires flash, la mémoire utilisée pour stocker les fichiers a une durée de vie de 100'000 cycles d'effacement environ. C'est assez, mais ce n'est pas illimité. Prenez donc garde à ne pas écrire et effacer inutilement des fichiers en boucle très rapidement, sous peine de détruire votre module.

5. 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.

5.1. Prérequis

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

Un ordinateur

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

Yoctopuce fournit les bibliothèques logicielles permettant de piloter ses modules pour les systèmes d'exploitation suivants: Windows, macOS, Linux et Android. Les modules Yoctopuce ne nécessitent pas l'installation de driver (ou pilote) spécifiques, car ils utilisent le driver HID¹ fourni en standard dans tous les systèmes d'exploitation.

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

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

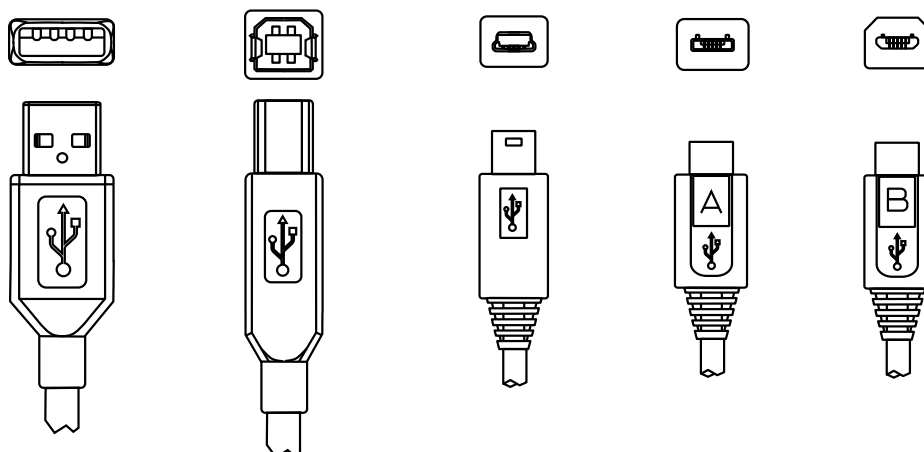
Les versions de Linux supportées sont les kernels 2.6, 3.x et 4.x. D'autres versions du kernel et même d'autres variantes d'Unix sont très susceptibles d'être utilisées sans problème, puisque le support de Linux est fait via l'API standard de la **libusb**, disponible aussi pour FreeBSD par exemple.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

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

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

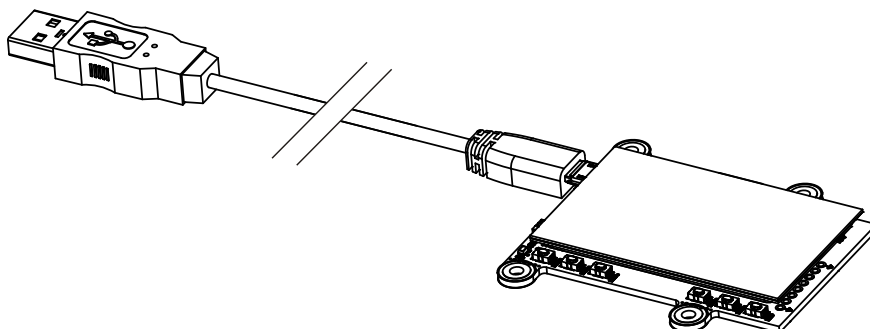
Un cable USB 2.0 de type A-micro B

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



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

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



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

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

5.2. Test de la connectivité USB

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

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

Rendez-vous sur le site de Yoctopuce et téléchargez le programme *Virtual Hub*³, Il est disponible pour Windows, Linux et Mac OS X. En temps normal le programme Virtual Hub sert de couche d'abstraction pour les langages qui ne peuvent pas accéder aux couches matérielles de votre ordinateur. Mais il offre aussi une interface sommaire pour configurer vos modules et tester les fonctions de base, on accède à cette interface à l'aide d'un simple browser web⁴. Lancez le *Virtual Hub* en ligne de commande, ouvrez votre browser préféré et tapez l'adresse <http://127.0.0.1:4444>. Vous devriez voir apparaître la liste des modules Yoctopuce raccordés à votre ordinateur.

Serial	Logical Name	Description	Action
VIRTHUB0-7d1a86fb0		VirtualHub	configure view log file
YD128X64-0AFFA		Yocto-MaxiDisplay	configure view log file beacon

[Show device functions](#)

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

5.3. Localisation

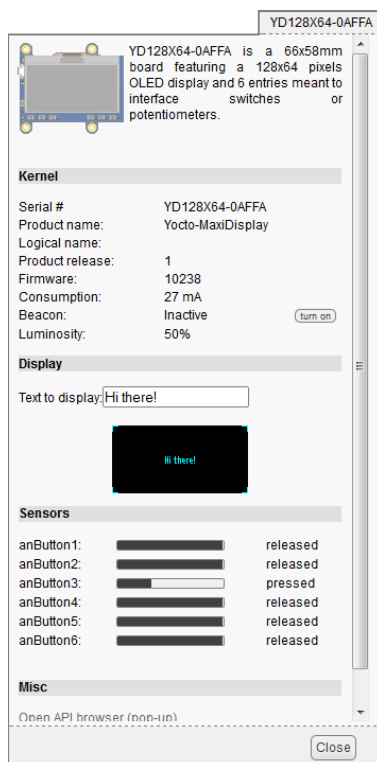
Il est alors possible de localiser physiquement chacun des modules affichés en cliquant sur le bouton **beacon**, cela a pour effet de mettre la Yocto-Led du module correspondant en mode "balise", elle se met alors à clignoter ce qui permet de la localiser facilement. Cela a aussi pour effet d'afficher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

5.4. 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-MaxiDisplay-G.

³ www.yoctopuce.com/FR/virtualhub.php

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

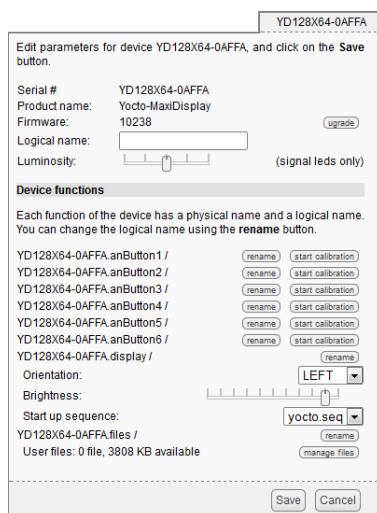


Propriétés du module Yocto-MaxiDisplay-G.

Cette fenêtre vous permet entre autres d'afficher un texte arbitraire sur l'écran et de contrôler l'état des entrées anButtons.

5.5. Configuration

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



Configuration du module Yocto-MaxiDisplay-G.

Firmware

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

Pour mettre à jour un firmware, cliquez simplement sur le bouton **upgrade** de la fenêtre de configuration et suivez les instructions. Si pour une raison ou une autre, la mise à jour venait à

échouer, débranchez puis rebranchez le module. Recommencer la procédure devrait résoudre alors le problème. Si le module a été débranché alors qu'il était en cours de reprogrammation, il ne fonctionnera probablement plus et ne sera plus listé dans l'interface. Mais il sera toujours possible de le reprogrammer correctement en utilisant le programme *Virtual Hub*⁵ en ligne de commande ⁶.

Nom logique du module

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

Luminosité

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

Nom logique des fonctions

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

Les fonctions fournies par le module Yocto-MaxiDisplay-G sont *display* qui correspond l'écran, *anButton1* à *anButton6* pour gérer les entrée potentiomètre et *files* qui correspond au file system.

⁵ www.yoctopuce.com/FR/virtualhub.php

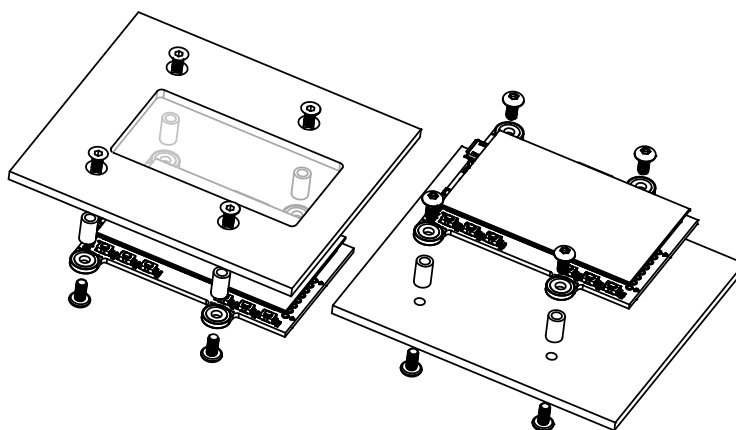
⁶ Consultez la documentation du virtual hub pour plus de détails

6. Montage et connectique

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

6.1. 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-MaxiDisplay-G dispose de trous de montage 3mm. Vous pouvez utiliser ces trous pour y passer des vis. Le diamètre de la tête de ces vis ne devra pas dépasser 8mm, sous peine d'endommager les circuits du module. Veillez à 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.

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

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

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

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

6.3. Compatibilité électromagnétique (EMI)

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

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

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

vous pourrez en général récupérer la conformité avec la norme d'émission en utilisant un boîtier métallique offrant une enveloppe de blindage externe.

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

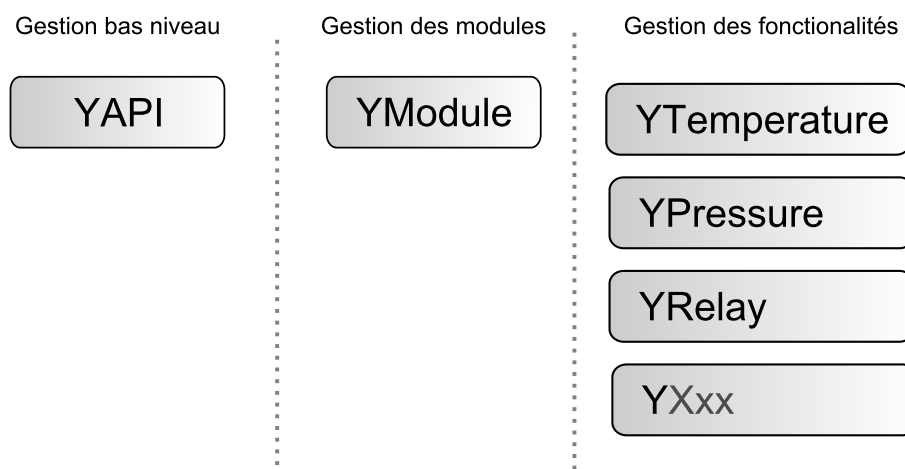
7. Programmation, concepts généraux

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-MaxiDisplay-G 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.

7.1. 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-MaxiDisplay-G 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.

7.2. Le module Yocto-MaxiDisplay-G

Le module Yocto-MaxiDisplay-G est un écran OLED de 128x64 pixels. Il inclut un filesystem permettant de stocker des fichiers (images, polices, séquences) et six instances de la fonction *AnButton*, correspondant aux six entrées analogiques (lecture de potentiomètre ou de bouton) présentes sur le module.

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

display : Display

attribut	type	modifiable ?
logicalName	Texte	modifiable
advertisedValue	Texte	modifiable
enabled	Booléen	modifiable
startupSeq	Texte	modifiable
brightness	0..100%	modifiable
orientation	Type énuméré	modifiable
displayWidth	Nombre entier	lecture seule
displayHeight	Nombre entier	lecture seule
displayType	Type énuméré	lecture seule
layerWidth	Nombre entier	lecture seule
layerHeight	Nombre entier	lecture seule
layerCount	Nombre entier	lecture seule
command	Texte	modifiable

files : Files

attribut	type	modifiable ?
logicalName	Texte	modifiable
advertisedValue	Texte	modifiable
filesCount	Nombre entier	lecture seule
freeSpace	Nombre entier	lecture seule

anButton1 : AnButton
anButton2 : AnButton
anButton3 : AnButton
anButton4 : AnButton
anButton5 : AnButton
anButton6 : AnButton

attribut	type	modifiable ?
logicalName	Texte	modifiable
advertisedValue	Texte	modifiable
calibratedValue	Nombre entier	lecture seule
rawValue	Nombre entier	lecture seule
analogCalibration	On/Off	modifiable
calibrationMax	Nombre entier	modifiable
calibrationMin	Nombre entier	modifiable
sensitivity	Nombre entier	modifiable
isPressed	Booléen	lecture seule
lastTimePressed	Temps	lecture seule
lastTimeReleased	Temps	lecture seule
pulseCounter	Nombre entier	modifiable
pulseTimer	Temps	lecture seule
inputType	Type énuméré	modifiable

7.3. 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-MaxiDisplay-G, ce numéro de série commence toujours par YD128G64. 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 74 en usine.

productRelease

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

firmwareRelease

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

persistentSettings

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

luminosity

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

beacon

Etat de la balise de localisation du module.

upTime

Temps écoulé depuis la dernière mise sous tension du module.

usbCurrent

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

rebootCountdown

Compte à rebours pour déclencher un redémarrage spontané du module.

userVar

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

7.4. Display

Interface pour interagir avec les écrans, disponibles par exemple dans le Yocto-Display, le Yocto-MaxiDisplay, le Yocto-MaxiDisplay-G et le Yocto-MiniDisplay

La classe `YDisplay` permet de piloter les écrans Yoctopuce. L'interface de contrôle des écrans Yoctopuce est conçue pour afficher facilement des informations et des images. Le module est capable de gérer seul la superposition de plusieurs couches graphiques, qui peuvent être dessinées individuellement, sans affichage immédiat, puis librement positionnées sur l'écran. Il est aussi capable de rejouer des séquences de commandes pré-enregistrées (animations). Pour afficher du contenu sur l'écran, il faut utiliser la méthode `display.get_displayLayer` pour récupérer la (ou les) couche(s) graphique(s) dans lesquelles vous voulez dessiner, puis écrire dedans à l'aide des méthodes de la classe `YDisplayLayer`.

logicalName

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

enabled

Etat d'activité de l'écran. L'écran peut être allumé ou éteint à volonté par cet attribut.

startupSeq

Nom de la séquence à jouer à la mise sous tension de l'écran.

brightness

Intensité lumineuse de l'écran. C'est une valeur entière variant entre 0 (écran très sombre) et 100 (écran très lumineux).

orientation

Orientation de l'écran. L'orientation est définie comme le côté de l'écran où se trouve la prise USB lorsque l'écran est droit.

displayWidth

Largeur de l'écran, en pixels.

displayHeight

Hauteur de l'écran, en pixels.

displayType

Type d'écran: monochrome (MONO), niveaux de gris (GRAY) ou couleur (RGB).

layerWidth

Largeur des couches affichables, en pixels.

layerHeight

Hauteur des couches affichables, en pixels.

layerCount

Nombre des couches affichables disponibles.

command

Attribut magique permettant d'envoyer du contenu à l'écran. Si une commande n'est pas interprétée comme attendue, consultez les logs du module.

7.5. AnButton

Interface pour interagir avec les entrées analogiques, disponibles par exemple dans le Yocto-Buzzer, le Yocto-Knob, le Yocto-MaxiBuzzer et le Yocto-MaxiDisplay

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

logicalName

Chaîne de caractères contenant le nom logique de l'entrée analogique, 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 à l'entrée analogique. Si deux entrées analogiques 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'entrée analogique, et qui sera publiée automatiquement jusqu'au hub parent. Pour une entrée analogique, la valeur publiée est la valeur mesurée recalibrée (a number between 0 and 1000).

calibratedValue

Valeur recalibrée de l'entrée analogique, sous forme d'un entier variant entre 0 et 1000 inclus. Si aucune calibration n'est été faite, la valeur recalibrée est simplement la valeur mesurée ramenée dans l'intervalle 0...1000, sans correction de linéarité.

rawValue

Valeur mesurée de l'entrée analogique telle-quelle, sous forme d'un entier variant entre 0 et 4095. Elle vaut zéro lorsque la résistance à l'entrée est nulle (contact fermé), et tends vers 4095 lorsque la résistance à l'entrée tends vers l'infini (contact ouvert). Attention, cette valeur ne varie pas proportionnellement à la résistance (donc à la position du potentiomètre). Pour obtenir une valeur proportionnelle, lancez une calibration et utilisez la valeur calculée `calibratedValue`.

analogCalibration

Permet d'enclencher et de déclencher la procédure de calibration automatique de l'entrée analogique. Lorsque la calibration est enclenchée, le module enregistre les valeurs mesurées minimales et maximales dans `calibrationMin` et `calibrationMax`. Une fois la calibration terminée (déclenchée), le module peut calculer automatiquement en permanence une valeur recalibrée de la mesure, variant linéairement avec la valeur de résistance mesurée.

calibrationMax

Valeur mesurée maximale observée durant la calibration. Vous pouvez aussi changer cette valeur par logiciel pour imposer une calibration théorique.

calibrationMin

Valeur mesurée minimale observée durant la calibration. Vous pouvez aussi changer cette valeur par logiciel pour imposer une calibration théorique.

sensitivity

Sensibilité de l'entrée analogique pour le déclenchement de callbacks utilisateur. La sensibilité correspond à la différence de valeur nécessaire pour déclencher la propagation d'une nouvelle valeur publiée et l'appel du callback utilisateur correspondant. Une valeur trop petite peut pourrait causer des appels inutiles si l'entrée mesurée n'est pas suffisamment stable.

isPressed

Etat logique de l'entrée, si on la traite comme une entrée binaire (bouton on/off). L'état logique est pressé lorsque l'entrée est fermée, et non pressé lorsque l'entrée est ouverte. Le module implémente un léger lissage et un schmitt trigger qui permettent une mesure logique convenable.

lastTimePressed

Temps absolu de la dernière occurrence de "pression de bouton" observée sur l'entrée (transition du contact de ouvert à fermé). La base de temps est la même que l'attribut `upTime` du module, c'est à dire le temps écoulé depuis la dernière mise sous tension du module.

lastTimeReleased

Temps absolu de la dernière occurrence de "relâchement de bouton" observée sur l'entrée (transition du contact de fermé à ouvert). La base de temps est la même que l'attribut `upTime` du module, c'est à dire le temps écoulé depuis la dernière mise sous tension du module. Si on soustrait à cette valeur le `lastTimePressed`, on obtien la durée de la dernière pression.

pulseCounter

Compteur d'impulsions 32 bits, incrémenté à chaque fois que l'état du bouton passe chaque d'état (PRESSED / RELEASED) ce qui signifie que le compteur est incrémenté de deux après chaque impulsion. Ce compteur commence à zéro à chaque redémarrage du module, il peut aussi être réinitialisé avec `resetCounter()`.

pulseTimer

Temps écoulé depuis la dernière initilialisation du compteur d'impulsion (millisecondes)

inputType

Type de dispositif connecté à l'entrée (entrée analogique ou entrées binaires multiplexées)

7.6. Files

Interface pour interagir avec les systèmes de fichier, disponibles par exemple dans le Yocto-Color-V2, le Yocto-Serial, le YoctoHub-Ethernet et le YoctoHub-Wireless-n

La class `YFiles` permet d'accéder au système de fichier embarqué sur certains modules Yoctopuce. Le stockage de fichiers permet par exemple de personnaliser un service web (dans le cas d'un module connecté au réseau) ou pour d'ajouter un police de caractères (dans le cas d'un module d'affichage).

logicalName

Chaîne de caractères contenant le nom logique du système de fichier, 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 système de fichier. Si deux systèmes de fichier 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 système de fichier, et qui sera publiée automatiquement jusqu'au hub parent. Pour un système de fichier, la valeur publiée est le nombre de fichiers présents.

filesCount

Nombre de fichiers présents dans le système de fichier.

freeSpace

Espace disponible dans le système de fichiers pour charger des nouveaux fichiers, en octets.

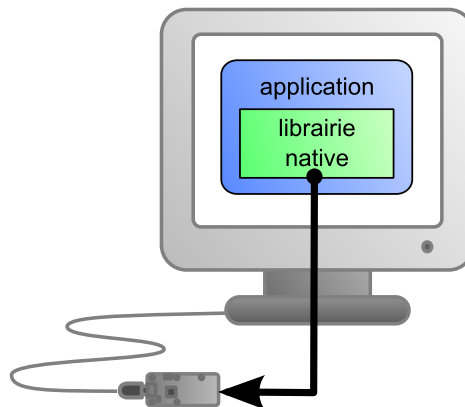
7.7. Quelle interface: Native, DLL ou Service?

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

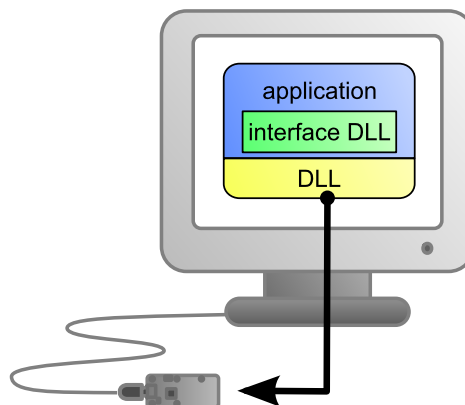
pour l'utilisateur final. Il lui suffira de brancher le câble USB et de lancer votre programme pour que tout fonctionne. Malheureusement, cette technique n'est pas toujours disponible ou même possible.



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

Contrôle natif par DLL

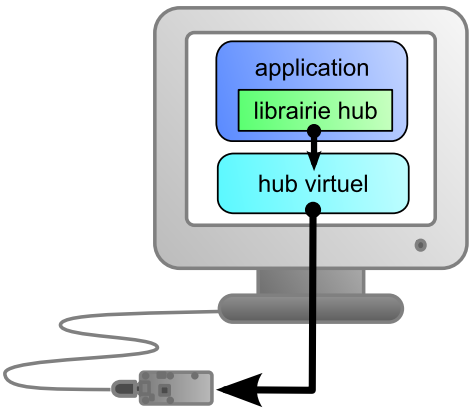
Ici l'essentiel du code permettant de contrôler les modules se trouve dans une DLL, et le programme est compilé avec une petite bibliothèque permettant de contrôler cette DLL. C'est la manière la plus rapide pour coder le support des modules dans un langage particulier. En effet la partie "utile" du 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 bibliothèque qui contrôle la DLL. Du point de vue de l'utilisateur final, il y a peu de différence: 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

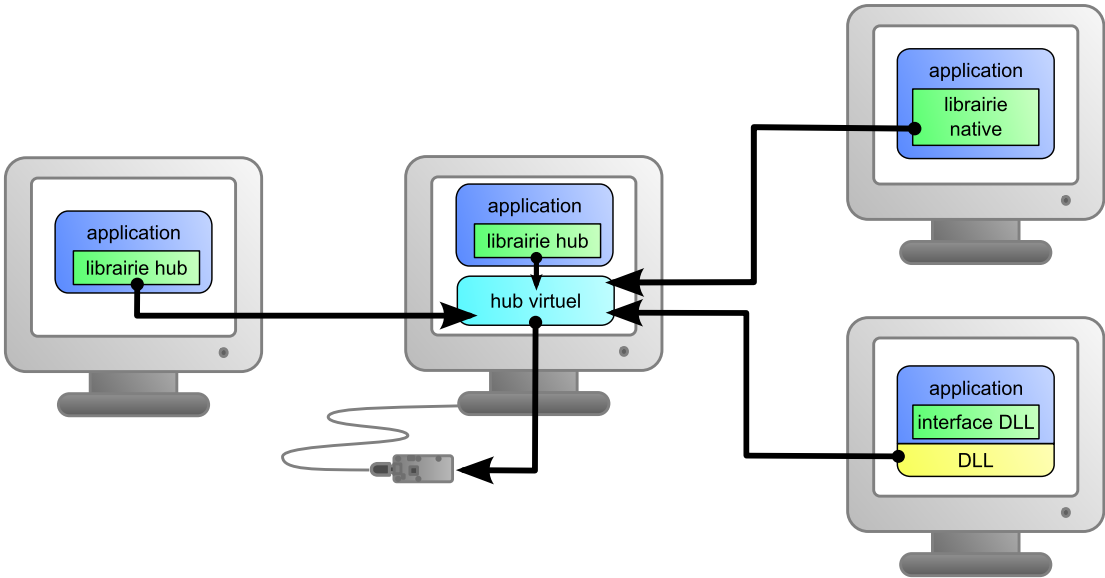
Contrôle par un service

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



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

En revanche la méthode de contrôle par un service offre un avantage non négligeable: l'application n'est pas obligé 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 un ou plusieurs VirtualHub.



Lorsqu'on utilise un VirtualHub, l'application de contrôle n'a plus besoin d'être sur la même machine que le module. Quel que soit langage de programmation choisi et le paradigme de contrôle utilisé; la programmation reste strictement identique. D'un langage à l'autre les fonctions ont exactement le même nom, prennent les mêmes paramètres. Les seules différences sont liées aux contraintes des langages eux-mêmes.

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

Méthode de support pour les différents langages.

Limitation des bibliothèques Yoctopuce

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

7.8. Programmation, par où commencer?

Arrivé à ce point du manuel, vous devriez connaître l'essentiel de la théorie à propos de votre Yocto-MaxiDisplay-G. Il est temps de passer à la pratique. Il vous faut télécharger la bibliothèque 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 bibliothèques de programmation. Dans certains langages, les bibliothèques 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-MaxiDisplay-G.

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

8. Utilisation du Yocto-MaxiDisplay-G en ligne de commande

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

L'API en ligne de commande se présente sous la forme d'un ensemble d'exécutables, un par type de fonctionnalité offerte par l'ensemble des produits Yoctopuce. Ces exécutables sont fournis pré-compilés pour toutes les plateformes/OS officiellement supportés par Yoctopuce. Bien entendu, les sources de ces exécutables sont aussi fournies¹.

8.1. 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-MaxiDisplay-G, ouvrir un shell et commencer à travailler en tapant par exemple:

```
UNABLE TO INCLUDE  
http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/cmdline/  
helloworld.cmd
```

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*.

8.2. 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]
```

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>

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

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 exécutable 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.

8.3. Contrôle de la fonction Display

Pour contrôler la fonction Display de votre Yocto-MaxiDisplay-G, vous avez besoin de l'exécutable YDisplay.

Vous pouvez par exemple lancer:

```
UNABLE TO INCLUDE
http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/cmdline/
helloworld.cmd
```

Cet exemple utilise la cible "any" pour signifier que l'on désire travailler sur la première fonction Display 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-MaxiDisplay-G avec le numéros de série YD128G64-123456 que vous auriez appelé "MonModule" et dont vous auriez nommé la fonction display "MaFonction", les cinq appels suivants seront strictement équivalents (pour autant que MaFonction ne soit définie qu'une fois, pour éviter toute ambiguïté).

```
C:\>YDisplay YD128G64-123456.display describe
C:\>YDisplay YD128G64-123456.MaFonction describe
C:\>YDisplay MonModule.display describe
C:\>YDisplay MonModule.MaFonction describe
C:\>YDisplay MaFonction describe
```

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

```
C:\>YDisplay all describe
```

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

```
C:\>YDisplay /help
```

8.4. 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 YD128G64-12346 set_logicalName MonPremierModule
```

```
C:\>YModule YD128G64-12346 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 YD128G64-12346 set_logicalName MonPremierModule
```

```
C:\>YModule YD128G64-12346 saveToFlash
```

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

```
C:\>YModule -s YD128G64-12346 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.

8.5. Limitations

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

Cette limitation peut aisément être contournée en utilisant un Virtual Hub: il suffit de faire tourner le VirtualHub³ sur la machine concernée et d'utiliser les executables de l'API en ligne de commande avec l'option `-r` par exemple, si vous utilisez:

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

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

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

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

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

9. Utilisation du Yocto-MaxiDisplay-G en Python

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

9.1. 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.

9.2. Librairie dynamique

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

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

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

9.3. Contrôle de la fonction Display

¹ <http://www.python.org/download/>

² www.yoctopuce.com/FR/libraries.php

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

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

# On récupère l'objet permettant d'interagir avec le module
display = YDisplay.FindDisplay("YD128G64-123456.display")

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

[...]
```

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.

YDisplay.FindDisplay

La fonction `YDisplay.FindDisplay` permet de retrouver un écran 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-MaxiDisplay-G avec le numéros de série *YD128G64-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *display* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128G64-123456.display")
display = YDisplay.FindDisplay("YD128G64-123456.MaFonction")
display = YDisplay.FindDisplay("MonModule.display")
display = YDisplay.FindDisplay("MonModule.MaFonction")
display = YDisplay.FindDisplay("MaFonction")
```

`YDisplay.FindDisplay` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'écran.

isOnline

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

get_displayLayer

La méthode `get_displayLayer()` de l'objet renvoyé par `YDisplay.FindDisplay` permet récupérer un objet correspondant à une des couches de l'écran. Cet objet fournit toutes les routines graphiques.

Un exemple réel

Lancez votre interpréteur Python et ouvrez le script correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-MaxiDisplay-G** 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.

UNABLE TO INCLUDE

<http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/python/helloworld.py>

9.4. 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()

```

9.5. Gestion des erreurs

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

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

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

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

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

10. Utilisation du Yocto-MaxiDisplay-G en C++

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

Les bibliothèques Yoctopuce² pour C++ vous sont fournies au format source dans leur intégralité. Une partie de la bibliothèque de bas-niveau est écrite en C pur sucre, mais vous n'aurez à priori pas besoin d'interagir directement avec elle: tout a été fait pour que l'interaction soit le plus simple possible depuis le C++. La bibliothèque vous est fournie bien entendu aussi sous forme binaire, de sorte à pouvoir la linker directement si vous le préférez.

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

Afin des les garder simples, tous les exemples fournis dans cette documentation sont des applications consoles. Il va de soit que que 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.

10.1. Contrôle de la fonction Display

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

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

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

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

² www.yoctopuce.com/FR/libraries.php

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

// On récupère l'objet permettant d'interagir avec le module
YDisplay *display;
display = YDisplay::FindDisplay("YD128G64-123456.display");

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.h et yocto_display.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_display.h` est nécessaire pour gérer les modules contenant un écran, comme le Yocto-MaxiDisplay-G.

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.

YDisplay::FindDisplay

La fonction `YDisplay::FindDisplay` permet de retrouver un écran 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-MaxiDisplay-G avec le numéro de série `YD128G64-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *display* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YDisplay *display = YDisplay::FindDisplay("YD128G64-123456.display");
YDisplay *display = YDisplay::FindDisplay("YD128G64-123456.MaFonction");
YDisplay *display = YDisplay::FindDisplay("MonModule.display");
YDisplay *display = YDisplay::FindDisplay("MonModule.MaFonction");
YDisplay *display = YDisplay::FindDisplay("MaFonction");
```

`YDisplay::FindDisplay` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'écran.

isOnline

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

get_displayLayer

La méthode `get_displayLayer()` de l'objet renvoyé par `YFindDisplay` permet récupérer un objet correspondant à une des couches de l'écran. Cet objet implémente toutes les routines graphiques.

Un exemple réel

Lancez votre environnement C++ et ouvrez le projet exemple correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-MaxiDisplay-G** 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.

UNABLE TO INCLUDE
<http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/C++/helloworld.cpp>

10.2. Contrôle de la partie module

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

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

#include "yocto_api.h"

using namespace std;

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

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

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

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

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

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

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

Modifications des réglages du module

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

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

#include "yocto_api.h"

using namespace std;

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

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

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

```

10.3. Gestion des erreurs

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

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

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

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

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

10.4. Intégration de la librairie Yoctopuce en C++

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

Intégration au format source (recommandé)

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

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

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

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

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

Intégration en librairie statique

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

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

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

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

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

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

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

Intégration en librairie dynamique

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

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

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

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

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


11. Utilisation du Yocto-MaxiDisplay-G en C#

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

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

11.1. Installation

Téléchargez la librairie Yoctopuce pour Visual C# depuis le site web de Yoctopuce². Il n'y a pas de programme d'installation, copiez simplement le contenu du fichier zip dans le répertoire de votre choix. Vous avez besoin essentiellement du contenu du répertoire *Sources*. Les autres répertoires contiennent la documentation et quelques programmes d'exemple. Les projets d'exemple sont des projets Visual C# 2010, si vous utilisez une version antérieure, il est possible que vous ayez à reconstruire la structure de ces projets.

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

La librairie Yoctopuce pour Visual C# .NET se présente sous la forme d'une DLL et de fichiers sources en Visual C#. La DLL n'est pas une DLL .NET mais une DLL classique, écrite en C, qui gère les communications à bas niveau avec les modules³. Les fichiers sources en Visual C# gèrent la partie haut niveau de l'API. Vous avez donc besoin de cette DLL et des fichiers .cs du répertoire *Sources* pour créer un projet gérant des modules Yoctopuce.

Configuration d'un projet Visual C#

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

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

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

² www.yoctopuce.com/FR/libraries.php

³ Les sources de cette DLL sont disponibles dans l'API C++

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

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

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

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

11.3. Contrôle de la fonction Display

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

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

// On récupère l'objet permettant d'interagir avec le module
YDisplay display = YDisplay.FindDisplay("YD128G64-123456.display");

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

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.

YDisplay.FindDisplay

La fonction `YDisplay.FindDisplay` permet de retrouver un écran 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-MaxiDisplay-G avec le numéros de série `YD128G64-123456` que vous auriez appelé `"MonModule"` et dont vous auriez nommé la fonction `display` `"MaFonction"`, les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128G64-123456.display");
display = YDisplay.FindDisplay("YD128G64-123456.MaFonction");
display = YDisplay.FindDisplay("MonModule.display");
```

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


```
display = YDisplay.FindDisplay("MonModule.MaFonction");
display = YDisplay.FindDisplay("MaFonction");
```

`YDisplay.FindDisplay` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'écran.

isOnline

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

get_displayLayer

La méthode `get_displayLayer()` de l'objet renvoyé par `YDisplay.FindDisplay` permet récupérer un objet correspondant à une des couches de l'écran. Cet objet fournit toutes les routines graphiques.

Un exemple réel

Lancez Visual C# et ouvrez le projet exemple correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-MaxiDisplay-G** 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.

UNABLE TO INCLUDE

<http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/C-sharp/helloworld.cs>

11.4. Contrôle de la partie module

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

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

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

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

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

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

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

```

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

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

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

```

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

Modifications des réglages du module

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

```

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

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

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

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

            if (YAPI.RegisterHub("usb", ref errmsg) != YAPI.SUCCESS) {

```

```

        Console.WriteLine("RegisterHub error: " + errormsg);
        Environment.Exit(0);
    }

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

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

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

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

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

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

```

11.5. 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.

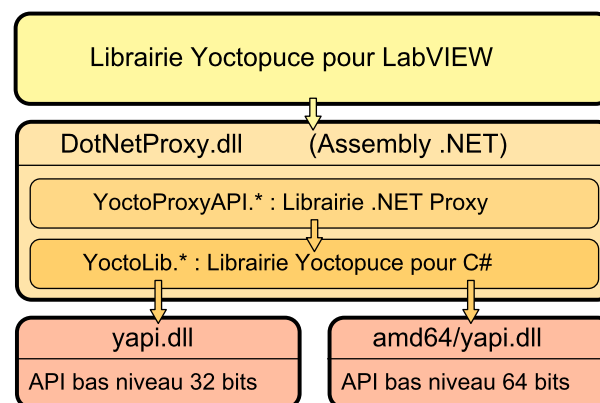
Quand vous travaillez sans les exceptions, il est possible d'obtenir un code d'erreur et un message expliquant l'origine de l'erreur en le demandant à l'objet qui a retourné une erreur à l'aide des méthodes `errType()` et `errMessage()`. Ce sont les mêmes informations qui auraient été associées à l'exception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

12. Utilisation du Yocto-MaxiDisplay-G 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.

12.1. 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`.

12.2. 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.

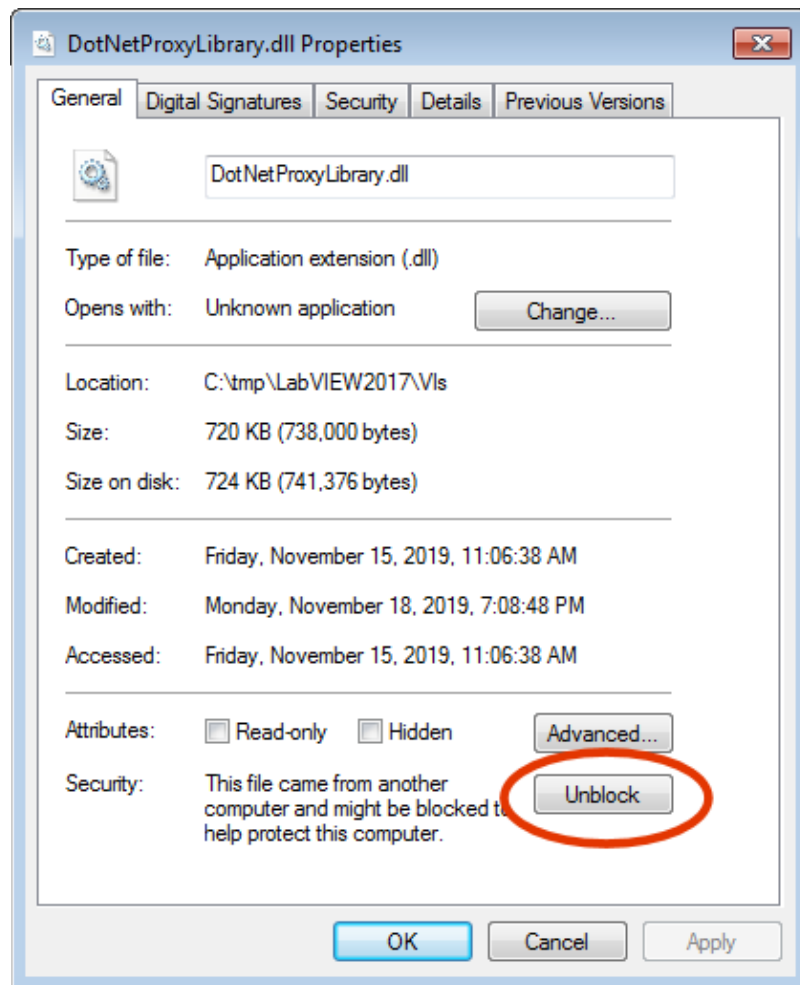
12.3. Installation

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.

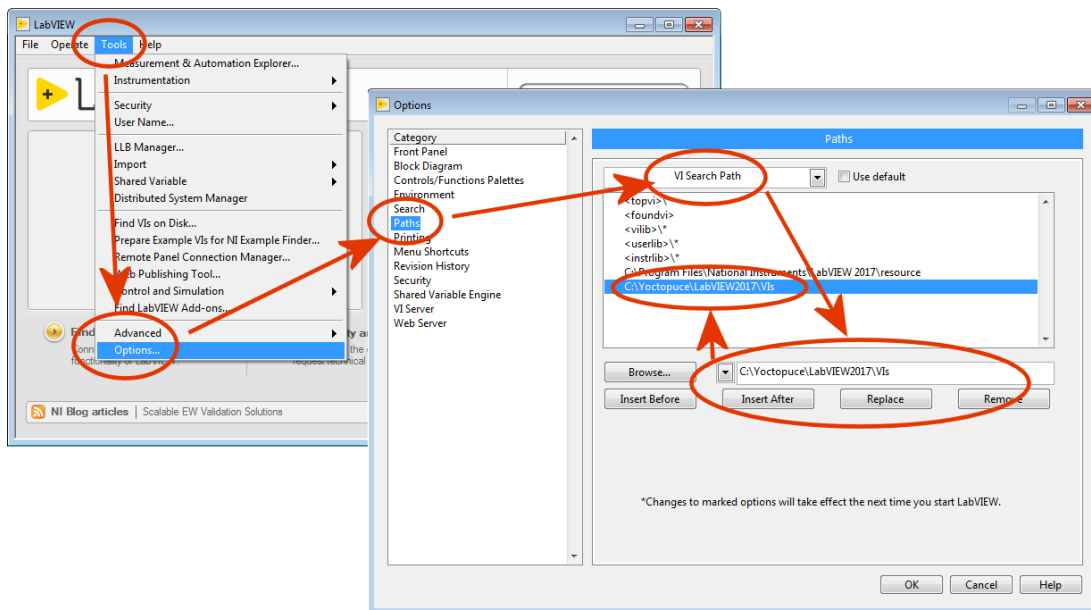
Pour installer l'API Yoctopuce pour LabVIEW vous avez plusieurs méthodes à votre disposition.

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*.

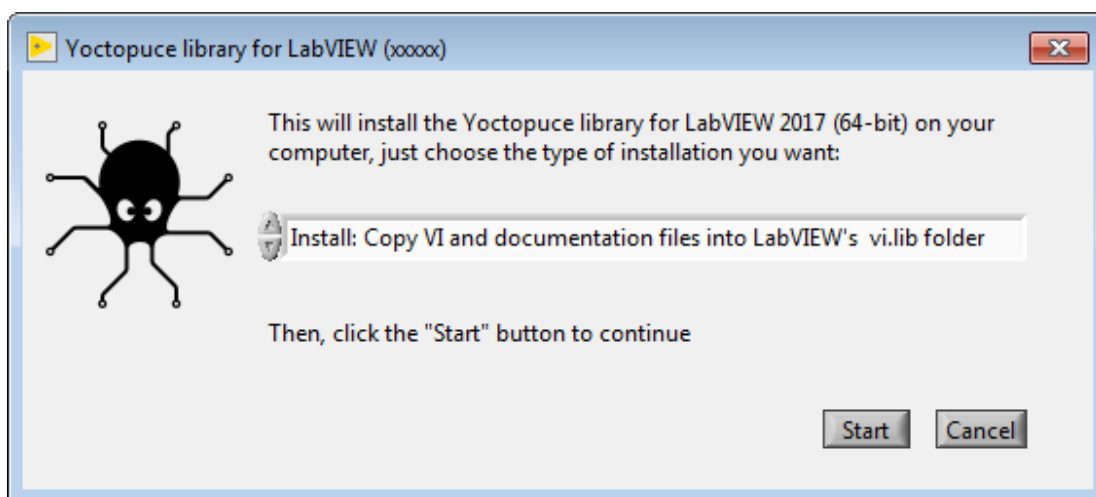
² <https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P8XnSAK>



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.

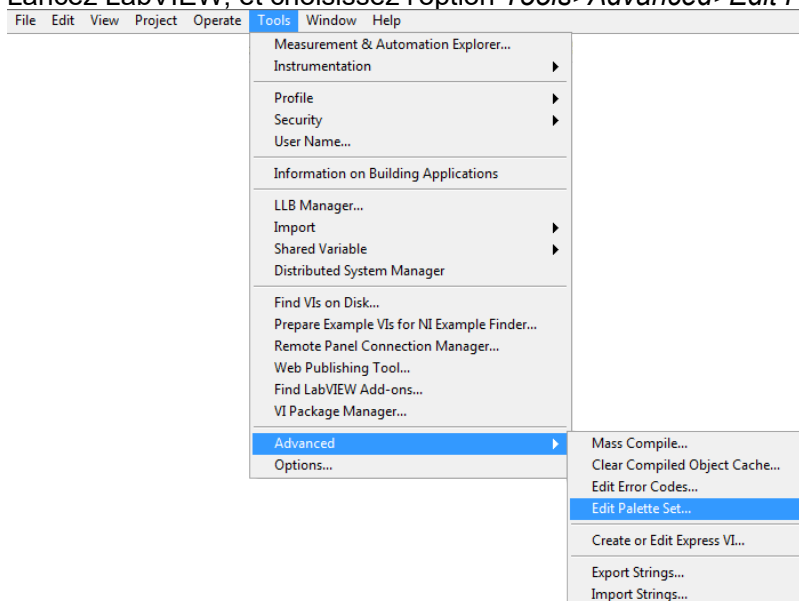
L'installateur reconnaît les répertoires contenant la librairie Yoctopuce en testant l'existence du fichier *YRegisterHub.vi*.

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:

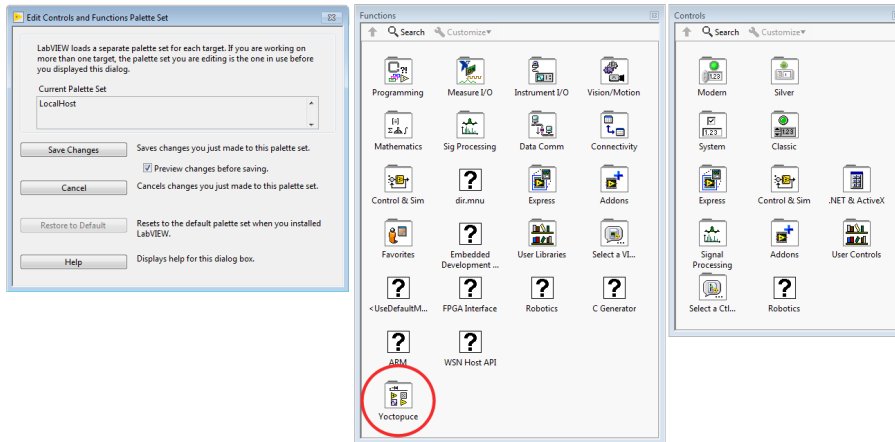
1. Créez un répertoire *Yoctopuce\API* dans le répertoire *C:\Program Files\National Instruments\LabVIEW xxxx\vi.lib*, et copiez tous les VIs et les DLL du répertoire *VIs* dedans.
2. Créez un répertoire *Yoctopuce* dans le répertoire *C:\Program Files\National Instruments\LabVIEW xxxx\menus\Categories*
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>

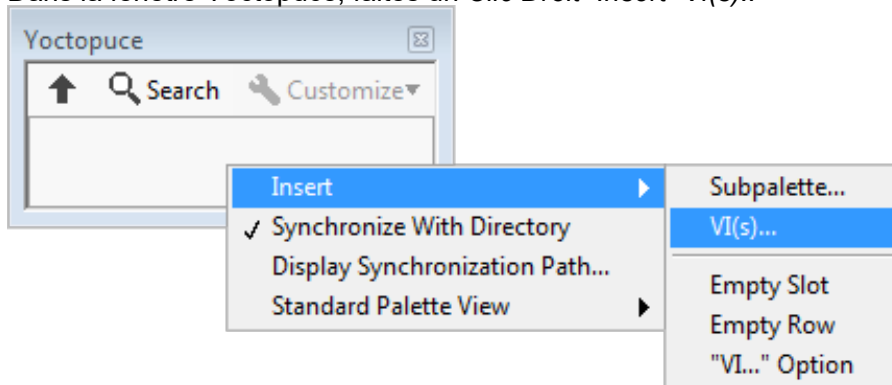
Trois fenêtres vont apparaître:

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

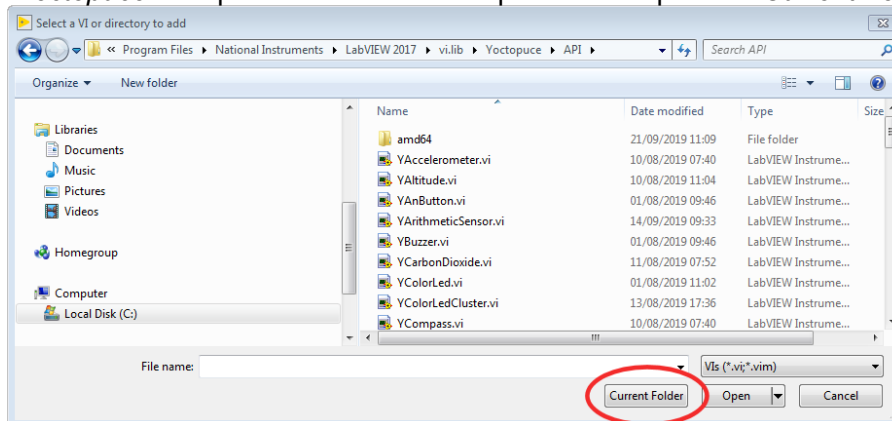


Dans la fenêtre *Function*, vous trouverez une icône *Yoctopuce*. Double-cliquez dessus, ce qui fera apparaître une fenêtre "Yoctopuce" vide.

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

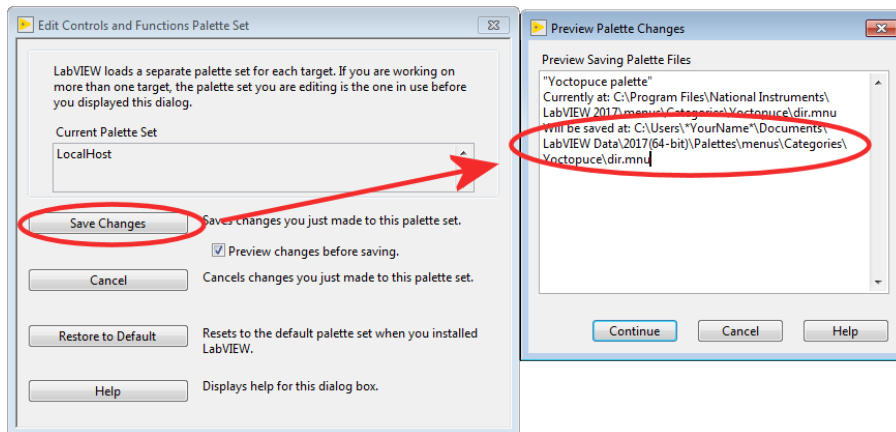


ce qui fera apparaître un sélecteur de fichier. Placer le sélecteur dans le répertoire *vi.lib* \Yoctopuce\API que vous avez créé au point 1 et cliquez sur *Current Folder*



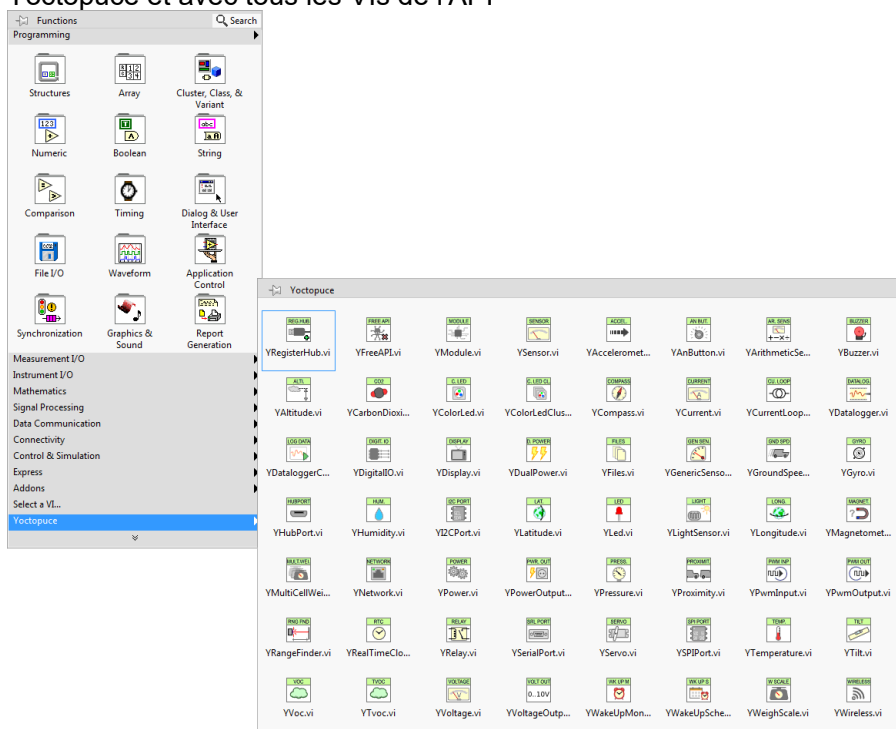
Tous les VIs Yoctopuce vont apparaître dans la fenêtre Yoctopuce. Par défaut, ils sont triés dans l'ordre alphabétique, mais vous pouvez les arranger comme bon vous semble en les glissant avec la souris. Pour que la palette soit bien utilisable, nous vous suggérons de réorganiser les icônes sur 8 colonnes.

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.

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

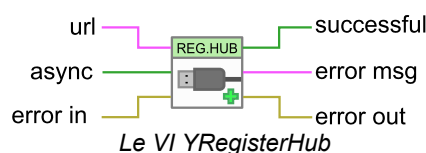


12.4. Présentation des VIs Yoctopuce

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.

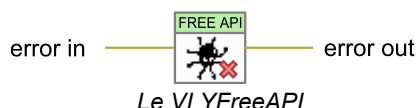
Dans le cas d'une adresse IP, le VI `YRegisterHub` va essayer de contacter cette adresse et générera une erreur s'il n'y arrive pas, à moins que le paramètre *async* ne soit mis à TRUE. Si *async* est mis à TRUE, aucune erreur ne sera générée, et les modules Yoctopuce correspondant à cette adresse IP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

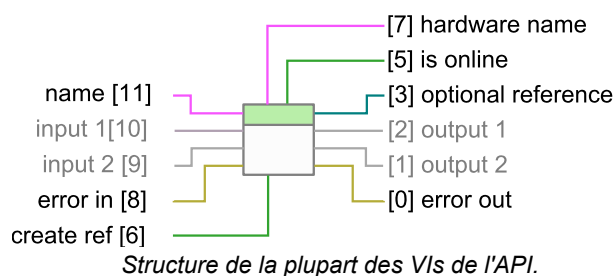
Le VI `YFreeAPI` permet de libérer les ressources allouées par l'API Yoctopuce.



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 terminée, 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.

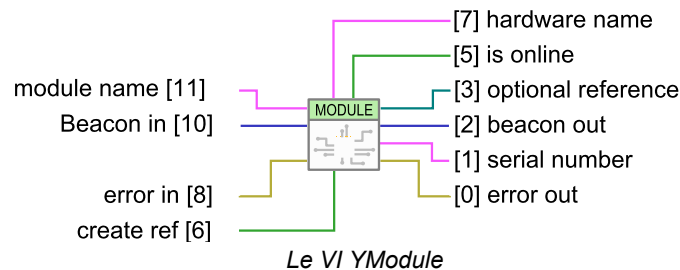
Vous trouverez la liste des fonctions disponibles sur votre Yocto-MaxiDisplay-G au chapitre *Programmation, concepts généraux*.

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.

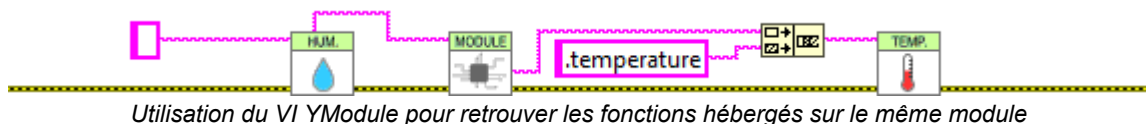
Si le paramètre *name* contient une chaîne vide, le VI ciblera la première fonction disponible du même type qu'il trouvera. Si aucune fonction n'est disponible, *is online* contiendra FALSE.

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.

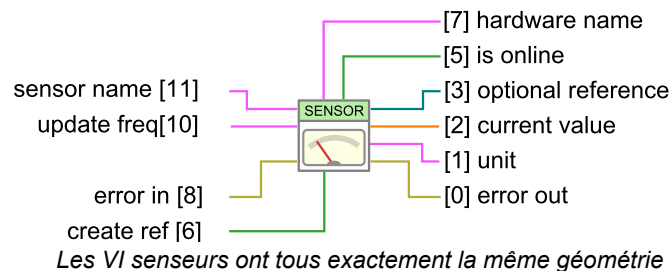


L'entrée *name* fonctionne de manière légèrement différente des autres VIs. S'il est appelé avec le paramètre *name* correspondant à un nom de fonction, le VI `YModule` trouvera la fonction *Module* du module hébergeant la fonction. Il est donc possible de trouver facilement le numéro de série du module 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.

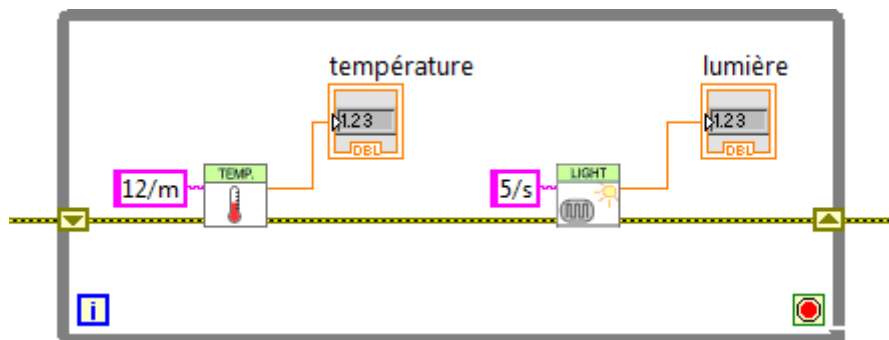


Le paramètre d'entrée *update freq* est une chaîne de caractères qui permet de configurer la façon dont la valeur de sortie est mis à jour:

- "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.

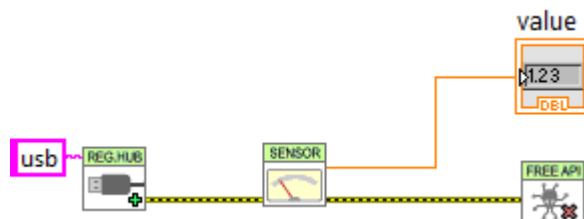


Changement de la fréquence de mise à jour du même module

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.

12.5. Fonctionnement et utilisation des VIs

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

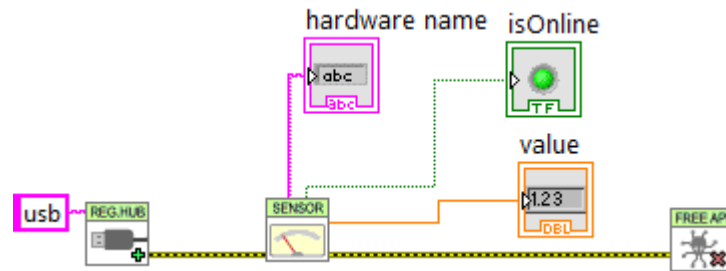


Exemple minimal d'utilisation de l'API Yoctopuce pour LabVIEW

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:

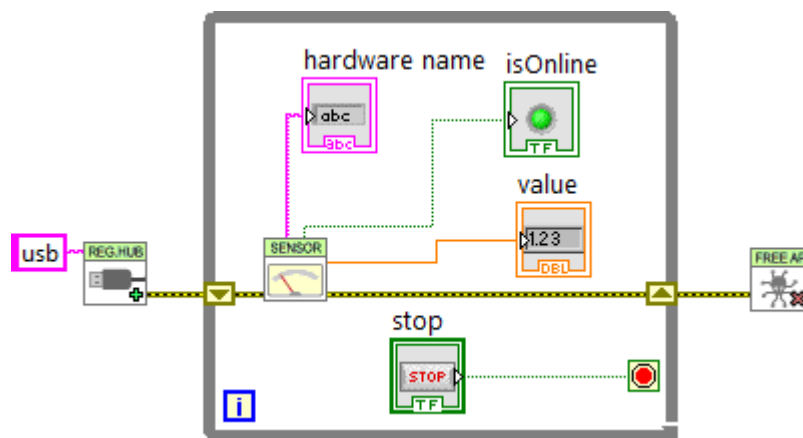
1. Il initialise l'API en mode natif ("usb") avec le VI `YRegisterHub`
2. Il affiche la valeur du premier capteur Yoctopuce qu'il trouve à l'aide du VI `YSensor`
3. Il libère l'API grâce au VI `YFreeAPI`

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

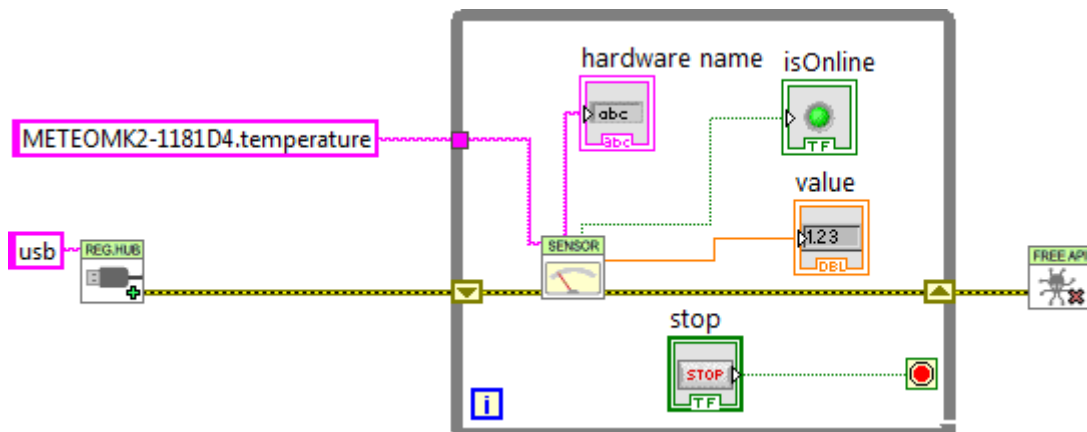
Les VIs de l'API Yoctopuce ne sont qu'une porte d'entrée sur la mécanique interne de la librairie Yoctopuce. Cette mécanique fonctionne indépendamment des VIs Yoctopuce. En effet, la plupart des communications avec les modules électroniques sont gérées automatiquement en arrière plan. C'est pourquoi 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.

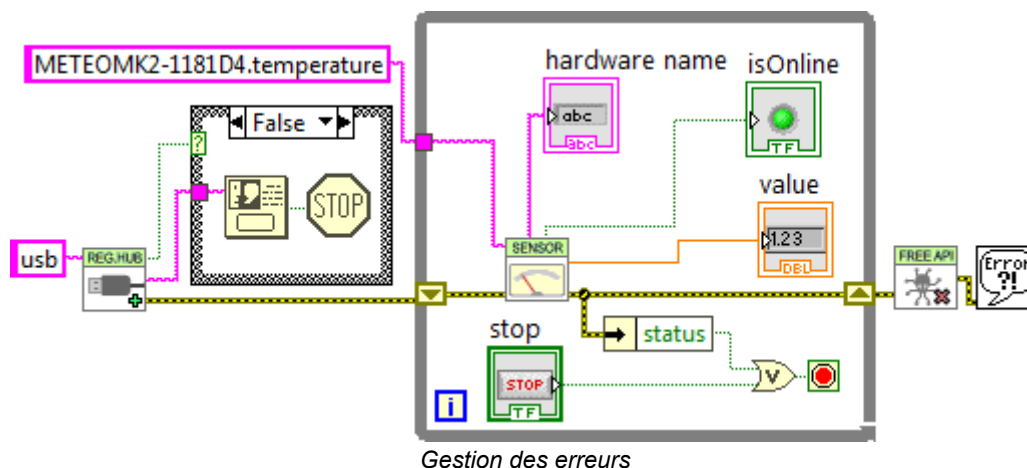


Utilisation de noms pour identifier les fonctions à utiliser

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.

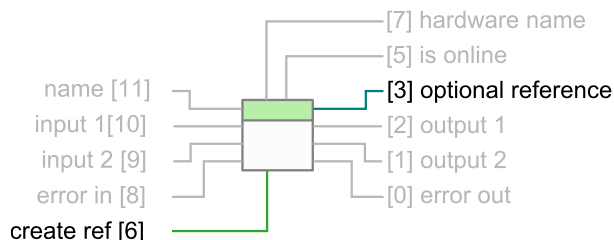


Le message d'erreur le plus courant est "*Another process is already using yAPI*". Il signifie qu'une autre application, LabVIEW ou autre, utilise déjà l'API en module USB natif. En effet, pour des raisons techniques, l'API USB native ne peut être utilisée que par une seule application à la fois sur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

12.6. 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.

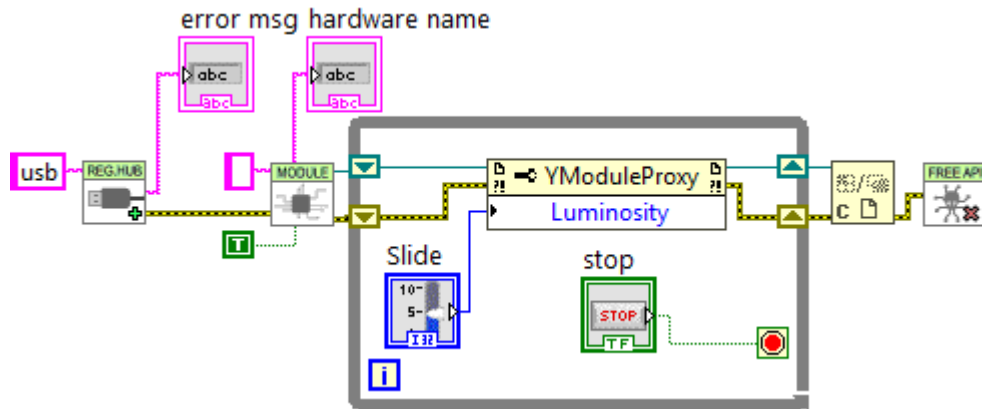
Chaque VI correspondant à une classe dispose de deux connecteurs *create ref* et *optional ref* qui permettent d'obtenir une référence sur l'objet *Proxy* de l'API .NET Proxy sur laquelle est construite la librairie LabVIEW.



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

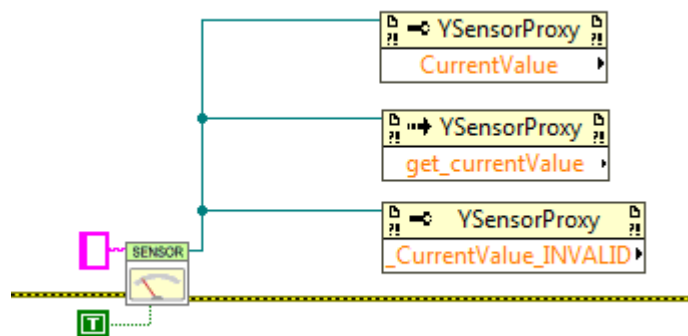
Pour obtenir cette référence, il suffit de mettre *optional ref* à TRUE. Attention, il est impératif de fermer toute référence créée de cette manière, sous peine de saturer rapidement la mémoire de l'ordinateur.

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.

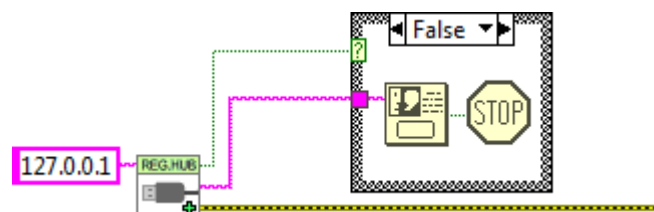


Noeuds Property et Invoke: Utilisation de propriétés, méthodes et constantes

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.

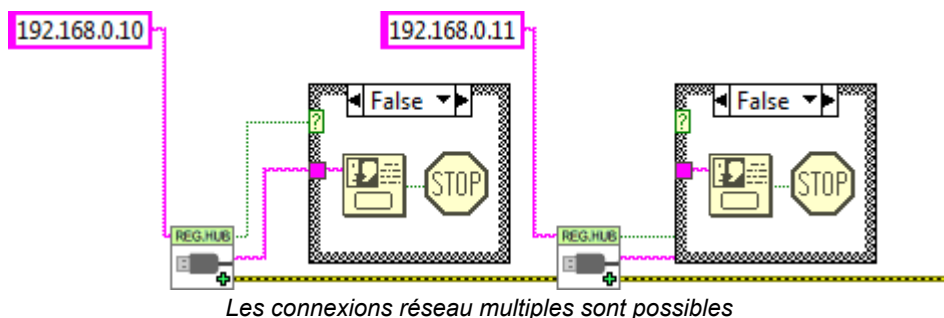


Utilisation en mode réseau

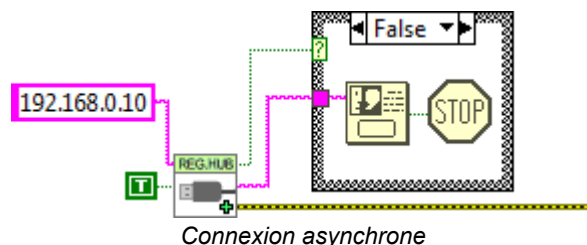
⁷ www.yoctopuce.com/FR/products/category/extensions-et-reseau

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.

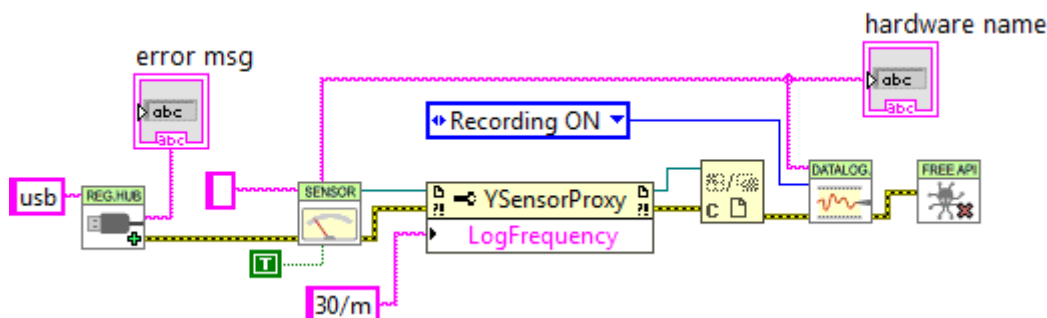


12.7. Gestion du datalogger

Quasiment tous les capteurs Yoctopuce disposent d'un enregistreur de données qui permet de stocker les mesures des capteurs 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

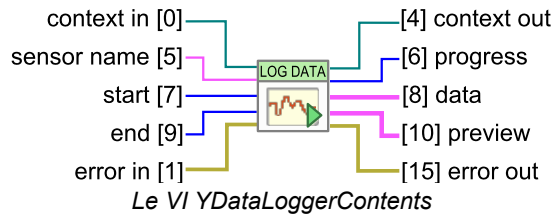
Pour ce faire, il faut configurer la fréquence d'enregistrement en utilisant la propriété "LogFrequency" que l'on atteint avec une référence sur l'objet *Proxy* du capteur utilisé, puis il faut mettre en marche l'enregistreur grâce au VI YDataLogger. Noter qu'à la manière du VI YModule, le VI YDataLogger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



Enclenchement de l'enregistrement de données dans le datalogger

Lecture

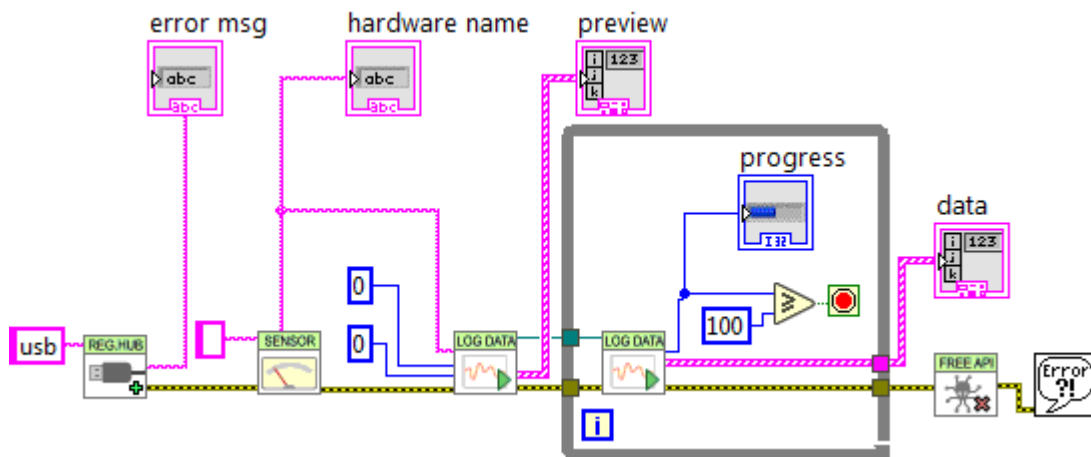
La récupération des données de l'enregistreur se fait à l'aide du VI YDataLoggerContents.



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



Récupération du contenu de l'enregistreur de données

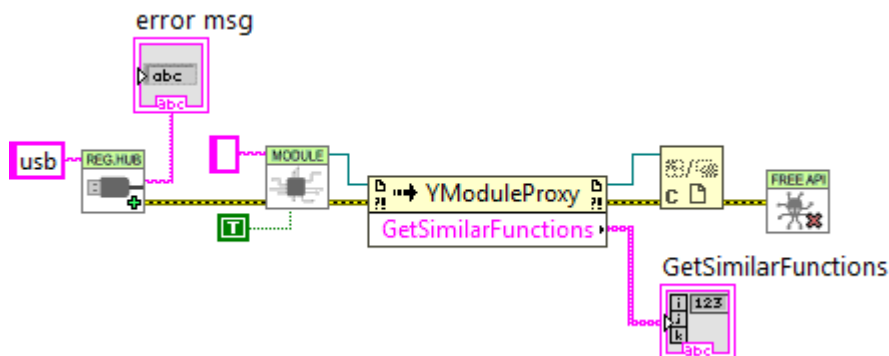
Les résultats et le résumé sont rendus sous la forme d'un tableau de structures qui contiennent les champs suivants:

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

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

12.8. Énumération de fonctions

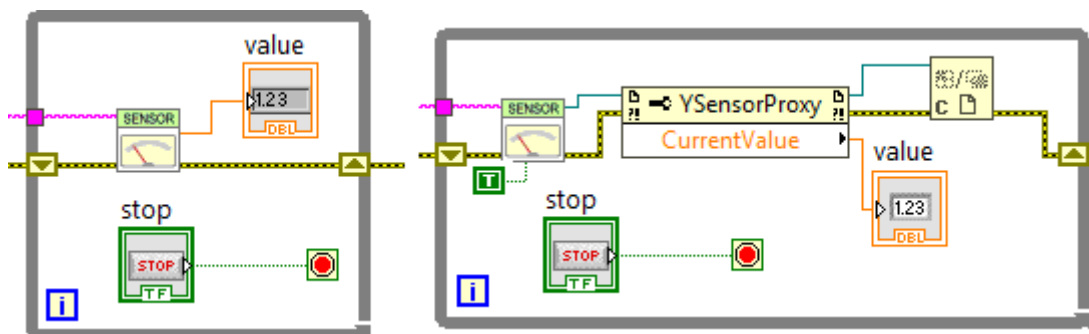
Chaque VI correspondant à un objet de l'API .NET Proxy permet de faire une énumération de toutes les fonctions de la même classe via la méthode *getSimilarfunctions()* de l'objet Proxy correspondant. Ainsi il est ainsi aisé de faire un inventaire de tous les modules connectés, de tous les capteurs connectés,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



Récupération de la liste de tous les modules connectés

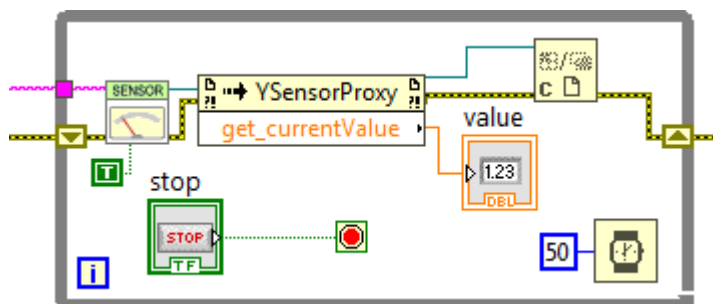
12.9. Un mot sur les performances

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

12.10. Un exemple complet de programme LabVIEW

UNABLE TO INCLUDE

<http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/doc-labview-example-FR.html>

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

12.11. 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*.

13. Utilisation du Yocto-MaxiDisplay-G en Java

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

13.1. Préparation

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

- La librairie de programmation pour Java¹
- Le programme VirtualHub² pour Windows, Mac OS X ou Linux selon l'OS que vous utilisez

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

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

13.2. Contrôle de la fonction Display

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

```
[...]
// 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
display = YDisplay.FindDisplay("YD128G64-123456.display");

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

¹ www.yoctopuce.com/FR/libraries.php

² www.yoctopuce.com/FR/virtualhub.php

```
{
    // Utiliser display.get_displayLayer()
    [...]
}

[...]
```

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.

YDisplay.FindDisplay

La fonction `YDisplay.FindDisplay` permet de retrouver un écran 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-MaxiDisplay-G avec le numéro de série `YD128G64-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `display` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128G64-123456.display")
display = YDisplay.FindDisplay("YD128G64-123456.MaFonction")
display = YDisplay.FindDisplay("MonModule.display")
display = YDisplay.FindDisplay("MonModule.MaFonction")
display = YDisplay.FindDisplay("MaFonction")
```

`YDisplay.FindDisplay` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'écran.

isOnline

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

get_displayLayer

La méthode `get_displayLayer()` de l'objet renvoyé par `YDisplay.FindDisplay` permet récupérer un objet correspondant à une des couches de l'écran. Cet objet fournit toutes les routines graphiques.

Un exemple réel

Lancez votre environnement java et ouvrez le projet correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-MaxiDisplay-G** 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.

UNABLE TO INCLUDE

<http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/Java/helloworld.java>

13.3. 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();
}

```

13.4. 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.

14. Utilisation du Yocto-MaxiDisplay-G 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.

14.1. Accès Natif et Virtual Hub.

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

14.2. 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.

14.3. Compatibilité

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

¹ www.yoctopuce.com/FR/libraries.php

Android 4.x

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

Support USB *host*

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

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

Matériel supporté

La librairie est testée et validée sur les machines suivantes:

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

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

14.4. Activer le port USB sous Android

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

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

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

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

```
</manifest>
```

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

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

Lancement automatique

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

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

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

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

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

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

14.5. Contrôle de la fonction Display

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

```
[...]
// On active la détection des modules sur USB
YAPI.EnableUSBHost(this);
YAPI.RegisterHub("usb");
[...]
// On récupère l'objet permettant de communiquer avec le module
display = YDisplay.FindDisplay("YD128G64-123456.display");

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

[...]
```

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.

YDisplay.FindDisplay

La fonction `YDisplay.FindDisplay` permet de retrouver un écran 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-MaxiDisplay-G avec le numéros de série `YD128G64-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *display* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128G64-123456.display")
display = YDisplay.FindDisplay("YD128G64-123456.MaFonction")
display = YDisplay.FindDisplay("MonModule.display")
display = YDisplay.FindDisplay("MonModule.MaFonction")
display = YDisplay.FindDisplay("MaFonction")
```

`YDisplay.FindDisplay` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'écran.

isOnline

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

get_displayLayer

La méthode `get_displayLayer()` de l'objet renvoyé par `YDisplay.FindDisplay` permet récupérer un objet correspondant à une des couches de l'écran. Cet objet fournit toutes les routines graphiques.

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.

UNABLE TO INCLUDE

<http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/Android/helloworld.java>

14.6. Contrôle de la partie module

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

```
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 OnItemClickListener
{
    private ArrayAdapter<String> aa;
    private YModule module = null;

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

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

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

```

}

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

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

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

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

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

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

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

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

```

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

Modifications des réglages du module

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

```
package com.yoctopuce.doc_examples;

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

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

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

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

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

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

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

    private void DisplayModuleInfo()
```

```

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

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

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

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

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

```

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

Enumeration des modules

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

```

package com.yoctopuce.doc_examples;

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

```

14.7. Gestion des erreurs

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

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

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

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

15. Utilisation du Yocto-MaxiDisplay-G 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`.

15.1. Utiliser la librairie Yoctopuce pour TypeScript

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

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

```
npm install -g typescript
```

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

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

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

```
npm install
```

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

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

15.2. 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.

15.3. Contrôle de la fonction Display

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

```
// 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 { YDisplay } from 'yoctolib-cjs/yocto_display.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 display: YDisplay = YDisplay.FindDisplay("YD128G64-123456.display");
```

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

```
if(await display.isOnline())
{
    // Utiliser display.get_displayLayer()
    [...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

Import de `yocto_api` et `yocto_display`

Ces deux imports permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api` doit toujours être inclus, et `yocto_display` est nécessaire pour gérer les modules contenant un écran, comme le Yocto-MaxiDisplay-G. 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`.

YDisplay.FindDisplay

La méthode `FindDisplay` permet de retrouver un écran 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-MaxiDisplay-G avec le numéro de série `YD128G64-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `display` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128G64-123456.display")
display = YDisplay.FindDisplay("YD128G64-123456.MaFonction")
display = YDisplay.FindDisplay("MonModule.display")
display = YDisplay.FindDisplay("MonModule.MaFonction")
display = YDisplay.FindDisplay("MaFonction")
```

`YDisplay.FindDisplay` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'écran.

isOnline

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

get_displayLayer

La méthode `get_displayLayer()` de l'objet renvoyé par `YDisplay.FindDisplay` permet récupérer un objet correspondant à une des couches de l'écran. Cet objet fournit toutes les routines graphiques.

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-MaxiDisplay-G** 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-MaxiDisplay-G 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-MaxiDisplay-G et où vous avez lancé le VirtualHub.

UNABLE TO INCLUDE

<http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/typescript/helloworld.ts>

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-MaxiDisplay-G**. 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.

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.

```
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();

```

15.5. Gestion des erreurs

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

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

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

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

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

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

16. Utilisation du Yocto-MaxiDisplay-G 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.

16.1. 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.

16.2. Utiliser la librairie Yoctopuce pour JavaScript / EcmaScript 2017

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

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

- La librairie de programmation pour Javascript / EcmaScript 2017¹
- Le programme VirtualHub² pour Windows, Mac OS X ou Linux selon l'OS que vous utilisez

Décompressez les fichiers de la librairie dans un répertoire de votre choix, branchez vos modules et lancez le programme VirtualHub. Vous n'avez pas besoin d'installer de driver.

Utiliser la librairie Yoctopuce officielle pour node.js

Commencez par installer sur votre machine de développement la version actuelle de Node.js (7.6 ou plus récente), C'est très simple. Vous pouvez l'obtenir sur le site officiel: <http://nodejs.org>. Assurez vous de l'installer entièrement, y compris npm, et de l'ajouter à votre system path.

Vous pouvez ensuite prendre l'exemple de votre choix dans le répertoire `example_nodejs` (par exemple `example_nodejs/Doc-Inventory`). Allez dans ce répertoire. Vous y trouverez un fichier décrivant l'application (`package.json`) et le code source de l'application (`demo.js`). Pour charger automatiquement et configurer les librairies nécessaires à l'exemple, tapez simplement:

```
npm install
```

Une fois que c'est fait, vous pouvez directement lancer le code de l'application:

```
node demo.js
```

Utiliser une copie locale de la librairie Yoctopuce avec node.js

Si pour une raison ou une autre vous devez faire des modifications au code de la librairie, vous pouvez facilement configurer votre projet pour utiliser le code source de la librairie qui se trouve dans le répertoire `lib/` plutôt que le package npm officiel. Pour cela, lancez simplement la commande suivante dans le répertoire de votre projet:

```
npm link ../../lib
```

Utiliser la librairie Yoctopuce dans un navigateur (HTML)

Pour les exemples HTML, c'est encore plus simple: il n'y a rien à installer. Chaque exemple est un simple fichier HTML que vous pouvez ouvrir directement avec un navigateur pour l'essayer. L'inclusion de la librairie Yoctopuce ne demande rien de plus qu'un simple tag HTML `<script>`.

Utiliser la librairie Yoctopuce avec des anciennes version de JavaScript

Si vous avez besoin d'utiliser cette librairie avec des moteurs JavaScript plus anciens, vous pouvez utiliser Babel³ pour transpiler votre code et la librairie dans une version antérieure du langage. Pour installer Babel avec les réglages usuels, tapez:

¹ www.yoctopuce.com/FR/libraries.php

² www.yoctopuce.com/FR/virtualhub.php

³ <http://babeljs.io>

```
npm instal -g babel-cli
npm instal babel-preset-env
```

Normalement vous demanderez à Babel de poser les fichiers transpilés dans un autre répertoire, nommé `compat` par exemple. Pour ce faire, utilisez par exemple les commandes suivantes:

```
babel --presets env demo.js --out-dir compat/
babel --presets env ../../lib --out-dir compat/
```

Bien que ces outils de transpilation soient basés sur `node.js`, ils fonctionnent en réalité pour traduire n'importe quel type de fichier JavaScript, y compris du code destiné à fonctionner dans un navigateur. La seule chose qui ne peut pas être faite aussi facilement est la transpilation de scripts codés en dur à l'intérieur même d'une page HTML. Il vous faudra donc sortir ce code dans un fichier `.js` externe si il utiliser la syntaxe EcmaScript 2017, afin de le transpiler séparément avec Babel.

Babel dispose de nombreuses fonctionnalités intéressantes, comme un mode de surveillance qui traduit automatiquement au vol vos fichiers dès qu'il détecte qu'un fichier source a changé. Consultez les détails dans la documentation de Babel.

Compatibilité avec l'ancienne librairie JavaScript

Cette nouvelle librairie n'est pas compatible avec l'ancienne librairie JavaScript, car il n'existe pas de possibilité d'implémenter l'ancienne API bloquante sur la base d'une API asynchrone. Toutefois, les noms des méthodes sont les mêmes, et l'ancien code source synchrone peut facilement être rendu asynchrone simplement en ajoutant le mot-clé `await` devant les appels de méthode. Remplacez par exemple:

```
beaconState = module.get_beacon();
```

par

```
beaconState = await module.get_beacon();
```

Mis à part quelques exceptions, la plupart des méthodes redondantes `XXX_async` ont été supprimées, car elles auraient introduit de la confusion sur la manière correcte de gérer les appels asynchrones. Si toutefois vous avez besoin d'appeler un callback explicitement, il est très facile de faire appeler une fonction de callback à la résolution d'une méthode `async`, en utilisant l'objet `Promise` retourné. Par exemple, vous pouvez réécrire:

```
module.get_beacon_async(callback, myContext);
```

par

```
module.get_beacon().then(function(res) { callback(myContext, module, res); });
```

Si vous portez une application vers la nouvelle librairie, vous pourriez être amené à désirer des méthodes synchrones similaires à l'ancienne librairie (sans objet `Promise`), quitte à ce qu'elles retournent la dernière valeur reçue du capteur telle que stockée en cache, puisqu'il n'est pas possible de faire des communications bloquantes. Pour cela, la nouvelle librairie introduit un nouveau type de classes appelés *proxys synchrones*. Un proxy synchrone est un objet qui reflète la dernière valeur connue d'un objet d'interface, mais peut être accédé à l'aide de fonctions synchrones habituelles. Par exemple, plutôt que d'utiliser:

```
async function logInfo(module)
{
  console.log('Name: '+await module.get_logicalName());
  console.log('Beacon: '+await module.get_beacon());
}
...
```

```
logInfo(myModule);
...
```

on peut utiliser:

```
function logInfoProxy(moduleSyncProxy)
{
    console.log('Name: '+moduleProxy.get_logicalName());
    console.log('Beacon: '+moduleProxy.get_beacon());
}

logInfoSync(await myModule.get_syncProxy());
```

Ce dernier appel asynchrone peut aussi être formulé comme:

```
myModule.get_syncProxy().then(logInfoProxy);
```

16.3. Contrôle de la fonction Display

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

```
// En Node.js, on utilise la fonction require()
// En HTML, on utiliserait <script src="...">
require('yoctolib-es2017/yocto_api.js');
require('yoctolib-es2017/yocto_display.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 display = YDisplay.FindDisplay("YD128G64-123456.display");

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

Voyons maintenant en détail ce que font ces quelques lignes.

Require de yocto_api et yocto_display

Ces deux imports permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api` doit toujours être inclus, `yocto_display` est nécessaire pour gérer les modules contenant un écran, comme le Yocto-MaxiDisplay-G. 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.

YDisplay.FindDisplay

La méthode `FindDisplay` permet de retrouver un écran 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-MaxiDisplay-G avec le numéros de série *YD128G64-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *display* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128G64-123456.display")
display = YDisplay.FindDisplay("YD128G64-123456.MaFonction")
display = YDisplay.FindDisplay("MonModule.display")
display = YDisplay.FindDisplay("MonModule.MaFonction")
display = YDisplay.FindDisplay("MaFonction")
```

`YDisplay.FindDisplay` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'écran.

isOnline

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

get_displayLayer

La méthode `get_displayLayer()` de l'objet renvoyé par `YDisplay.FindDisplay` permet récupérer un objet correspondant à une des couches de l'écran. Cet objet fournit toutes les routines graphiques.

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-MaxiDisplay-G** 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-MaxiDisplay-G 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-MaxiDisplay-G et où vous avez lancé le VirtualHub.

UNABLE TO INCLUDE

<http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/ecmascript/node.js>

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-MaxiDisplay-G**. 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

UNABLE TO INCLUDE

<http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/ecmascript/>

helloworld.html

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

16.4. Contrôle de la partie module

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

```
"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);
}

```

16.5. Gestion des erreurs

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

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

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

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

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

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

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

17. Utilisation du Yocto-MaxiDisplay-G en PHP

PHP est, tout comme Javascript, un langage assez atypique lorsqu'il s'agit de discuter avec du hardware. Néanmoins, utiliser PHP avec des modules Yoctopuce offre l'opportunité de construire très facilement des sites web capables d'interagir avec leur environnement physique, ce qui n'est pas donné à tous les serveurs web. Cette technique trouve une application directe dans la domotique: quelques modules Yoctopuce, un serveur PHP et vous pourrez interagir avec votre maison depuis n'importe où dans le monde. Pour autant que vous ayez une connexion internet.

PHP fait lui aussi partie de ces langages qui ne vous permettront pas d'accéder directement aux couches matérielles de votre ordinateur. C'est pourquoi vous devrez faire tourner un hub virtuel sur la machine à laquelle sont branchés les modules

Pour démarrer vos essais en PHP, vous allez avoir besoin d'un serveur PHP 5.3 ou plus ¹ de préférence en local sur votre machine. Si vous souhaitez utiliser celui qui se trouve chez votre provider internet, c'est possible, mais vous devrez probablement configurer votre routeur ADSL pour qu'il accepte et forward les requêtes TCP sur le port 4444.

17.1. Préparation

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

- La librairie de programmation pour PHP²
- Le programme VirtualHub³ pour Windows, Mac OS X ou Linux selon l'OS que vous utilisez

Décompressez les fichiers de la librairie dans un répertoire de votre choix accessible à votre serveur web, branchez vos modules, lancez le programme VirtualHub, et vous pouvez commencer vos premiers test. Vous n'avez pas besoin d'installer de driver.

17.2. Contrôle de la fonction Display

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

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

¹ Quelques serveurs PHP gratuits: easyPHP pour windows, MAMP pour Mac Os X

² www.yoctopuce.com/FR/libraries.php

³ www.yoctopuce.com/FR/virtualhub.php

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

// On récupère l'objet permettant d'interagir avec le module
$display = YDisplay::FindDisplay("YD128G64-123456.display");

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.php et yocto_display.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_display.php` est nécessaire pour gérer les modules contenant un écran, comme le Yocto-MaxiDisplay-G.

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`.

YDisplay::FindDisplay

La fonction `YDisplay::FindDisplay` permet de retrouver un écran 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-MaxiDisplay-G avec le numéro de série `YD128G64-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `display` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
$display = YDisplay::FindDisplay("YD128G64-123456.display");
$display = YDisplay::FindDisplay("YD128G64-123456.MaFonction");
$display = YDisplay::FindDisplay("MonModule.display");
$display = YDisplay::FindDisplay("MonModule.MaFonction");
$display = YDisplay::FindDisplay("MaFonction");
```

`YDisplay::FindDisplay` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'écran.

isOnline

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

get_displayLayer

La méthode `get_displayLayer()` de l'objet renvoyé par `YFindDisplay` permet récupérer un objet correspondant à une des couches de l'écran. Cet objet implémente toutes les routines graphiques.

Un exemple réel

Ouvrez votre éditeur de texte préféré⁴, copiez le code ci dessous, sauvez-le dans un répertoire accessible par votre serveur web/PHP avec les fichiers de la librairie, et ouvrez-la page avec votre

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

browser favori. Vous trouverez aussi ce code dans le répertoire **Exemples/Doc-GettingStarted-Yocto-MaxiDisplay-G** 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.

UNABLE TO INCLUDE

<http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/php/helloworld.php>

17.3. Contrôle de la partie module

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

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

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

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

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

    if (isset($_GET['beacon'])) {
      if ($_GET['beacon']=='ON')
        $module->set_beacon(Y_BEACON_ON);
      else
        $module->set_beacon(Y_BEACON_OFF);
    }
    printf('serial: %s<br>', $module->get_serialNumber());
    printf('logical name: %s<br>', $module->get_logicalName());
    printf('luminosity: %s<br>', $module->get_luminosity());
    print('beacon: ');
    if($module->get_beacon() == Y_BEACON_ON) {
      printf("<input type='radio' name='beacon' value='ON' checked>ON ");
      printf("<input type='radio' name='beacon' value='OFF'>OFF<br>");
    } else {
      printf("<input type='radio' name='beacon' value='ON'>ON ");
      printf("<input type='radio' name='beacon' value='OFF' checked>OFF<br>");
    }
    printf('upTime: %s sec<br>', intval($module->get_upTime()/1000));
    printf('USB current: %smA<br>', $module->get_usbCurrent());
```

```

printf('logs:<br><pre>%s</pre>', $module->get_lastLogs());
YAPI::FreeAPI();
?>
<input type='submit' value='refresh'>
</FORM>
</BODY>
</HTML>

```

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

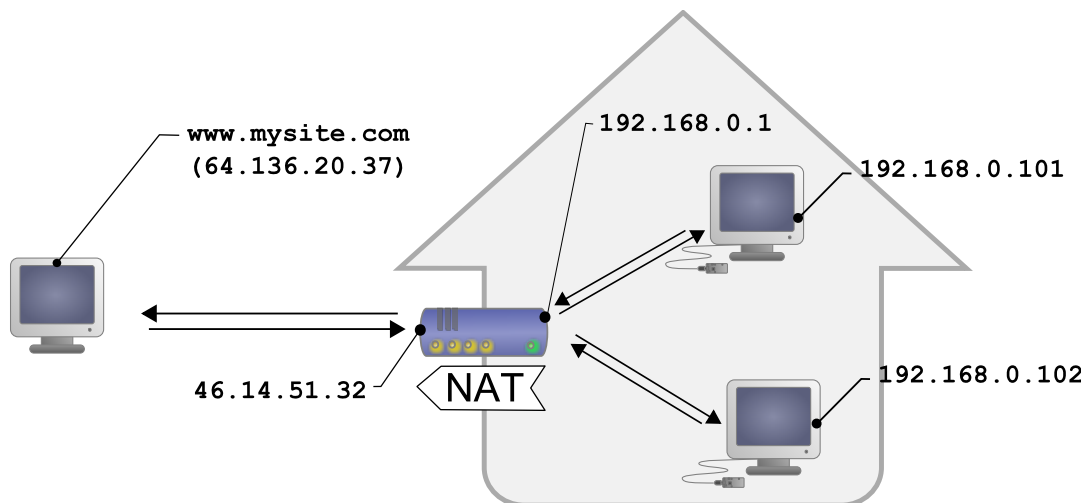
```
<HTML>
<HEAD>
<TITLE>inventory</TITLE>
</HEAD>
<BODY>
<H1>Device list</H1>
<TT>
<?php
    include('yocto_api.php');
    YAPI::RegisterHub("http://127.0.0.1:4444/");
    $module = YModule::FirstModule();
    while (!is_null($module)) {
        printf("%s (%s)<br>", $module->get_serialNumber(),
            $module->get_productName());
        $module=$module->nextModule();
    }
    YAPI::FreeAPI();
?>
</TT>
</BODY>
</HTML>
```

17.4. 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.

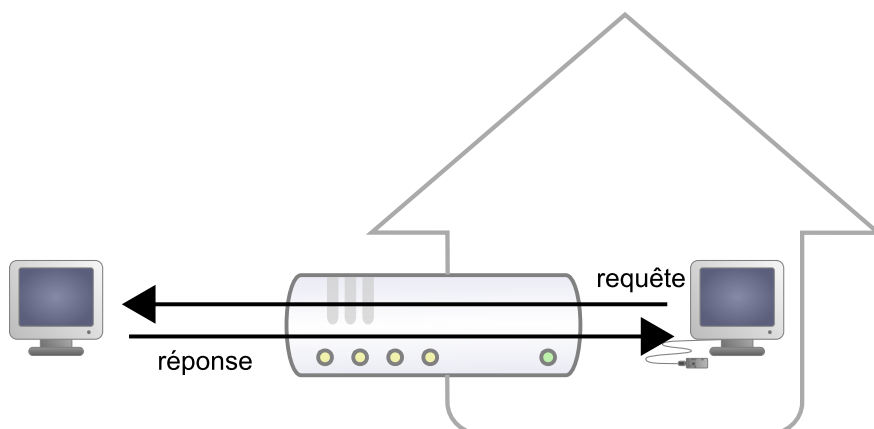
Le filtre NAT, avantages et inconvénients

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.

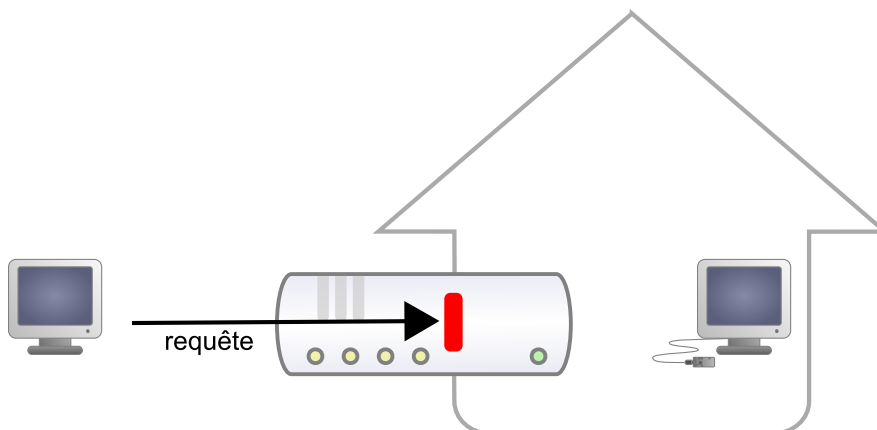


Configuration DSL typique, les machines du LAN sont isolées de l'extérieur par le router DSL

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.

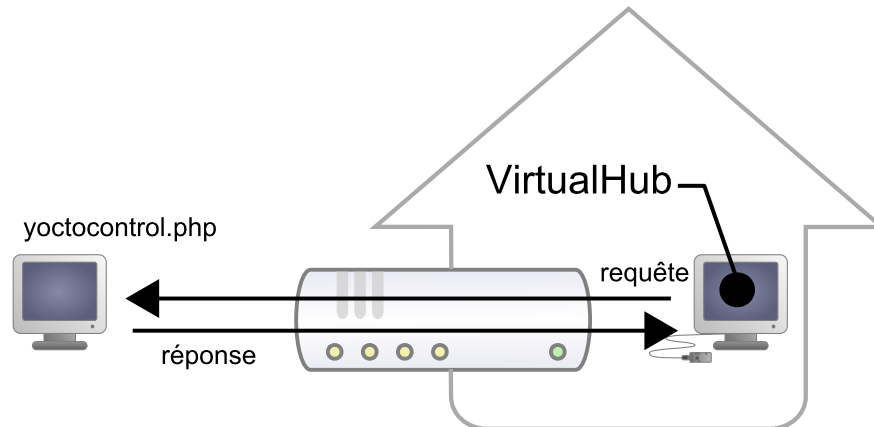


Mais les requêtes venant de l'extérieur sont bloquées.

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 publique chez soi pour une installation domotique et offrir un accès depuis l'extérieur. Une solution à ce problème, préconisée par de nombreux vendeurs de domotique, consiste à donner une visibilité externe au serveur de domotique lui-même, en ouvrant un port et en ajoutant une règle de routage dans la configuration

NAT du routeur DSL. Le problème de cette solution est qu'il expose le serveur de domotique aux attaques externes.

L'API par callback HTTP résout ce problème sans qu'il soit nécessaire de modifier la configuration du routeur DSL. Le script de contrôle des modules est placé sur un site externe, et c'est le *Virtual Hub* qui est chargé de l'appeler à intervalle régulier.



L'API par callback HTTP utilise le VirtualHub, et c'est lui qui initie les requêtes.

Configuration

L'API callback se sert donc du *Virtual Hub* comme passerelle. Toutes les communications sont initiées par le *Virtual Hub*, ce sont donc des communication sortantes, et par conséquent parfaitement autorisée par le routeur DSL.

Il faut configurer le *VirtualHub* pour qu'il appelle le script PHP régulièrement. Pour cela il faut:

1. Lancer un *VirtualHub*
2. Accéder à son interface, généralement 127.0.0.1:4444
3. Cliquer sur le bouton **configure** de la ligne correspondant au *VirtualHub* lui-même
4. Cliquer sur le bouton **edit** de la section **Outgoing callbacks**

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

Cliquer sur le bouton "configure" de la première ligne

VIRTHUB0-7d1a86fb09

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

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

Incoming connections

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

Outgoing callbacks

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

Save Cancel

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

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

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

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

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

Callback URL:

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

Username:
 Password:

4. Setup the desired frequency of notifications:

No less than seconds between two notification
 But notify after seconds in any case

5. Press on the **Test** button to check your parameters.

6. When everything works, press on the **OK** button.

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.

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 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 mêmes informations qui auraient été associées à l'exception si elles avaient été actives.

18. Utilisation du Yocto-MaxiDisplay-G en VisualBasic .NET

VisualBasic a longtemps été la porte d'entrée privilégiée vers le monde Microsoft. Nous nous devons donc d'offrir notre interface pour ce langage, même si la nouvelle tendance est le C#. Tous les exemples et les modèles de projet sont testés avec Microsoft Visual Basic 2010 Express, disponible gratuitement sur le site de Microsoft ¹.

18.1. Installation

Téléchargez la librairie Yoctopuce pour Visual Basic depuis le site web de Yoctopuce². Il n'y a pas de programme d'installation, copiez simplement le contenu du fichier zip dans le répertoire de votre choix. Vous avez besoin essentiellement du contenu du répertoire *Sources*. Les autres répertoires contiennent la documentation et quelques programmes d'exemple. Les projets d'exemple sont des projets Visual Basic 2010, si vous utilisez une version antérieure, il est possible que vous ayez à reconstruire la structure de ces projets.

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

La librairie Yoctopuce pour Visual Basic .NET se présente sous la forme d'une DLL et de fichiers sources en Visual Basic. La DLL n'est pas une DLL .NET mais une DLL classique, écrite en C, qui gère les communications à bas niveau avec les modules³. Les fichiers sources en Visual Basic gèrent la partie haut niveau de l'API. Vous avez donc besoin de cette DLL et des fichiers .vb du répertoire *Sources* pour créer un projet gérant des modules Yoctopuce.

Configuration d'un projet Visual Basic

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

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

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

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

² www.yoctopuce.com/FR/libraries.php

³ Les sources de cette DLL sont disponibles dans l'API C++

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

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

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

18.3. Contrôle de la fonction Display

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

```
[...]
' 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 display As YDisplay
display = YDisplay.FindDisplay("YD128G64-123456.display")

' Pour gérer le hot-plug, on vérifie que le module est là
If (display.isOnline()) Then
    ' Utiliser display.get_displayLayer()
    [...]
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.

YDisplay.FindDisplay

La fonction `YDisplay.FindDisplay` permet de retrouver un écran 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-MaxiDisplay-G avec le numéros de série `YD128G64-123456` que vous auriez appelé `"MonModule"` et dont vous auriez nommé la fonction `display` `"MaFonction"`, les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128G64-123456.display")
display = YDisplay.FindDisplay("YD128G64-123456.MaFonction")
display = YDisplay.FindDisplay("MonModule.display")
display = YDisplay.FindDisplay("MonModule.MaFonction")
display = YDisplay.FindDisplay("MaFonction")
```

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

`YDisplay.FindDisplay` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'écran.

isOnline

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

get_displayLayer

La méthode `get_displayLayer()` de l'objet renvoyé par `YFindDisplay` permet récupérer un objet correspondant à une des couches de l'écran. Cet objet implémente toutes les routines graphiques.

Un exemple réel

Lancez Microsoft VisualBasic et ouvrez le projet exemple correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-MaxiDisplay-G** 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.

UNABLE TO INCLUDE

<http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/VB/helloworld.vb>

18.4. 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")
            Else
                Console.WriteLine("OFF")
            End If
        End If
    End Sub
End Module
```

```

        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
```

18.5. 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.

19. Utilisation du Yocto-MaxiDisplay-G en Delphi

Delphi est l'héritier de Turbo-Pascal. A l'origine, Delphi était produit par Borland, mais c'est maintenant Embarcadero qui l'édite. Sa force réside dans sa facilité d'utilisation, il permet à quiconque ayant des notions de Pascal de programmer une application Windows en deux temps trois mouvements. Son seul défaut est d'être payant¹.

Les librairies pour Delphi sont fournies non pas sous forme de composants VCL, mais directement sous forme de fichiers source. Ces fichiers sont compatibles avec la plupart des version de Delphi ².

Afin des les garder simples, tous les exemples fournis dans cette documentation sont des applications consoles. Il va de soit que le fonctionnement des librairies est strictement identique avec des applications VCL.

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

19.1. Préparation

Connectez-vous sur le site de Yoctopuce et téléchargez la la librairie Yoctopuce pour Delphi³. Décompressez le tout dans le répertoire de votre choix, et ajoutez le sous-répertoire *sources* de l'archive dans la liste des répertoires des librairies de Delphi⁴.

Par défaut la librairie Yoctopuce pour Delphi utilise une DLL *yapi.dll*, toutes les applications que vous créerez avec Delphi devront avoir accès à cette DLL. Le plus simple est de faire en sorte qu'elle soit présente dans le même répertoire que l'exécutable de votre application.

19.2. Contrôle de la fonction Display

Il suffit de quelques lignes de code pour piloter un Yocto-MaxiDisplay-G. Voici le squelette d'un fragment de code Delphi qui utilise la fonction Display.

```
uses yocto_api, yocto_display;
```

¹ En fait, Borland a diffusé des versions gratuites (pour usage personnel) de Delphi 2006 et Delphi 2007, en cherchant un peu sur internet il est encore possible de les télécharger.

² Les librairies Delphi sont régulièrement testées avec Delphi 5 et Delphi XE2

³ www.yoctopuce.com/FR/libraries.php

⁴ Utilisez le menu **outils / options d'environnement**

```

var errmsg: string;
    display: TYDisplay;

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

// On récupère l'objet permettant d'interagir avec le module
display = yFindDisplay("YD128G64-123456.display")

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api et yocto_display

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

yRegisterHub

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

yFindDisplay

La fonction `yFindDisplay` permet de retrouver un écran 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-MaxiDisplay-G avec le numéro de série `YD128G64-123456` que vous auriez appelé *"MonModule"* et dont vous auriez nommé la fonction `display` *"MaFonction"*, les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```

display := yFindDisplay("YD128G64-123456.display");
display := yFindDisplay("YD128G64-123456.MaFonction");
display := yFindDisplay("MonModule.display");
display := yFindDisplay("MonModule.MaFonction");
display := yFindDisplay("MaFonction");
```

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

isOnline

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

get_displayLayer

La méthode `get_displayLayer()` de l'objet renvoyé par `yFindDisplay` permet récupérer un objet correspondant à une des couches de l'écran. Cet objet implémente toutes les routines graphiques.

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.

UNABLE TO INCLUDE

<http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/delphi/helloworld.dpr>

19.3. 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 = 'YD128G64-123456'; // 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 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 de changer le nom logique d'un module.

```
program saveSettings;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Yocto_API;

const
  serial = 'YD128G64-123456'; // 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.
```

19.4. 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 mêmes informations qui auraient été associées à l'exception si elles avaient été actives.

20. Utilisation du Yocto-MaxiDisplay-G 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.

20.1. 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.

20.2. 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.

20.3. 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-MaxiDisplay-G voici ce qu'il faut ajouter dans le n u d "Capabilities":

```
<DeviceCapability Name="humaninterfacedevice">
  <!-- Yocto-MaxiDisplay-G -->
  <Device Id="vidpid:24e0 004A">
    <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.

20.4. Contrôle de la fonction Display

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

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

// On récupère l'objet permettant d'interagir avec le module
YDisplay display = YDisplay.FindDisplay("YD128G64-123456.display");

// Pour gérer le hot-plug, on vérifie que le module est là
if (await display.IsOnline())
{
    // Use display.get_displayLayer()
    ...
}

[...]
```

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.

YDisplay.FindDisplay

La fonction `YDisplay.FindDisplay` permet de retrouver un écran 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-MaxiDisplay-G avec le numéro de série `YD128G64-123456` que vous auriez appelé "MonModule" et dont vous auriez nommé la fonction `display` "MaFonction", les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128G64-123456.display");
display = YDisplay.FindDisplay("YD128G64-123456.MaFonction");
display = YDisplay.FindDisplay("MonModule.display");
display = YDisplay.FindDisplay("MonModule.MaFonction");
display = YDisplay.FindDisplay("MaFonction");
```

`YDisplay.FindDisplay` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'écran.

isOnline

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

get_displayLayer

La méthode `get_displayLayer()` de l'objet renvoyé par `YDisplay.FindDisplay` permet récupérer un objet correspondant à une des couches de l'écran. Cet objet fourni toutes les routines graphiques.

20.5. Un exemple concret

Lancez Visual Studio et ouvrez le projet correspondant, fourni dans le répertoire **Examples/Doc-GettingStarted-Yocto-MaxiDisplay-G** 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.

UNABLE TO INCLUDE

<http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/UWP/helloworld.cs>

20.6. Contrôle de la partie module

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

```
using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.UI.Xaml.Controls;
using com.yoctopuce.YoctoAPI;

namespace Demo
{
    public class Demo : DemoBase
    {
        public string HubURL { get; set; }
        public string Target { get; set; }
        public bool Beacon { get; set; }

        public override async Task<int> Run()
        {
            YModule m;
            string errmsg = "";

            if (await YAPI.RegisterHub(HubURL) != YAPI.SUCCESS) {
                WriteLine("RegisterHub error: " + errmsg);
                return -1;
            }
            m = YModule.FindModule(Target + ".module"); // use serial or logical name
            if (await m.isOnline()) {
                if (Beacon) {
                    await m.set_beacon(YModule.BEACON_ON);
                } else {
                    await m.set_beacon(YModule.BEACON_OFF);
                }
            }

            WriteLine("serial: " + await m.get_serialNumber());
        }
    }
}
```

```

        WriteLine("logical name: " + await m.get_logicalName());
        WriteLine("luminosity: " + await m.get_luminosity());
        Write("beacon: ");
        if (await m.get_beacon() == YModule.BEACON_ON)
            WriteLine("ON");
        else
            WriteLine("OFF");
        WriteLine("upTime: " + (await m.get_upTime() / 1000) + " sec");
        WriteLine("USB current: " + await m.get_usbCurrent() + " mA");
        WriteLine("Logs:\r\n" + await m.get_lastLogs());
    } else {
        WriteLine(Target + " not connected on" + HubURL +
            "(check identification and USB cable)");
    }
    YAPI.FreeAPI();
    return 0;
}
}
}

```

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

Modifications des réglages du module

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

```

using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.UI.Xaml.Controls;
using com.yoctopuce.YoctoAPI;

namespace Demo
{
    public class Demo : DemoBase
    {
        public string HubURL { get; set; }
        public string Target { get; set; }
        public string LogicalName { get; set; }

        public override async Task<int> Run()
        {
            try {
                YModule m;

                await YAPI.RegisterHub(HubURL);

                m = YModule.FindModule(Target); // use serial or logical name
                if (await m.isOnline()) {
                    if (!YAPI.CheckLogicalName(LogicalName)) {
                        WriteLine("Invalid name (" + LogicalName + ")");
                        return -1;
                    }

                    await m.set_logicalName(LogicalName);
                    await m.saveToFlash(); // do not forget this
                    Write("Module: serial= " + await m.get_serialNumber());
                    WriteLine(" / name= " + await m.get_logicalName());
                } else {
                    Write("not connected (check identification and USB cable)");
                }
            } catch (YAPI_Exception ex) {
                WriteLine("RegisterHub error: " + ex.Message);
            }
        }
    }
}

```

```

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

```

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

Enumeration des modules

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

```

using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.UI.Xaml.Controls;
using com.yoctopuce.YoctoAPI;

namespace Demo
{
    public class Demo : DemoBase
    {
        public string HubURL { get; set; }

        public override async Task<int> Run()
        {
            YModule m;
            try {
                await YAPI.RegisterHub(HubURL);

                WriteLine("Device list");
                m = YModule.FirstModule();
                while (m != null) {
                    WriteLine(await m.get_serialNumber()
                        + " (" + await m.get_productName() + ")");
                    m = m.nextModule();
                }
            } catch (YAPI_Exception ex) {
                WriteLine("Error:" + ex.Message);
            }
            YAPI.FreeAPI();
            return 0;
        }
    }
}

```

20.7. 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);  
}
```


21. Utilisation du Yocto-MaxiDisplay-G en Objective-C

Objective-C est le langage de prédilection pour programmer sous Mac OS X, en raison de son intégration avec le générateur d'interfaces Cocoa. Pour pouvoir utiliser la librairie Objective-C vous aurez impérativement besoin de XCode 4.2, qui est disponible gratuitement sous Lion. Si vous êtes encore sous Snow Leopard il vous faudra être enregistré comme développeur auprès d'Apple pour pouvoir télécharger XCode 4.2. La librairie Yoctopuce est compatible ARC. Il vous sera donc possible de coder vos projet soit en utilisant la traditionnelle méthode de *retain / release*, soit en activant l'*Automatic Reference Counting*.

Les librairies Yoctopuce¹ pour Objective-C vous sont fournies au format source dans leur intégralité. Une partie de la librairie de bas-niveau est écrite en C pur sucre, mais vous n'aurez à priori pas besoin d'interagir directement avec elle: tout a été fait pour que l'interaction soit le plus simple possible depuis Objective-C.

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

Afin des les garder simples, tous les exemples fournis dans cette documentation sont des applications consoles. Il va de soit que que les fonctionnement des librairies est strictement identiques si vous les intégrez dans une application dotée d'une interface graphique. Vous trouverez sur le blog de Yoctopuce un exemple détaillé² avec des séquences vidéo montrant comment intégrer les fichiers de la librairie à vos projets.

21.1. Contrôle de la fonction Display

Il suffit de quelques lignes de code pour piloter un Yocto-MaxiDisplay-G. Voici le squelette d'un fragment de code Objective-C qui utilise la fonction Display.

```
#import "yocto_api.h"
#import "yocto_display.h"

...
NSError *error;
[YAPI RegisterHub:@"usb": &error]
...
```

¹ www.yoctopuce.com/FR/libraries.php

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

```
// On récupère l'objet représentant le module (ici connecté en local sur USB)
display = [YDisplay FindDisplay:@"YD128G64-123456.display"];

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.h et yocto_display.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_display.h` est nécessaire pour gérer les modules contenant un écran, comme le Yocto-MaxiDisplay-G.

[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.

[Display FindDisplay]

La fonction `[Display FindDisplay]`, permet de retrouver un écran 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-MaxiDisplay-G avec le numéros de série `YD128G64-123456` que vous auriez appelé *"MonModule"* et dont vous auriez nommé la fonction *display* *"MaFonction"*, les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YDisplay *display = [YDisplay FindDisplay:@"YD128G64-123456.display"];
YDisplay *display = [YDisplay FindDisplay:@"YD128G64-123456.MaFonction"];
YDisplay *display = [YDisplay FindDisplay:@"MonModule.display"];
YDisplay *display = [YDisplay FindDisplay:@"MonModule.MaFonction"];
YDisplay *display = [YDisplay FindDisplay:@"MaFonction"];
```

`[YDisplay FindDisplay]` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'écran.

isOnline

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

get_displayLayer

La méthode `get_displayLayer()` de l'objet renvoyé par `YDisplay.FindDisplay` permet récupérer un objet correspondant à une des couches de l'écran. Cet objet fournit toutes les routines graphiques.

Un exemple réel

Lancez Xcode 4.2 et ouvrez le projet exemple correspondant, fourni dans le répertoire **Examples/Doc-GettingStarted-Yocto-MaxiDisplay-G** 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.

UNABLE TO INCLUDE

<http://172.17.17.77/tu/projects/yoctodisplay-128x64-G/public/examples/Objective-C/>

helloworld.m

21.2. Contrôle de la partie module

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

```
#import <Foundation/Foundation.h>
#import "yocto_api.h"

static void usage(const char *exe)
{
    NSLog(@"usage: %s <serial or logical name> [ON/OFF]\n", exe);
    exit(1);
}

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

    @autoreleasepool {
        // Setup the API to use local USB devices
        if([YAPI RegisterHub:@"usb": &error] != YAPI_SUCCESS) {
            NSLog(@"RegisterHub error: %@", [error localizedDescription]);
            return 1;
        }
        if(argc < 2)
            usage(argv[0]);
        NSString *serial_or_name = [NSString stringWithUTF8String:argv[1]];
        // use serial or logical name
        YModule *module = [YModule FindModule:serial_or_name];
        if ([module isOnline]) {
            if (argc > 2) {
                if (strcmp(argv[2], "ON") == 0)
                    [module setBeacon:Y_BEACON_ON];
                else
                    [module setBeacon:Y_BEACON_OFF];
            }
            NSLog(@"serial:      %@\n", [module serialNumber]);
            NSLog(@"logical name: %@\n", [module logicalName]);
            NSLog(@"luminosity:   %d\n", [module luminosity]);
            NSLog(@"beacon:      ");
            if ([module beacon] == Y_BEACON_ON)
                NSLog(@"ON\n");
            else
                NSLog(@"OFF\n");
            NSLog(@"upTime:      %ld sec\n", [module upTime] / 1000);
            NSLog(@"USB current: %d mA\n", [module usbCurrent]);
            NSLog(@"logs:       %@\n", [module get_lastLogs]);
        } else {
            NSLog(@"%% not connected (check identification and USB cable)\n",
                serial_or_name);
        }
        [YAPI FreeAPI];
    }
    return 0;
}
```

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

Modifications des réglages du module

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

courante dans sa mémoire non volatile. Pour cela il faut utiliser la méthode `saveToFlash`. Inversement il est possible de forcer le module à oublier ses réglages courants en utilisant la méthode `revertFromFlash`. Ce petit exemple ci-dessous vous permet changer le nom logique d'un module.

```
#import <Foundation/Foundation.h>
#import "yocto_api.h"

static void usage(const char *exe)
{
    NSLog(@"usage: %s <serial> <newLogicalName>\n", exe);
    exit(1);
}

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

    @autoreleasepool {
        // Setup the API to use local USB devices
        if([YAPI RegisterHub:@"usb" :&error] != YAPI_SUCCESS) {
            NSLog(@"RegisterHub error: %@", [error localizedDescription]);
            return 1;
        }

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

        NSString *serial_or_name = [NSString stringWithUTF8String:argv[1]];
        // use serial or logical name
        YModule *module = [YModule FindModule:serial_or_name];

        if (module.isOnline) {
            if (argc >= 3) {
                NSString *newname = [NSString stringWithUTF8String:argv[2]];
                if (![YAPI CheckLogicalName:newname]) {
                    NSLog(@"Invalid name (%@)\n", newname);
                    usage(argv[0]);
                }
                module.logicalName = newname;
                [module saveToFlash];
            }
            NSLog(@"Current name: %@\n", module.logicalName);
        } else {
            NSLog(@"%@ not connected (check identification and USB cable)\n",
                serial_or_name);
        }
        [YAPI FreeAPI];
    }
    return 0;
}
```

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

```
#import <Foundation/Foundation.h>
#import "yocto_api.h"

int main (int argc, const char * argv[])
{
    NSLog(@"yFirstModule()");
    YModule *module = yFirstModule();
    while (module != NULL) {
        NSLog(@"Module: %@", module);
        module = module->nextModule();
    }
}
```

```

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;
}

```

21.3. 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.

22. 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.

22.1. 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.

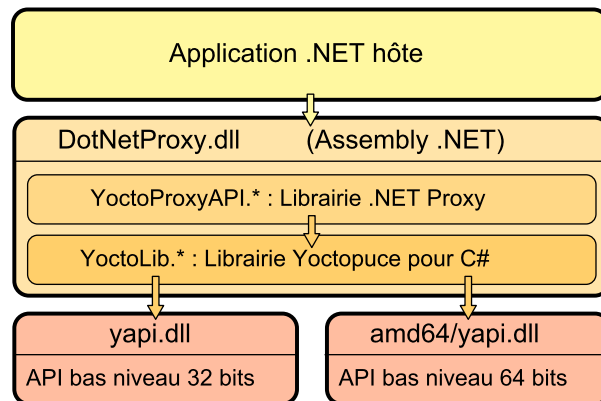
22.2. Assembly .NET

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.

22.3. 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-MaxiDisplay-G avec le numéro de de série *YD128G64-12345* et le nom logique *monModule*. L'URL suivante permettra de connaître l'état du module.

```
http://127.0.0.1:4444/bySerial/YD128G64-12345/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/YD128G64-12345/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/YD128G64-12345/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 display, il suffit de construire l'URL suivante.

```
http://127.0.0.1:4444/bySerial/YD128G64-12345/api/display.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/YD128G64-12345/api/display/logicalName
```

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

```
http://127.0.0.1:4444/bySerial/YD128G64-12345/api/display?logicalName=maFonction
```

Vous trouverez la liste des attributs disponibles pour votre Yocto-MaxiDisplay-G 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/YD128G64-12345/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/YD128G64-12345/dataLogger.json?id=display&utc=1389801080
```

22.4. 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. *bufsize* 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 = 'YD128G64-12345';
    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.
```

Inventaire des modules

Pour procéder à l'inventaire des modules Yoctopuce, deux fonctions de la librairie dynamique sont nécessaires

```
int yapiGetAllDevices(int *buffer, int maxsize, int *neededsize, char *errmsg);
int yapiGetDeviceInfo(int devdesc, yDeviceSt *infos, char *errmsg);
```

La fonction *yapiGetAllDevices* permet d'obtenir la liste des modules connectés sous la forme d'une liste de handles. *buffer* pointe sur un tableau d'entiers 32 bits qui contiendra les handles retournés. *Maxsize* est la taille en bytes du buffer. *neededsize* contiendra au retour la taille nécessaire pour stocker tous les handles. Cela permet d'en déduire le nombre de module connectés, ou si le buffer passé en entrée est trop petit. Le paramètre *errmsg* devra pointer sur un buffer de 255 caractères pour récupérer un éventuel message d'erreur. Ce pointeur peut aussi être à *null*. Cette fonction retourne un entier négatif en cas d'erreur, ou zéro dans le cas contraire.

La fonction *yapiGetDeviceInfo* permet de récupérer les informations relatives à un module à partir de son handle. *devdesc* est un entier 32bit qui représente le module, et qui a été obtenu grâce à *yapiGetAllDevices*. *infos* pointe sur une structure de données dans laquelle sera stocké le résultat. Le format de cette structure est le suivant:

Nom	Type	Taille (bytes)	Description
vendorid	int	4	ID USB de Yoctopuce
deviceid	int	4	ID USB du module
devrelease	int	4	Version du module
nbinbterfaces	int	4	Nombre d'interfaces USB utilisée par le module
manufacturer	char[]	20	Yoctopuce (null terminé)
productname	char[]	28	Modèle (null terminé)
serial	char[]	20	Numéro de série (null terminé)
logicalname	char[]	20	Nom logique (null terminé)
firmware	char[]	22	Version du firmware (null terminé)
beacon	byte	1	Etat de la balise de localisation (0/1)

Le paramètre *errmsg* devra pointer sur un buffer de 255 caractères pour récupérer un éventuel message d'erreur.

Voici un programme d'exemple écrit en pascal qui utilise la DLL *yapi.dll* pour lister les modules connectés.

```
// device description structure
type yDeviceSt = packed record
  vendorid      : word;
  deviceid      : word;
  devrelease    : word;
  nbinbterfaces : word;
  manufacturer  : array [0..19] of ansichar;
  productname   : array [0..27] of ansichar;
  serial        : array [0..19] of ansichar;
  logicalname   : array [0..19] of ansichar;
  firmware      : array [0..21] of ansichar;
  beacon        : byte;
end;

// Dll function import
function yapiInitAPI(mode:integer;
  errmsg : pansichar):integer;cdecl;
  external 'yapi.dll' name 'yapiInitAPI';

function yapiUpdateDeviceList(force:integer;errmsg : pansichar):integer;cdecl;
  external 'yapi.dll' name 'yapiUpdateDeviceList';

function yapiGetAllDevices( buffer:pointer;
  maxsize:integer;
```

```

        var neededsize:integer;
        errmsg : pansichar):integer; cdecl;
        external 'yapi.dll' name 'yapiGetAllDevices';

function  apiGetDeviceInfo(d:integer; var infos:yDeviceSt;
        errmsg : pansichar):integer; cdecl;
        external 'yapi.dll' name 'yapiGetDeviceInfo';

var
errmsgBuffer  : array [0..256] of ansichar;
dataBuffer    : array [0..127] of integer;    // max of 128 USB devices
errmsg,data    : pansichar;
neededsize,i   : integer;
devinfos       : yDeviceSt;

begin
    errmsg := @errmsgBuffer;

    // API initialisation
    if(yapiInitAPI(1,errmsg)<0) then
        begin
            writeln(errmsg);
            halt;
        end;

    // forces a device inventory
    if( yapiUpdateDeviceList(1,errmsg)<0) then
        begin
            writeln(errmsg);
            halt;
        end;

    // loads all device handles into dataBuffer
    if yapiGetAllDevices(@dataBuffer,sizeof(dataBuffer),neededsize,errmsg)<0 then
        begin
            writeln(errmsg);
            halt;
        end;

    // gets device info from each handle
    for i:=0 to neededsize div sizeof(integer)-1 do
        begin
            if (apiGetDeviceInfo(dataBuffer[i], devinfos, errmsg)<0) then
                begin
                    writeln(errmsg);
                    halt;
                end;
            writeln(pansichar(@devinfos.serial)+' ('+pansichar(@devinfos.productname)+' '));
        end;
    end.
end.

```

VB6 et yapi.dll

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

22.5. 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.

23. Programmation avancée

Les chapitres précédents vous ont présenté dans chaque langage disponible les fonctions de programmation de base utilisables avec votre module Yocto-MaxiDisplay-G. 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.

23.1. 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 *Examples/Prog-EventBased*.

Attention: dans la plupart des langages, les callbacks doivent être des procédures globales, et non pas des méthodes. Si vous souhaitez que le callback appelle une méthode d'un objet, définissez votre callback sous la forme d'une procédure globale qui ensuite appellera votre méthode.

24. Référence de l'API de haut niveau

Ce chapitre résume les fonctions de l'API de haut niveau pour commander votre Yocto-MaxiDisplay-G. 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>
dnp	<code>import YoctoProxyAPI.YAPIProxy</code>
cp	<code>#include "yocto_api_proxy.h"</code>
ml	<code>import YoctoProxyAPI.YAPIProxy"</code>
js	<code><script type='text/javascript' src='yocto_api.js'></script></code>
cpp	<code>#include "yocto_api.h"</code>
m	<code>#import "yocto_api.h"</code>
pas	<code>uses yocto_api;</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>
php	<code>require_once('yocto_api.php');</code>
ts	in HTML: <code>import { YAPI, YErrorMsg, YModule, YSensor } from '../dist/esm/yocto_api_browser.js';</code> in Node.js: <code>import { YAPI, YErrorMsg, YModule, YSensor } from 'yoctolib-cjs/yocto_api_nodejs.js';</code>
es	in HTML: <code><script src='../lib/yocto_api.js'></script></code> in node.js: <code>require('yoctolib-es2017/yocto_api.js');</code>
vi	<code>YModule.vi</code>

Fonction globales

YAPI.CheckLogicalName(name)

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

YAPI.ClearHTTPCallbackCacheDir(bool_removeFiles)

Désactive le cache de callback HTTP.

YAPI.DisableExceptions()

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

YAPI.EnableExceptions()

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

YAPI.EnableUSBHost(osContext)

Cette fonction est utilisée uniquement sous Android.

YAPI.FreeAPI()

Attends que toutes les communications en cours avec les modules Yoctopuce soient terminées puis libère les ressources utilisées par la librairie Yoctopuce.

YAPI.GetAPIVersion()

Retourne la version de la librairie Yoctopuce utilisée.

YAPI.GetCacheValidity()

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

YAPI.GetDeviceListValidity()

	Retourne le délai entre chaque énumération forcée des YoctoHubs utilisés.
YAPI.GetDllArchitecture()	Retourne l'architecture du binaire de la librairie de communication Yoctopuce utilisée.
YAPI.GetDllPath()	Retourne l'emplacement sur le disque des librairies Yoctopuce utilisées.
YAPI.GetLog(lastLogLine)	Récupère les messages de logs de la librairie de communication Yoctopuce.
YAPI.GetNetworkTimeout()	Retourne le délai de connexion réseau pour <code>yRegisterHub()</code> et <code>yUpdateDeviceList()</code> .
YAPI.GetTickCount()	Retourne la valeur du compteur monotone de temps (en millisecondes).
YAPI.HandleEvents(errmsg)	Maintient la communication de la librairie avec les modules Yoctopuce.
YAPI.InitAPI(mode, errmsg)	Initialise la librairie de programmation de Yoctopuce explicitement.
YAPI.PreregisterHub(url, errmsg)	Alternative plus tolérante à <code>yRegisterHub()</code> .
YAPI.RegisterDeviceArrivalCallback(arrivalCallback)	Enregistre une fonction de callback qui sera appelée à chaque fois qu'un module est branché.
YAPI.RegisterDeviceRemovalCallback(removalCallback)	Enregistre une fonction de callback qui sera appelée à chaque fois qu'un module est débranché.
YAPI.RegisterHub(url, errmsg)	Configure la librairie Yoctopuce pour utiliser les modules connectés sur une machine donnée.
YAPI.RegisterHubDiscoveryCallback(hubDiscoveryCallback)	Enregistre une fonction de callback qui est appelée chaque fois qu'un hub réseau s'annonce avec un message SSDP.
YAPI.RegisterHubWebsocketCallback(ws, errmsg, authpwd)	Variante de la fonction <code>yRegisterHub()</code> 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.
YAPI.SelectArchitecture(arch)	Sélectionne manuellement l'architecture de la librairie dynamique à utiliser pour accéder à USB.
YAPI.SetCacheValidity(cacheValidityMs)	Change la durée de validité des données chargées par la librairie.
YAPI.SetDelegate(object)	(Objective-C uniquement) Enregistre un objet délégué qui doit se conformer au protocole YDeviceHotPlug.
YAPI.SetDeviceListValidity(deviceListValidity)	Change le délai entre chaque énumération forcée des YoctoHub utilisés.
YAPI.SetHTTPCallbackCacheDir(str_directory)	Active le cache du callback HTTP.
YAPI.SetNetworkTimeout(networkMsTimeout)	Change le délai de connexion réseau pour <code>yRegisterHub()</code> et <code>yUpdateDeviceList()</code> .
YAPI.SetTimeout(callback, ms_timeout, args)	Appelle le callback spécifié après un temps d'attente spécifié.

YAPI.SetUSBPacketAckMs(pktAckDelay)

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

YAPI.Sleep(ms_duration, errmsg)

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

YAPI.TestHub(url, mstimeout, errmsg)

Test si un hub est joignable.

YAPI.TriggerHubDiscovery(errmsg)

Relance une détection des hubs réseau.

YAPI.UnregisterHub(url)

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

YAPI.UpdateDeviceList(errmsg)

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

YAPI.UpdateDeviceList_async(callback, context)

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

YAPI.CheckLogicalName()

YAPI.CheckLogicalName()

YAPI

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

js	function yCheckLogicalName (name)
cpp	bool CheckLogicalName (string name)
m	+(BOOL) CheckLogicalName :(NSString *) name
pas	boolean yCheckLogicalName (name : string): boolean
vb	function CheckLogicalName (ByVal name As String) As Boolean
cs	static bool CheckLogicalName (string name)
java	boolean CheckLogicalName (String name)
uwp	bool CheckLogicalName (string name)
py	CheckLogicalName (name)
php	function CheckLogicalName (\$name)
ts	async CheckLogicalName (name : string): Promise<boolean>
es	async CheckLogicalName (name)

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

Paramètres :

name une chaîne de caractères contenant le nom vérifier.

Retourne :

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

YAPI.ClearHTTPCallbackCacheDir()**YAPI****YAPI.ClearHTTPCallbackCacheDir()**

Désactive le cache de callback HTTP.

```
php function ClearHTTPCallbackCacheDir( $bool_removeFiles)
```

Cette méthode désactive la cache de callback HTTP, et permet également d'en effacer le contenu.

Paramètres :

bool_removeFiles Vrai pour effacer le contenu du répertoire de cache.

Retourne :

nothing.

YAPI.DisableExceptions() YAPI.DisableExceptions()

YAPI

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

js	function yDisableExceptions ()
cpp	void DisableExceptions ()
m	+(void) DisableExceptions
pas	yDisableExceptions ()
vb	procedure DisableExceptions ()
cs	static void DisableExceptions ()
uwp	void DisableExceptions ()
py	DisableExceptions ()
php	function DisableExceptions ()
ts	async DisableExceptions (): Promise<void>
es	async DisableExceptions ()

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

YAPI.EnableExceptions()**YAPI****YAPI.EnableExceptions()**

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

js	function yEnableExceptions ()
cpp	void EnableExceptions ()
m	+(void) EnableExceptions
pas	yEnableExceptions ()
vb	procedure EnableExceptions ()
cs	static void EnableExceptions ()
uwp	void EnableExceptions ()
py	EnableExceptions ()
php	function EnableExceptions ()
ts	async EnableExceptions (): Promise<void>
es	async EnableExceptions ()

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

YAPI.EnableUSBHost() YAPI.EnableUSBHost()

YAPI

Cette fonction est utilisée uniquement sous Android.

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

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

Paramètres :

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

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

Attends que toutes les communications en cours avec les modules Yoctopuce soient terminées puis libère les ressources utilisées par la librairie Yoctopuce.

js	function yFreeAPI ()
cpp	void FreeAPI ()
m	+(void) FreeAPI
pas	yFreeAPI ()
vb	procedure FreeAPI ()
cs	static void FreeAPI ()
java	void FreeAPI ()
uwp	void FreeAPI ()
py	FreeAPI ()
php	function FreeAPI ()
ts	async FreeAPI (): Promise<void>
es	async FreeAPI ()
dnp	static void FreeAPI ()
cp	static void FreeAPI ()

Du point de vue du système d'exploitation, il n'est en général pas nécessaire d'appeler cette fonction puisque que l'OS libèrera automatiquement les ressources allouées dès que le programme sera terminé. Cependant il existe deux situations où vous auriez un intérêt à l'appeler: - Vous désirez libérer tous les blocs de mémoire alloués dynamiquement dans le but d'identifier une source de blocs perdus par exemple. - Votre code envoie des commandes aux modules juste avant de se terminer. Les commandes étant envoyées de manière asynchrone, sans cet appel le programme risquerait de se terminer avant que toutes les commandes n'aient eu le temps de partir. Vous ne devez plus appeler aucune fonction de la librairie après avoir appelé `yFreeAPI()`, sous peine de crash.

YAPI.GetAPIVersion() YAPI.GetAPIVersion()

YAPI

Retourne la version de la librairie Yoctopuce utilisée.

js	function yGetAPIVersion ()
cpp	string GetAPIVersion ()
m	+(NSString*) GetAPIVersion
pas	string yGetAPIVersion (): string
vb	function GetAPIVersion () As String
cs	static String GetAPIVersion ()
java	static String GetAPIVersion ()
uwp	static string GetAPIVersion ()
py	GetAPIVersion ()
php	function GetAPIVersion ()
ts	async GetAPIVersion ()
es	async GetAPIVersion ()
dnp	static string GetAPIVersion ()
cp	static string GetAPIVersion ()

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

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

Retourne :

une chaîne de caractères décrivant la version de la librairie.

YAPI.GetCacheValidity()**YAPI****YAPI.GetCacheValidity()**

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

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

Cette méthode retourne la durée de mise en cache de tous les attributs des fonctions du module. Note: Cette fonction doit être appelée après `yInitAPI`.

Retourne :

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

YAPI.GetDeviceListValidity() YAPI.GetDeviceListValidity()

YAPI

Retourne le délai entre chaque énumération forcée des YoctoHubs utilisés.

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

Note: Cette fonction doit être appelée après `yInitAPI`.

Retourne :

le nombre de secondes entre chaque énumération.

YAPI.GetDllArchitecture() YAPI.GetDllArchitecture()

YAPI

Retourne l'architecture du binaire de la librairie de communication Yoctopuce utilisée.

```
dnf static string GetDllArchitecture( )
```

Pour Windows, l'architecture peut être "Win32" ou "Win64". Pour les machines ARM, l'architecture est "Armf32" ou "Aarch64". Pour les autres machines Linux, l'architecture est "Linux32" ou "Linux64". Pour MacOS, l'architecture est "MacOs32" ou "MacOs64".

Retourne :

une chaîne de caractères décrivant l'architecture du binaire de la librairie de communication Yoctopuce utilisée.

YAPI.GetDllPath()**YAPI****YAPI.GetDllPath()**

Retourne l'emplacement sur le disque des librairies Yoctopuce utilisées.

```
dnsp static string GetDllPath( )
```

Pour les architectures qui demandent l'utilisation de plusieurs DLLs, et en particulier dans le cadre de l'utilisation de la librairie sous forme de .NET Assembly, la chaîne retournée est au format suivant: "DotNetProxy=/...; yapi=/...;", où le premier path correspond à l'emplacement de la DLL .NET Assembly, et le deuxième path correspond à la librairie de communication de bas niveau.

Retourne :

une chaîne de caractères décrivant l'emplacement de la DLL.

YAPI.GetLog()**YAPI****YAPI.GetLog()**

Récupère les messages de logs de la librairie de communication Yoctopuce.

```
dnf static string GetLog( string lastLogLine)
```

```
cp static string GetLog( string lastLogLine)
```

Cette méthode permet de récupérer progressivement les messages de logs, ligne par ligne. La méthode doit être appelée en boucle, jusqu'à ce qu'elle retourne une chaîne vide. Prenez garde à sortir de votre boucle dès qu'une chaîne vide est retournée, car si vous repassez une chaîne vide dans le paramètre `lastLogLine` lors de l'appel suivant, la fonction recommencera à énumérer les messages depuis le plus vieux message disponible.

Paramètres :

lastLogLine Lors du premier appel, passez une chaîne vide. Pour les appels suivants, passez le dernier message retourné par `GetLog()`.

Retourne :

une chaîne de caractères contenant le message de log qui suit immédiatement celui passé en argument, si une telle ligne existe. Si ce n'est pas le cas, lorsque tous les messages ont été retournés, retourne une chaîne vide.

YAPI.GetNetworkTimeout()

YAPI.GetNetworkTimeout()

YAPI

Retourne le délai de connexion réseau pour `yRegisterHub()` et `yUpdateDeviceList()`.

cpp	static int GetNetworkTimeout ()
m	+(int) GetNetworkTimeout
pas	LongInt yGetNetworkTimeout (): LongInt
vb	function GetNetworkTimeout () As Integer
cs	int GetNetworkTimeout ()
java	int GetNetworkTimeout ()
uwp	async Task<int> GetNetworkTimeout ()
py	GetNetworkTimeout ()
php	function GetNetworkTimeout ()
ts	async GetNetworkTimeout (): Promise<number>
es	async GetNetworkTimeout ()
dnp	static int GetNetworkTimeout ()
cp	static int GetNetworkTimeout ()

Ce délai n'affecte que les YoctoHubs et VirtualHub qui sont accessibles par réseau. Par défaut, ce délai est de 20000 millisecondes, mais en fonction de votre réseau il peut être souhaitable de changer ce délai, par exemple, si votre infrastructure réseau utilise une connexion GSM.

Retourne :

le délai de connexion réseau en millisecondes.

YAPI.GetTickCount()**YAPI****YAPI.GetTickCount()**

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

js	function yGetTickCount ()
c++	u64 GetTickCount ()
m	+(u64) GetTickCount
pas	u64 yGetTickCount (): u64
vb	function GetTickCount () As Long
cs	static ulong GetTickCount ()
java	static long GetTickCount ()
uwp	static ulong GetTickCount ()
py	GetTickCount ()
php	function GetTickCount ()
ts	GetTickCount (): number
es	GetTickCount ()

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

Retourne :

un long entier contenant la valeur du compteur de millisecondes.

YAPI.HandleEvents() YAPI.HandleEvents()

YAPI

Maintient la communication de la librairie avec les modules Yoctopuce.

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

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

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

Paramètres :

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

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

YAPI.InitAPI()**YAPI****YAPI.InitAPI()**

Initialise la librairie de programmation de Yoctopuce explicitement.

js	function yInitAPI (mode , errmsg)
cpp	YRETCODE InitAPI (int mode , string errmsg)
m	+(YRETCODE) InitAPI :(int) mode :(NSError**) errmsg
pas	integer yInitAPI (mode : integer, var errmsg : string): integer
vb	function InitAPI (ByVal mode As Integer, ByRef errmsg As String) As Integer
cs	static int InitAPI (int mode , ref string errmsg)
java	int InitAPI (int mode)
uwp	async Task<int> InitAPI (int mode)
py	InitAPI (mode , errmsg =None)
php	function InitAPI (\$mode , & \$errmsg)
ts	async InitAPI (mode : number, errmsg : YErrorMsg): Promise<number>
es	async InitAPI (mode , errmsg)

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

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

Paramètres :

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

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

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

YAPI.PreregisterHub() YAPI.PreregisterHub()

YAPI

Alternative plus tolérante à `yRegisterHub()`.

js	<code>function yPreregisterHub(url, errmsg)</code>
cpp	<code>YRETCODE PreregisterHub(string url, string errmsg)</code>
m	<code>+(YRETCODE) PreregisterHub :(NSString *) url :(NSError**) errmsg</code>
pas	<code>integer yPreregisterHub(url: string, var errmsg: string): integer</code>
vb	<code>function PreregisterHub(ByVal url As String, ByRef errmsg As String) As Integer</code>
cs	<code>static int PreregisterHub(string url, ref string errmsg)</code>
java	<code>int PreregisterHub(String url)</code>
uwp	<code>async Task<int> PreregisterHub(string url)</code>
py	<code>PreregisterHub(url, errmsg=None)</code>
php	<code>function PreregisterHub(\$url, &\$errmsg)</code>
ts	<code>async PreregisterHub(url: string, errmsg: YErrorMsg): Promise<number></code>
es	<code>async PreregisterHub(url, errmsg)</code>
dnp	<code>static string PreregisterHub(string url)</code>
cp	<code>static string PreregisterHub(string url)</code>

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

Paramètres :

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

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

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

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

js	function yRegisterDeviceArrivalCallback (arrivalCallback)
cpp	void RegisterDeviceArrivalCallback (yDeviceUpdateCallback arrivalCallback)
m	+(void) RegisterDeviceArrivalCallback :(yDeviceUpdateCallback) arrivalCallback
pas	yRegisterDeviceArrivalCallback (arrivalCallback : yDeviceUpdateFunc)
vb	procedure RegisterDeviceArrivalCallback (ByVal arrivalCallback As yDeviceUpdateFunc)
cs	static void RegisterDeviceArrivalCallback (yDeviceUpdateFunc arrivalCallback)
java	void RegisterDeviceArrivalCallback (DeviceArrivalCallback arrivalCallback)
uwp	void RegisterDeviceArrivalCallback (DeviceUpdateHandler arrivalCallback)
py	RegisterDeviceArrivalCallback (arrivalCallback)
php	function RegisterDeviceArrivalCallback (\$arrivalCallback)
ts	async RegisterDeviceArrivalCallback (arrivalCallback : YDeviceUpdateCallback null): Promise<void>
es	async RegisterDeviceArrivalCallback (arrivalCallback)

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

Paramètres :

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

YAPI.RegisterDeviceRemovalCallback()

YAPI.RegisterDeviceRemovalCallback()

YAPI

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

js	function yRegisterDeviceRemovalCallback (removalCallback)
cpp	void RegisterDeviceRemovalCallback (yDeviceUpdateCallback removalCallback)
m	+(void) RegisterDeviceRemovalCallback :(yDeviceUpdateCallback) removalCallback
pas	yRegisterDeviceRemovalCallback (removalCallback : yDeviceUpdateFunc)
vb	procedure RegisterDeviceRemovalCallback (ByVal removalCallback As yDeviceUpdateFunc)
cs	static void RegisterDeviceRemovalCallback (yDeviceUpdateFunc removalCallback)
java	void RegisterDeviceRemovalCallback (DeviceRemovalCallback removalCallback)
uwp	void RegisterDeviceRemovalCallback (DeviceUpdateHandler removalCallback)
py	RegisterDeviceRemovalCallback (removalCallback)
php	function RegisterDeviceRemovalCallback (\$removalCallback)
ts	async RegisterDeviceRemovalCallback (removalCallback : YDeviceUpdateCallback null): Promise<void>
es	async RegisterDeviceRemovalCallback (removalCallback)

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

Paramètres :

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

YAPI.RegisterHub()

YAPI

YAPI.RegisterHub()

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

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

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

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

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

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

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

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

http://nom:mot_de_passe@adresse:port

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

Paramètres :

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

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

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

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

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

cpp	void RegisterHubDiscoveryCallback (YHubDiscoveryCallback hubDiscoveryCallback)
m	+(void) RegisterHubDiscoveryCallback : (YHubDiscoveryCallback) hubDiscoveryCallback
pas	yRegisterHubDiscoveryCallback (hubDiscoveryCallback : YHubDiscoveryCallback)
vb	procedure RegisterHubDiscoveryCallback (ByVal hubDiscoveryCallback As YHubDiscoveryCallback)
cs	static void RegisterHubDiscoveryCallback (YHubDiscoveryCallback hubDiscoveryCallback)
java	void RegisterHubDiscoveryCallback (HubDiscoveryCallback hubDiscoveryCallback)
uwp	async Task RegisterHubDiscoveryCallback (HubDiscoveryHandler hubDiscoveryCallback)
py	RegisterHubDiscoveryCallback (hubDiscoveryCallback)
ts	async RegisterHubDiscoveryCallback (hubDiscoveryCallback : YHubDiscoveryCallback): Promise<number>
es	async RegisterHubDiscoveryCallback (hubDiscoveryCallback)

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

Paramètres :

hubDiscoveryCallback une procédure qui prend deux chaîne de caractères en paramètre, le numéro de série et l'URL du hub découvert. Pour supprimer un callback déjà enregistré,

YAPI.RegisterHubWebsocketCallback()**YAPI****YAPI.RegisterHubWebsocketCallback()**

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

Paramètres :

- ws** l'objet WebSocket lié à au callback WebSocket entrant.
- errmsg** une chaîne de caractères passée par référence, dans laquelle sera stocké un éventuel message d'erreur.
- authpwd** le mot de passe optionnel, nécessaire seulement si une authentification WebSocket est configurée sur le hub appelant.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

YAPI.RegisterLogFunction()**YAPI****YAPI.RegisterLogFunction()**

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

cpp	void RegisterLogFunction (yLogFunction logfun)
m	+(void) RegisterLogFunction :(yLogCallback) logfun
pas	yRegisterLogFunction (logfun : yLogFunc)
vb	procedure RegisterLogFunction (ByVal logfun As yLogFunc)
cs	static void RegisterLogFunction (yLogFunc logfun)
java	void RegisterLogFunction (LogCallback logfun)
uwp	void RegisterLogFunction (LogHandler logfun)
py	RegisterLogFunction (logfun)
ts	async RegisterLogFunction (logfun : YLogCallback): Promise<number>
es	async RegisterLogFunction (logfun)

Utile pour déboguer le fonctionnement de l'API.

Paramètres :

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

YAPI.SelectArchitecture() YAPI.SelectArchitecture()

YAPI

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

 **SelectArchitecture(arch)**

Par défaut, la librairie Python détecte automatiquement la version de la librairie dynamique à utiliser pour accéder au port USB. Sous Linux ARM il n'est pas possible de détecter de manière fiable si il s'agit d'une installation Soft float (armel) ou Hard float (armhf). Dans ce cas, il est donc recommandé d'appeler `SelectArchitecture()` avant tout autre appel à la librairie pour forcer l'utilisation d'une architecture spécifiée.

Paramètres :

arch une chaîne de caractère spécifiant l'architecture à utiliser. Les valeurs possibles sont "armhf", "armel", "aarch64", "i386", "x86_64", "32bit", "64bit"

Retourne :

rien.

En cas d'erreur, déclenche une exception.

YAPI.SetCacheValidity()

YAPI.SetCacheValidity()

YAPI

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

cpp	static void SetCacheValidity (u64 cacheValidityMs)
m	+(void) SetCacheValidity : (u64) cacheValidityMs
pas	ySetCacheValidity (cacheValidityMs : u64)
vb	procedure SetCacheValidity (ByVal cacheValidityMs As Long)
cs	void SetCacheValidity (ulong cacheValidityMs)
java	void SetCacheValidity (long cacheValidityMs)
uwp	async Task SetCacheValidity (ulong cacheValidityMs)
py	SetCacheValidity (cacheValidityMs)
php	function SetCacheValidity (\$cacheValidityMs)
ts	async SetCacheValidity (cacheValidityMs : number): Promise<void>
es	async SetCacheValidity (cacheValidityMs)

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour changer cette durée, par exemple dans le but de réduire le trafic réseau ou USB. Ce paramètre n'affecte pas les callbacks de changement de valeur Note: Cette fonction doit être appelée après `yInitAPI`.

Paramètres :

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

YAPI.SetDelegate()**YAPI****YAPI.SetDelegate()**

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

```
m +(void) SetDelegate :(id) object
```

Les méthodes `yDeviceArrival` et `yDeviceRemoval` seront appelées pendant l'exécution de la fonction `yUpdateDeviceList`, que vous devrez appeler régulièrement.

Paramètres :

object un objet qui doit se conformer au protocole `YAPIDelegate`, ou `nil`

YAPI.SetDeviceListValidity()**YAPI****YAPI.SetDeviceListValidity()**

Change le délai entre chaque énumération forcée des YoctoHub utilisés.

cpp	static void SetDeviceListValidity (int deviceListValidity)
m	+(void) SetDeviceListValidity : (int) deviceListValidity
pas	ySetDeviceListValidity (deviceListValidity : LongInt)
vb	procedure SetDeviceListValidity (ByVal deviceListValidity As Integer)
cs	void SetDeviceListValidity (int deviceListValidity)
java	void SetDeviceListValidity (int deviceListValidity)
uwp	async Task SetDeviceListValidity (int deviceListValidity)
py	SetDeviceListValidity (deviceListValidity)
php	function SetDeviceListValidity (\$ deviceListValidity)
ts	async SetDeviceListValidity (deviceListValidity : number): Promise<void>
es	async SetDeviceListValidity (deviceListValidity)

Par défaut la librairie effectue une énumération complète toute les 10 secondes. Pour réduire le trafic réseau, il est possible d'augmenter ce délai. C'est particulièrement utile lors un YoctoHub est connecté à un réseau GSM où le trafic est facturé. Ce paramètre n'affecte pas les modules connectés par USB, ni le fonctionnement des callback de connexion/déconnexion de module. Note: Cette fonction doit être appelée après `yInitAPI`.

Paramètres :

deviceListValidity nombre de secondes entre chaque énumération.

YAPI.SetHTTPCallbackCacheDir() YAPI.SetHTTPCallbackCacheDir()

YAPI

Active le cache du callback HTTP.

```
php function SetHTTPCallbackCacheDir( $str_directory)
```

Le cache du callback HTTP permet de réduire de 50 à 70 % la quantité de données transmise au script PHP. Pour activer le cache, la méthode `ySetHTTPCallbackCacheDir()` doit être appelée avant premier appel à `yRegisterHub()`. Cette méthode prend en paramètre le path du répertoire dans lequel seront stockés les données entre chaque callback. Ce répertoire doit exister et le script PHP doit y avoir les droits d'écriture. Il est recommandé d'utiliser un répertoire qui n'est pas accessible depuis le serveur Web, car la librairie va y stocker des données provenant des modules Yoctopuce.

Note : Cette fonctionnalité est supportée par les YoctoHub et VirtualHub depuis la version 27750

Paramètres :

str_directory le path du répertoire utilisé comme cache.

Retourne :

nothing.

On failure, throws an exception.

YAPI.SetNetworkTimeout()

YAPI.SetNetworkTimeout()

YAPI

Change le délai de connexion réseau pour `yRegisterHub()` et `yUpdateDeviceList()`.

cpp	<code>static void SetNetworkTimeout(int networkMsTimeout)</code>
m	<code>+(void) SetNetworkTimeout : (int) networkMsTimeout</code>
pas	<code>ySetNetworkTimeout(networkMsTimeout: LongInt)</code>
vb	<code>procedure SetNetworkTimeout(ByVal networkMsTimeout As Integer)</code>
cs	<code>void SetNetworkTimeout(int networkMsTimeout)</code>
java	<code>void SetNetworkTimeout(int networkMsTimeout)</code>
uwp	<code>async Task SetNetworkTimeout(int networkMsTimeout)</code>
py	<code>SetNetworkTimeout(networkMsTimeout)</code>
php	<code>function SetNetworkTimeout(\$networkMsTimeout)</code>
ts	<code>async SetNetworkTimeout(networkMsTimeout: number): Promise<void></code>
es	<code>async SetNetworkTimeout(networkMsTimeout)</code>
dnf	<code>static void SetNetworkTimeout(int networkMsTimeout)</code>
cp	<code>static void SetNetworkTimeout(int networkMsTimeout)</code>

Ce délai n'affecte que les YoctoHubs et VirtualHub qui sont accessibles par réseau. Par défaut ce délai est de 20000 millisecondes, mais en fonction de votre réseau il peut être souhaitable de changer ce délai, par exemple si votre infrastructure réseau utilise une connexion GSM.

Paramètres :

networkMsTimeout le délais de connexion réseau en millisecondes.

YAPI.SetTimeout()**YAPI****YAPI.SetTimeout()**

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

js	function ySetTimeout (callback , ms_timeout , args)
ts	SetTimeout (callback : Function, ms_timeout : number): number
es	SetTimeout (callback , ms_timeout , args)

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

Paramètres :

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

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

YAPI.SetUSBPacketAckMs()**YAPI****YAPI.SetUSBPacketAckMs()**

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

```
java void SetUSBPacketAckMs( int pktAckDelay)
```

Cette fonction permet à la librairie de fonctionner même sur les téléphones Android qui perdent des paquets USB. Par défaut, la quittance est désactivée, car elle double le nombre de paquets échangés et donc ralentit sensiblement le fonctionnement de L'API. La quittance des paquets USB ne doit donc être activée que sur des tablette ou des téléphones qui posent problème. Un délais de 50 millisecondes est en général suffisant. En cas de doute contacter le support Yoctopuce. Pour désactiver la quittance des paquets USB, appeler cette fonction avec la valeur 0. Note : Cette fonctionnalité est disponible uniquement sous Android.

Paramètres :

pktAckDelay nombre de ms avant que le module ne renvoie

YAPI.Sleep()**YAPI****YAPI.Sleep()**

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

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

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

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

Paramètres :

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

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

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

YAPI.TestHub()

YAPI.TestHub()

Test si un hub est joignable.

```
YRETCODE TestHub( string url, int mstimeout, string errmsg)
```

```
m +(YRETCODE) TestHub : (NSString*) url
                        : (int) mstimeout
                        : (NSError**) errmsg
```

```
integer yTestHub( url: string,  
                 mstimeout: integer,  
                 var errmsg: string): integer
```

```
vb function TestHub( ByVal url As String,  
                    ByVal mstimeout As Integer,  
                    ByRef errmsg As String) As Integer
```

```
CS static int TestHub( string url, int mstimeout, ref string errmsg)
```

```
java int TestHub( String url, int mstimeout)
```

```
uwp async Task<int> TestHub( string url, uint mstimeout)
```

```
py TestHub( url, mstimeout, errmsg=None)
```

```
function TestHub( $url, $mstimeout, &$errmsg)
```

```
ts async TestHub( url: string, mtimeout: number, errmsg: YErrorMsg): Promise<number>
```

```
es async TestHub( url, mstimeout, errmsg)
```

```
static string TestHub( string url, int mtimeout)
```

```
cp static string TestHub( string url, int mtimeout)
```

Cette méthode n'enregistre pas le hub, elle ne fait que de vérifier que le hub est joignable. Le paramètre url suit les mêmes conventions que la méthode `yRegisterHub`. Cette méthode est utile pour vérifier les paramètres d'authentification d'un hub. Il est possible de forcer la méthode à rendre la main après `mstimeout` millisecondes.

Paramètres :

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

mstimeout le nombre de millisecondes disponible pour tester la connexion.

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

Retourne :

YAPI .SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur retourne un code d'erreur négatif.

YAPI.TriggerHubDiscovery() YAPI.TriggerHubDiscovery()

YAPI

Relance une détection des hubs réseau.

cpp	<code>YRETCODE TriggerHubDiscovery(string errmsg)</code>
m	<code>+(YRETCODE) TriggerHubDiscovery : (NSError**) errmsg</code>
pas	<code>integer yTriggerHubDiscovery(var errmsg: string): integer</code>
vb	<code>function TriggerHubDiscovery(ByRef errmsg As String) As Integer</code>
cs	<code>static int TriggerHubDiscovery(ref string errmsg)</code>
java	<code>int TriggerHubDiscovery()</code>
uwp	<code>Task<int> TriggerHubDiscovery()</code>
py	<code>TriggerHubDiscovery(errmsg=None)</code>
ts	<code>async TriggerHubDiscovery(errmsg: YErrorMsg null): Promise<number></code>
es	<code>async TriggerHubDiscovery(errmsg)</code>

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

Paramètres :

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

Retourne :

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

YAPI.UnregisterHub()**YAPI****YAPI.UnregisterHub()**

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

js	function yUnregisterHub (url)
cpp	void UnregisterHub (string url)
m	+(void) UnregisterHub :(NSString *) url
pas	yUnregisterHub (url : string)
vb	procedure UnregisterHub (ByVal url As String)
cs	static void UnregisterHub (string url)
java	void UnregisterHub (String url)
uwp	async Task UnregisterHub (string url)
py	UnregisterHub (url)
php	function UnregisterHub (\$url)
ts	async UnregisterHub (url : string): Promise<void>
es	async UnregisterHub (url)

Paramètres :

url une chaîne de caractères contenant "usb" ou

YAPI.UpdateDeviceList()**YAPI****YAPI.UpdateDeviceList()**

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

js	function yUpdateDeviceList (errmsg)
c++	YRETCode UpdateDeviceList (string errmsg)
m	+(YRETCode) UpdateDeviceList :(NSError**) errmsg
pas	integer yUpdateDeviceList (var errmsg : string): integer
vb	function UpdateDeviceList (ByRef errmsg As String) As YRETCode
cs	static YRETCode UpdateDeviceList (ref string errmsg)
java	int UpdateDeviceList ()
uwp	async Task<int> UpdateDeviceList ()
py	UpdateDeviceList (errmsg =None)
php	function UpdateDeviceList (&\$errmsg)
ts	async UpdateDeviceList (errmsg : YErrorMsg null): Promise<number>
es	async UpdateDeviceList (errmsg)

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

Cette fonction peut être appelée aussi souvent que désiré, afin de rendre l'application réactive aux événements de hot-plug. Néanmoins la détection des modules étant un processus assez lourd, il est recommandé de ne pas appeler `UpdateDeviceList` plus d'une fois toutes les deux secondes.

Paramètres :

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

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

YAPI.UpdateDeviceList_async()**YAPI****YAPI.UpdateDeviceList_async()**

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

```
js function yUpdateDeviceList_async( callback, context)
```

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

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

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

Paramètres :

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

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

Retourne :

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

24.2. La classe YModule

Interface de contrôle des paramètres généraux des modules Yoctopuce

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

YModule.FindModule(func)

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

YModule.FindModuleInContext(yctx, func)

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

YModule.FirstModule()

Commence l'énumération des modules accessibles par la librairie.

Propriétés des objets YModuleProxy

module→Beacon [modifiable]

état de la balise de localisation.

module→FirmwareRelease [lecture seule]

Version du logiciel embarqué du module.

module→FunctionId [lecture seule]

Identifiant matériel de la *nième* fonction du module.

module→HardwareId [lecture seule]

Identifiant unique du module.

module→IsOnline [lecture seule]

Vérifie si le module est joignable.

module→LogicalName [modifiable]

Nom logique du module.

module→Luminosity [modifiable]

Luminosité des leds informatives du module (valeur entre 0 et 100).

module→**ProductId** [*lecture seule*]

Identifiant USB du module, préprogrammé en usine.

module→**ProductName** [*lecture seule*]

Nom commercial du module, préprogrammé en usine.

module→**ProductRelease** [*lecture seule*]

Numéro uméro de révision du module hardware, préprogrammé en usine.

module→**SerialNumber** [*lecture seule*]

Numéro de série du module, préprogrammé en usine.

Méthodes des objets YModule

module→**checkFirmware**(**path**, **onlynew**)

Teste si le fichier byn est valide pour le module.

module→**clearCache**()

Invalide le cache.

module→**describe**()

Retourne un court texte décrivant le module.

module→**download**(**pathname**)

Télécharge le fichier choisi du module et retourne son contenu.

module→**functionBaseType**(**functionIndex**)

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

module→**functionCount**()

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

module→**functionId**(**functionIndex**)

Retourne l'identifiant matériel de la *nième* fonction du module.

module→**functionName**(**functionIndex**)

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

module→**functionType**(**functionIndex**)

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

module→**functionValue**(**functionIndex**)

Retourne la valeur publiée par la *nième* fonction du module.

module→**get_allSettings**()

Retourne tous les paramètres de configuration du module.

module→**get_beacon**()

Retourne l'état de la balise de localisation.

module→**get_errorMessage**()

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

module→**get_errorType**()

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

module→**get_firmwareRelease**()

Retourne la version du logiciel embarqué du module.

module→**get_functionIds**(**funType**)

Retourne les identifiants matériels des fonctions correspondant au type passé en argument.

module→**get_hardwareId**()

Retourne l'identifiant unique du module.

module→**get_icon2d**()

	Retourne l'icône du module.
module→get_lastLogs()	Retourne une chaîne de caractère contenant les derniers logs du module.
module→get_logicalName()	Retourne le nom logique du module.
module→get_luminosity()	Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).
module→get_parentHub()	Retourne le numéro de série du YoctoHub sur lequel est connecté le module.
module→get_persistentSettings()	Retourne l'état courant des réglages persistents du module.
module→get_productId()	Retourne l'identifiant USB du module, préprogrammé en usine.
module→get_productName()	Retourne le nom commercial du module, préprogrammé en usine.
module→get_productRelease()	Retourne le numéro de révision du module hardware, préprogrammé en usine.
module→get_rebootCountdown()	Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.
module→get_serialNumber()	Retourne le numéro de série du module, préprogrammé en usine.
module→get_subDevices()	Retourne la liste des modules branchés au module courant.
module→get_upTime()	Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module
module→get_url()	Retourne l'URL utilisée pour accéder au module.
module→get_usbCurrent()	Retourne le courant consommé par le module sur le bus USB, en milliampères.
module→get_userData()	Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userData.
module→get_userVar()	Retourne la valeur entière précédemment stockée dans cet attribut.
module→hasFunction(funcId)	Teste la présence d'une fonction pour le module courant.
module→isOnline()	Vérifie si le module est joignable, sans déclencher d'erreur.
module→isOnline_async(callback, context)	Vérifie si le module est joignable, sans déclencher d'erreur.
module→load(msValidity)	Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.
module→load_async(msValidity, callback, context)	Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.
module→log(text)	Ajoute un message arbitraire dans les logs du module.

module→nextModule()

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

module→reboot(secBeforeReboot)

Agende un simple redémarrage du module dans un nombre donné de secondes.

module→registerBeaconCallback(callback)

Enregistre une fonction de callback qui sera appelée à chaque changement d'état de la balise de localisation du module.

module→registerConfigChangeCallback(callback)

Enregistre une fonction de callback qui sera appelée à chaque fois qu'un réglage persistant d'un module est modifié (par exemple changement d'unité de mesure, etc.)

module→registerLogCallback(callback)

Enregistre une fonction de callback qui sera appelée à chaque fois le module émet un message de log.

module→revertFromFlash()

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

module→saveToFlash()

Sauve les réglages courants dans la mémoire non volatile du module.

module→set_allSettings(settings)

Rétablit tous les paramètres du module.

module→set_allSettingsAndFiles(settings)

Rétablit tous les paramètres de configuration et fichiers sur un module.

module→set_beacon(newval)

Allume ou éteint la balise de localisation du module.

module→set_logicalName(newval)

Change le nom logique du module.

module→set_luminosity(newval)

Modifie la luminosité des leds informatives du module.

module→set_userData(data)

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

module→set_userVar(newval)

Stocke une valeur 32 bits dans la mémoire volatile du module.

module→triggerConfigChangeCallback()

Force le déclenchement d'un callback de changement de configuration, afin de vérifier si ils sont disponibles ou pas.

module→triggerFirmwareUpdate(secBeforeReboot)

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

module→updateFirmware(path)

Prepare une mise à jour de firmware du module.

module→updateFirmwareEx(path, force)

Prepare une mise à jour de firmware du module.

module→wait_async(callback, context)

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

YModule.FindModule()

YModule.FindModule()

YModule

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

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

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

Si un appel à la méthode `is_online()` de cet objet renvoie FAUX alors que vous êtes sûr que le module est bien branché, vérifiez que vous n'avez pas oublié d'appeler `registerHub()` à l'initialisation de l'application.

Paramètres :

func une chaîne de caractères contenant soit le numéro de série, soit le nom logique du module désiré

Retourne :

un objet de classe `YModule` qui permet ensuite de contrôler le module ou d'obtenir de plus amples informations sur le module.

YModule.FindModuleInContext()

YModule.FindModuleInContext()

YModule

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

java	static YModule FindModuleInContext (YAPIContext yctx , String func)
uwp	static YModule FindModuleInContext (YAPIContext yctx , string func)
ts	static FindModuleInContext (yctx : YAPIContext, func : string): YModule
es	static FindModuleInContext (yctx , func)

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence le module sans ambiguïté, par exemple `MyDevice.module`.

Retourne :

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

YModule.FirstModule()

YModule.FirstModule()

YModule

Commence l'énumération des modules accessibles par la librairie.

js	function yFirstModule ()
cpp	YModule * FirstModule ()
m	+(YModule*) FirstModule
pas	TYModule yFirstModule (): TYModule
vb	function FirstModule () As YModule
cs	static YModule FirstModule ()
java	static YModule FirstModule ()
uwp	static YModule FirstModule ()
py	FirstModule ()
php	function FirstModule ()
ts	static FirstModule (): YModule null
es	static FirstModule ()

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

Retourne :

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

module→Beacon

YModule

état de la balise de localisation.

`dnsp` `int Beacon`

Modifiable. Allume ou éteint la balise de localisation du module.

module→FirmwareRelease**YModule**

Version du logiciel embarqué du module.

dnp	string FirmwareRelease
-----	-------------------------------

module→FunctionId**YModule**

Identifiant matériel de la *n*ième fonction du module.

dnf

`string FunctionId`

@param functionIndex : l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

module→**HardwareId****YModule**

Identifiant unique du module.

`dnv` `string` **HardwareId**

L'identifiant unique est composé du numéro de série du module suivi de la chaîne ".module".

module→IsOnline**YModule**

Vérifie si le module est joignable.

dnp

bool IsOnline

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

module→LogicalName**YModule**

Nom logique du module.

dnf `string LogicalName`

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

module→Luminosity**YModule**

Luminosité des leds informatives du module (valeur entre 0 et 100).

dnp

int Luminosity

Modifiable. Modifie la luminosité des leds informatives du module. Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

module→**ProductId****YModule**

Identifiant USB du module, préprogrammé en usine.

dnsp

int ProductId

module→**ProductName**

YModule

Nom commercial du module, préprogrammé en usine.

dnf string **ProductName**

module→ProductRelease**YModule**

Numéro uméro de révision du module hardware, préprogrammé en usine.

dnf **int ProductRelease**

La révision originale du retourne la valeur 1, la révision B retourne la valeur 2, etc.

module→**SerialNumber**

YModule

Numéro de série du module, préprogrammé en usine.

dnp `string` **SerialNumber**

module→checkFirmware()**YModule**

Teste si le fichier byn est valide pour le module.

js	function checkFirmware (path , onlynew)
cpp	string checkFirmware (string path , bool onlynew)
m	-(NSString*) checkFirmware : (NSString*) path : (bool) onlynew
pas	string checkFirmware (path : string, onlynew : boolean): string
vb	function checkFirmware (ByVal path As String, ByVal onlynew As Boolean) As String
cs	string checkFirmware (string path , bool onlynew)
java	String checkFirmware (String path , boolean onlynew)
uwp	async Task<string> checkFirmware (string path , bool onlynew)
py	checkFirmware (path , onlynew)
php	function checkFirmware (\$ path , \$ onlynew)
ts	async checkFirmware (path : string, onlynew : boolean): Promise<string>
es	async checkFirmware (path , onlynew)
dnf	string checkFirmware (string path , bool onlynew)
cp	string checkFirmware (string path , bool onlynew)
cmd	YModule target checkFirmware path onlynew

Cette méthode est utile pour vérifier si il est nécessaire de mettre à jour le module avec un nouveau firmware. Il est possible de passer un répertoire qui contient plusieurs fichiers .byn. Dans ce cas cette méthode retourne le path du fichier .byn compatible le plus récent. Si le paramètre `onlynew` est vrai, les firmwares équivalents ou plus anciens que le firmware actuellement installé sont ignorés.

Paramètres :

- path** le path d'un fichier .byn ou d'un répertoire contenant plusieurs fichiers .byn
- onlynew** retourne uniquement les fichiers strictement plus récents

Retourne :

le path du fichier .byn à utiliser, ou une chaîne vide si aucun firmware plus récent n'est disponible. En cas d'erreur, déclenche une exception ou retourne une chaîne de caractère qui commence par "error:".

module→**clearCache()****YModule**

Invalide le cache.

js	function clearCache ()
cpp	void clearCache ()
m	-(void) clearCache
pas	clearCache ()
vb	procedure clearCache ()
cs	void clearCache ()
java	void clearCache ()
py	clearCache ()
php	function clearCache ()
ts	async clearCache (): Promise<void>
es	async clearCache ()

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

module→**describe()****YModule**

Retourne un court texte décrivant le module.

js	function describe ()
cpp	string describe ()
m	-(NSString*) describe
pas	string describe (): string
vb	function describe () As String
cs	string describe ()
java	String describe ()
py	describe ()
php	function describe ()
ts	async describe (): Promise<string>
es	async describe ()

Ce texte peut contenir soit le nom logique du module, soit son numéro de série.

Retourne :

une chaîne de caractères décrivant le module

module→download()**YModule**

Télécharge le fichier choisi du module et retourne son contenu.

js	function download (pathname)
c++	string download (string pathname)
m	-(NSMutableData*) download : (NSString*) pathname
pas	TByteArray download (pathname : string): TByteArray
vb	function download (ByVal pathname As String) As Byte
cs	byte[] download (string pathname)
java	byte[] download (String pathname)
uwp	async Task<byte[]> download (string pathname)
py	download (pathname)
php	function download (\$pathname)
ts	async download (pathname : string): Promise<Uint8Array>
es	async download (pathname)
dnp	byte[] download (string pathname)
cp	string download (string pathname)
cmd	YModule target download pathname

Paramètres :

pathname nom complet du fichier

Retourne :

le contenu du fichier chargé

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

module→functionBaseType()**YModule**

Retourne le type de base de la *n*ième fonction du module.

js	function functionBaseType (functionIndex)
cpp	string functionBaseType (int functionIndex)
pas	string functionBaseType (functionIndex : integer): string
vb	function functionBaseType (ByVal functionIndex As Integer) As String
cs	string functionBaseType (int functionIndex)
java	String functionBaseType (int functionIndex)
py	functionBaseType (functionIndex)
php	function functionBaseType (\$functionIndex)
ts	async functionBaseType (functionIndex : number): Promise<string>
es	async functionBaseType (functionIndex)

Par exemple, le type de base de toutes les fonctions de mesure est "Sensor".

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant au type de base de la fonction

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

module→functionCount()**YModule**

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

js	function functionCount ()
c++	int functionCount ()
m	-(int) functionCount
pas	integer functionCount (): integer
vb	function functionCount () As Integer
cs	int functionCount ()
java	int functionCount ()
py	functionCount ()
php	function functionCount ()
ts	async functionCount (): Promise<number>
es	async functionCount ()

Retourne :

le nombre de fonctions sur le module

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

module→**functionId()****YModule**

Retourne l'identifiant matériel de la *n*ème fonction du module.

js	function functionId (functionIndex)
c++	string functionId (int functionIndex)
m	-(NSString*) functionId : (int) functionIndex
pas	string functionId (functionIndex : integer): string
vb	function functionId (ByVal functionIndex As Integer) As String
cs	string functionId (int functionIndex)
java	String functionId (int functionIndex)
py	functionId (functionIndex)
php	function functionId (\$functionIndex)
ts	async functionId (functionIndex : number): Promise<string>
es	async functionId (functionIndex)

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant à l'identifiant matériel unique de la fonction désirée

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

module→**functionName()****YModule**

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

js	function functionName (functionIndex)
c++	string functionName (int functionIndex)
m	-(NSString*) functionName : (int) functionIndex
pas	string functionName (functionIndex : integer): string
vb	function functionName (ByVal functionIndex As Integer) As String
cs	string functionName (int functionIndex)
java	String functionName (int functionIndex)
py	functionName (functionIndex)
php	function functionName (\$functionIndex)
ts	async functionName (functionIndex : number): Promise<string>
es	async functionName (functionIndex)

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant au nom logique de la fonction désirée

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

module→functionType()**YModule**

Retourne le type de la *n*ème fonction du module.

js	function functionType (functionIndex)
cpp	string functionType (int functionIndex)
pas	string functionType (functionIndex : integer): string
vb	function functionType (ByVal functionIndex As Integer) As String
cs	string functionType (int functionIndex)
java	String functionType (int functionIndex)
py	functionType (functionIndex)
php	function functionType (\$functionIndex)
ts	async functionType (functionIndex : number): Promise<string>
es	async functionType (functionIndex)

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant au type de la fonction

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

module→functionValue()

YModule

Retourne la valeur publiée par la *nième* fonction du module.

js	function functionValue (functionIndex)
c++	string functionValue (int functionIndex)
m	-(NSString*) functionValue : (int) functionIndex
pas	string functionValue (functionIndex : integer): string
vb	function functionValue (ByVal functionIndex As Integer) As String
cs	string functionValue (int functionIndex)
java	String functionValue (int functionIndex)
py	functionValue (functionIndex)
php	function functionValue (\$functionIndex)
ts	async functionValue (functionIndex : number): Promise<string>
es	async functionValue (functionIndex)

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant à la valeur publiée par la fonction désirée

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

module→**get_allSettings()****YModule****module**→**allSettings()**

Retourne tous les paramètres de configuration du module.

js	function get_allSettings ()
cpp	string get_allSettings ()
m	-(NSMutableData*) allSettings
pas	TByteArray get_allSettings (): TByteArray
vb	function get_allSettings () As Byte
cs	byte[] get_allSettings ()
java	byte[] get_allSettings ()
uwp	async Task<byte[]> get_allSettings ()
py	get_allSettings ()
php	function get_allSettings ()
ts	async get_allSettings (): Promise<Uint8Array>
es	async get_allSettings ()
dnp	byte[] get_allSettings ()
cp	string get_allSettings ()
cmd	YModule target get_allSettings

Utile pour sauvgarder les noms logiques, les calibrations et fichiers uploadés d'un module.

Retourne :

un objet binaire avec tous les paramètres

En cas d'erreur, déclenche une exception ou retourne un objet binaire de taille 0.

module→get_beacon()**YModule****module→beacon()**

Retourne l'état de la balise de localisation.

js	function get_beacon ()
cpp	Y_BEACON_enum get_beacon ()
m	-(Y_BEACON_enum) beacon
pas	Integer get_beacon (): Integer
vb	function get_beacon () As Integer
cs	int get_beacon ()
java	int get_beacon ()
uwp	async Task<int> get_beacon ()
py	get_beacon ()
php	function get_beacon ()
ts	async get_beacon (): Promise<YModule_Beacon>
es	async get_beacon ()
dnp	int get_beacon ()
cp	int get_beacon ()
cmd	YModule target get_beacon

Retourne :

soit YModule.BEACON_OFF, soit YModule.BEACON_ON, selon l'état de la balise de localisation

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

module→**get_errorMessage()****YModule****module**→**errorMessage()**

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

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

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

Retourne :

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

module→**get_errorType()****YModule****module**→**errorType()**

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

js	function get_errorType ()
cpp	YRETCODE get_errorType ()
m	-(YRETCODE) errorType
pas	YRETCODE get_errorType (): YRETCODE
vb	function get_errorType () As YRETCODE
cs	YRETCODE get_errorType ()
java	int get_errorType ()
py	get_errorType ()
php	function get_errorType ()
ts	get_errorType (): number
es	get_errorType ()

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

Retourne :

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

module→**get_firmwareRelease()****YModule****module**→**firmwareRelease()**

Retourne la version du logiciel embarqué du module.

js	function get_firmwareRelease ()
cpp	string get_firmwareRelease ()
m	-(NSString*) firmwareRelease
pas	string get_firmwareRelease (): string
vb	function get_firmwareRelease () As String
cs	string get_firmwareRelease ()
java	String get_firmwareRelease ()
uwp	async Task<string> get_firmwareRelease ()
py	get_firmwareRelease ()
php	function get_firmwareRelease ()
ts	async get_firmwareRelease (): Promise<string>
es	async get_firmwareRelease ()
dnp	string get_firmwareRelease ()
cp	string get_firmwareRelease ()
cmd	YModule target get_firmwareRelease

Retourne :

une chaîne de caractères représentant la version du logiciel embarqué du module

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

module→get_functionIds()**YModule****module→functionIds()**

Retourne les identifiants matériels des fonctions correspondant au type passé en argument.

js	function get_functionIds (funType)
cpp	vector<string> get_functionIds (string funType)
m	-(NSMutableArray*) functionIds : (NSString*) funType
pas	TStringArray get_functionIds (funType : string): TStringArray
vb	function get_functionIds (ByVal funType As String) As List
cs	List<string> get_functionIds (string funType)
java	ArrayList<String> get_functionIds (String funType)
uwp	async Task<List<string>> get_functionIds (string funType)
py	get_functionIds (funType)
php	function get_functionIds (\$ funType)
ts	async get_functionIds (funType : string): Promise<string[]>
es	async get_functionIds (funType)
dnp	string[] get_functionIds (string funType)
cp	vector<string> get_functionIds (string funType)
cmd	YModule target get_functionIds funType

Paramètres :

funType Le type de fonction (Relay, LightSensor, Voltage,...)

Retourne :

un tableau de chaînes de caractère.

module→**get_hardwareId()****YModule****module**→**hardwareId()**

Retourne l'identifiant unique du module.

js	function get_hardwareId ()
cpp	string get_hardwareId ()
m	-(NSString*) hardwareId
vb	function get_hardwareId () As String
cs	string get_hardwareId ()
java	String get_hardwareId ()
py	get_hardwareId ()
php	function get_hardwareId ()
ts	async get_hardwareId (): Promise<string>
es	async get_hardwareId ()
dnp	string get_hardwareId ()
cp	string get_hardwareId ()
pas	string get_hardwareId (): string
uwp	async Task<string> get_hardwareId ()
cmd	YModule target get_hardwareId

L'identifiant unique est composé du numéro de série du module suivi de la chaîne ".module".

Retourne :

une chaîne de caractères identifiant la fonction

module→**get_icon2d()****YModule****module**→**icon2d()**

Retourne l'icône du module.

js	function get_icon2d ()
cpp	string get_icon2d ()
m	-(NSData*) icon2d
pas	TByteArray get_icon2d (): TByteArray
vb	function get_icon2d () As Byte
cs	byte[] get_icon2d ()
java	byte[] get_icon2d ()
uwp	async Task<byte[]> get_icon2d ()
py	get_icon2d ()
php	function get_icon2d ()
ts	async get_icon2d (): Promise<Uint8Array>
es	async get_icon2d ()
dnp	byte[] get_icon2d ()
cp	string get_icon2d ()
cmd	YModule target get_icon2d

L'icone est au format PNG et a une taille maximale de 1536 octets.

Retourne :

un buffer binaire contenant l'icone, au format png. En cas d'erreur, déclenche une exception ou retourne YAPI_INVALID_STRING.

module→**get_lastLogs()****YModule****module**→**lastLogs()**

Retourne une chaîne de caractère contenant les derniers logs du module.

js	function get_lastLogs ()
cpp	string get_lastLogs ()
m	-(NSString*) lastLogs
pas	string get_lastLogs (): string
vb	function get_lastLogs () As String
cs	string get_lastLogs ()
java	String get_lastLogs ()
uwp	async Task<string> get_lastLogs ()
py	get_lastLogs ()
php	function get_lastLogs ()
ts	async get_lastLogs (): Promise<string>
es	async get_lastLogs ()
dnp	string get_lastLogs ()
cp	string get_lastLogs ()
cmd	YModule target get_lastLogs

Cette méthode retourne les derniers logs qui sont encore stocké dans le module.

Retourne :

une chaîne de caractère contenant les derniers logs du module. En cas d'erreur, déclenche une exception ou retourne YAPI_INVALID_STRING.

module→**get_logicalName()****YModule****module**→**logicalName()**

Retourne le nom logique du module.

js	function get_logicalName ()
cpp	string get_logicalName ()
m	-(NSString*) logicalName
pas	string get_logicalName (): string
vb	function get_logicalName () As String
cs	string get_logicalName ()
java	String get_logicalName ()
uwp	async Task<string> get_logicalName ()
py	get_logicalName ()
php	function get_logicalName ()
ts	async get_logicalName (): Promise<string>
es	async get_logicalName ()
dnp	string get_logicalName ()
cp	string get_logicalName ()
cmd	YModule target get_logicalName

Retourne :

une chaîne de caractères représentant le nom logique du module

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

module→**get_luminosity()****YModule****module**→**luminosity()**

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

js	function get_luminosity ()
cpp	int get_luminosity ()
m	-(int) luminosity
pas	LongInt get_luminosity (): LongInt
vb	function get_luminosity () As Integer
cs	int get_luminosity ()
java	int get_luminosity ()
uwp	async Task<int> get_luminosity ()
py	get_luminosity ()
php	function get_luminosity ()
ts	async get_luminosity (): Promise<number>
es	async get_luminosity ()
dnp	int get_luminosity ()
cp	int get_luminosity ()
cmd	YModule target get_luminosity

Retourne :

un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

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

module→**get_parentHub()****YModule****module**→**parentHub()**

Retourne le numéro de série du YoctoHub sur lequel est connecté le module.

js	function get_parentHub ()
c++	string get_parentHub ()
m	-(NSString*) parentHub
pas	string get_parentHub (): string
vb	function get_parentHub () As String
cs	string get_parentHub ()
java	String get_parentHub ()
uwp	async Task<string> get_parentHub ()
py	get_parentHub ()
php	function get_parentHub ()
ts	async get_parentHub (): Promise<string>
es	async get_parentHub ()
dnp	string get_parentHub ()
cp	string get_parentHub ()
cmd	YModule target get_parentHub

Si le module est connecté par USB, ou si le module est le YoctoHub racine, une chaîne vide est retournée.

Retourne :

une chaîne de caractères contenant le numéro de série du YoctoHub, ou une chaîne vide.

module→**get_persistentSettings()****YModule****module**→**persistentSettings()**

Retourne l'état courant des réglages persistents du module.

js	function get_persistentSettings ()
cpp	Y_PERSISTENTSETTINGS_enum get_persistentSettings ()
m	-(Y_PERSISTENTSETTINGS_enum) persistentSettings
pas	Integer get_persistentSettings (): Integer
vb	function get_persistentSettings () As Integer
cs	int get_persistentSettings ()
java	int get_persistentSettings ()
uwp	async Task<int> get_persistentSettings ()
py	get_persistentSettings ()
php	function get_persistentSettings ()
ts	async get_persistentSettings (): Promise<YModule_PersistentSettings>
es	async get_persistentSettings ()
dnp	int get_persistentSettings ()
cp	int get_persistentSettings ()
cmd	YModule target get_persistentSettings

Retourne :

une valeur parmi YModule.PERSISTENTSETTINGS_LOADED, YModule.PERSISTENTSETTINGS_SAVED et YModule.PERSISTENTSETTINGS_MODIFIED représentant l'état courant des réglages persistents du module

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

module→get_productId()

YModule

module→productId()

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

js	function get_productId ()
c++	int get_productId ()
m	-(int) productId
pas	LongInt get_productId (): LongInt
vb	function get_productId () As Integer
cs	int get_productId ()
java	int get_productId ()
uwp	async Task<int> get_productId ()
py	get_productId ()
php	function get_productId ()
ts	async get_productId (): Promise<number>
es	async get_productId ()
dnp	int get_productId ()
cp	int get_productId ()
cmd	YModule target get_productId

Retourne :

un entier représentant l'identifiant USB du module, préprogrammé en usine

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

module→**get_productName()****YModule****module**→**productName()**

Retourne le nom commercial du module, préprogrammé en usine.

js	function get_productName ()
cpp	string get_productName ()
m	-(NSString*) productName
pas	string get_productName (): string
vb	function get_productName () As String
cs	string get_productName ()
java	String get_productName ()
uwp	async Task<string> get_productName ()
py	get_productName ()
php	function get_productName ()
ts	async get_productName (): Promise<string>
es	async get_productName ()
dnp	string get_productName ()
cp	string get_productName ()
cmd	YModule target get_productName

Retourne :

une chaîne de caractères représentant le nom commercial du module, préprogrammé en usine

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

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

Retourne le numéro uméro de révision du module hardware, préprogrammé en usine.

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

La révision originale du retourne la valeur 1, la révision B retourne la valeur 2, etc.

Retourne :

un entier représentant le numéro uméro de révision du module hardware, préprogrammé en usine

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

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

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

js	function get_rebootCountdown ()
cpp	int get_rebootCountdown ()
m	-(int) rebootCountdown
pas	LongInt get_rebootCountdown (): LongInt
vb	function get_rebootCountdown () As Integer
cs	int get_rebootCountdown ()
java	int get_rebootCountdown ()
uwp	async Task<int> get_rebootCountdown ()
py	get_rebootCountdown ()
php	function get_rebootCountdown ()
ts	async get_rebootCountdown (): Promise<number>
es	async get_rebootCountdown ()
dnp	int get_rebootCountdown ()
cp	int get_rebootCountdown ()
cmd	YModule target get_rebootCountdown

Retourne :

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

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

module→**get_serialNumber()****YModule****module**→**serialNumber()**

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

js	function get_serialNumber ()
cpp	string get_serialNumber ()
m	-(NSString*) serialNumber
pas	string get_serialNumber (): string
vb	function get_serialNumber () As String
cs	string get_serialNumber ()
java	String get_serialNumber ()
uwp	async Task<string> get_serialNumber ()
py	get_serialNumber ()
php	function get_serialNumber ()
ts	async get_serialNumber (): Promise<string>
es	async get_serialNumber ()
dnp	string get_serialNumber ()
cp	string get_serialNumber ()
cmd	YModule target get_serialNumber

Retourne :

une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine

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

module→**get_subDevices()****YModule****module**→**subDevices()**

Retourne la liste des modules branchés au module courant.

js	function get_subDevices ()
cpp	vector<string> get_subDevices ()
m	-(NSMutableArray*) subDevices
pas	TStringArray get_subDevices (): TStringArray
vb	function get_subDevices () As List
cs	List<string> get_subDevices ()
java	ArrayList<String> get_subDevices ()
uwp	async Task<List<string>> get_subDevices ()
py	get_subDevices ()
php	function get_subDevices ()
ts	async get_subDevices (): Promise<string[]
es	async get_subDevices ()
dnp	string[] get_subDevices ()
cp	vector<string> get_subDevices ()
cmd	YModule target get_subDevices

Cette fonction n'est pertinente que lorsqu'elle appelée pour un YoctoHub ou pour le VirtualHub. Dans le cas contraire, un tableau vide est retourné.

Retourne :

un tableau de chaînes de caractères contenant les numéros de série des sous-modules connectés au module

module→**get_upTime()****YModule****module**→**upTime()**

Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module

js	function get_upTime ()
c++	s64 get_upTime ()
m	-(s64) upTime
pas	int64 get_upTime (): int64
vb	function get_upTime () As Long
cs	long get_upTime ()
java	long get_upTime ()
uwp	async Task<long> get_upTime ()
py	get_upTime ()
php	function get_upTime ()
ts	async get_upTime (): Promise<number>
es	async get_upTime ()
dnp	long get_upTime ()
cp	s64 get_upTime ()
cmd	YModule target get_upTime

Retourne :

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

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

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

Retourne l'URL utilisée pour accéder au module.

js	function get_url ()
cpp	string get_url ()
m	-(NSString*) url
pas	string get_url (): string
vb	function get_url () As String
cs	string get_url ()
java	String get_url ()
uwp	async Task<string> get_url ()
py	get_url ()
php	function get_url ()
ts	async get_url (): Promise<string>
es	async get_url ()
dnp	string get_url ()
cp	string get_url ()
cmd	YModule target get_url

Si le module est connecté par USB la chaîne de caractère 'usb' est retournée.

Retourne :

une chaîne de caractère contenant l'URL du module.

module→get_usbCurrent()

YModule

module→usbCurrent()

Retourne le courant consommé par le module sur le bus USB, en milliampères.

js	function get_usbCurrent ()
c++	int get_usbCurrent ()
m	-(int) usbCurrent
pas	LongInt get_usbCurrent (): LongInt
vb	function get_usbCurrent () As Integer
cs	int get_usbCurrent ()
java	int get_usbCurrent ()
uwp	async Task<int> get_usbCurrent ()
py	get_usbCurrent ()
php	function get_usbCurrent ()
ts	async get_usbCurrent (): Promise<number>
es	async get_usbCurrent ()
dnp	int get_usbCurrent ()
cp	int get_usbCurrent ()
cmd	YModule target get_usbCurrent

Retourne :

un entier représentant le courant consommé par le module sur le bus USB, en milliampères

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

module→**get_userData()****YModule****module**→**userData()**

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

js	<code>function get_userData()</code>
cpp	<code>void * get_userData()</code>
m	<code>-(id) userData</code>
pas	<code>Tobject get_userData(): Tobject</code>
vb	<code>function get_userData() As Object</code>
cs	<code>object get_userData()</code>
java	<code>Object get_userData()</code>
py	<code>get_userData()</code>
php	<code>function get_userData()</code>
ts	<code>async get_userData(): Promise<object null></code>
es	<code>async get_userData()</code>

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

Retourne :

l'objet stocké précédemment par l'appelant.

module→**get_userVar()****YModule****module**→**userVar()**

Retourne la valeur entière précédemment stockée dans cet attribut.

js	function get_userVar ()
cpp	int get_userVar ()
m	-(int) userVar
pas	LongInt get_userVar (): LongInt
vb	function get_userVar () As Integer
cs	int get_userVar ()
java	int get_userVar ()
uwp	async Task<int> get_userVar ()
py	get_userVar ()
php	function get_userVar ()
ts	async get_userVar (): Promise<number>
es	async get_userVar ()
dnp	int get_userVar ()
cp	int get_userVar ()
cmd	YModule target get_userVar

Au démarrage du module (ou après un redémarrage), la valeur est toujours zéro.

Retourne :

un entier représentant la valeur entière précédemment stockée dans cet attribut

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

module→hasFunction()**YModule**

Teste la présence d'une fonction pour le module courant.

js	function hasFunction (funcId)
cpp	bool hasFunction (string funcId)
m	-(bool) hasFunction : (NSString*) funcId
pas	boolean hasFunction (funcId : string): boolean
vb	function hasFunction (ByVal funcId As String) As Boolean
cs	bool hasFunction (string funcId)
java	boolean hasFunction (String funcId)
uwp	async Task<bool> hasFunction (string funcId)
py	hasFunction (funcId)
php	function hasFunction (\$funcId)
ts	async hasFunction (funcId : string): Promise<boolean>
es	async hasFunction (funcId)
dnf	bool hasFunction (string funcId)
cp	bool hasFunction (string funcId)
cmd	YModule target hasFunction funcId

La méthode prend en paramètre l'identifiant de la fonction (relay1, voltage2,...) et retourne un booléen.

Paramètres :

funcId identifiant matériel de la fonction

Retourne :

vrai si le module inclut la fonction demandée

module→**isOnline()****YModule**

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

js	function isOnline ()
cpp	bool isOnline ()
m	-(BOOL) isOnline
pas	boolean isOnline (): boolean
vb	function isOnline () As Boolean
cs	bool isOnline ()
java	boolean isOnline ()
py	isOnline ()
php	function isOnline ()
ts	async isOnline (): Promise<boolean>
es	async isOnline ()
dnp	bool isOnline ()
cp	bool isOnline ()

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

Retourne :

true si le module est joignable, false sinon

module→isOnline_async()**YModule**

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

```
js function isOnline_async( callback, context)
```

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

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

Paramètres :

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

Retourne :

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

module→load()**YModule**

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

js	function load (msValidity)
c++	YRETCODE load (int msValidity)
m	-(YRETCODE) load : (u64) msValidity
pas	YRETCODE load (msValidity : u64): YRETCODE
vb	function load (ByVal msValidity As Long) As YRETCODE
cs	YRETCODE load (ulong msValidity)
java	int load (long msValidity)
py	load (msValidity)
php	function load (\$msValidity)
ts	async load (msValidity : number): Promise<number>
es	async load (msValidity)

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

Paramètres :

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

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

module→load_async()**YModule**

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

```
js function load_async( msValidity, callback, context)
```

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

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

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet module concerné et le code d'erreur (ou `YAPI . SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

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

module→log()

YModule

Ajoute un message arbitraire dans les logs du module.

js	function log (text)
c++	int log (string text)
m	-(int) log : (NSString*) text
pas	LongInt log (text : string): LongInt
vb	function log (ByVal text As String) As Integer
cs	int log (string text)
java	int log (String text)
uwp	async Task<int> log (string text)
py	log (text)
php	function log (\$text)
ts	async log (text : string): Promise<number>
es	async log (text)
dnp	int log (string text)
cp	int log (string text)
cmd	YModule target log text

Cette fonction est utile en particulier pour tracer l'exécution de callbacks HTTP. Si un saut de ligne est désiré après le message, il doit être inclus dans la chaîne de caractère.

Paramètres :

text le message à ajouter aux logs du module.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

module→nextModule()**YModule**

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

js	function nextModule ()
cpp	YModule * nextModule ()
m	-(nullable YModule*) nextModule
pas	TYModule nextModule (): TYModule
vb	function nextModule () As YModule
cs	YModule nextModule ()
java	YModule nextModule ()
uwp	YModule nextModule ()
py	nextModule ()
php	function nextModule ()
ts	nextModule (): YModule null
es	nextModule ()

Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les modules sont retournés. Si vous souhaitez retrouver un module spécifique, utilisez `Module.findModule()` avec un `hardwareID` ou un nom logique.

Retourne :

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

module→reboot()**YModule**

Agende un simple redémarrage du module dans un nombre donné de secondes.

js	function reboot (secBeforeReboot)
c++	int reboot (int secBeforeReboot)
m	-(int) reboot : (int) secBeforeReboot
pas	LongInt reboot (secBeforeReboot : LongInt): LongInt
vb	function reboot (ByVal secBeforeReboot As Integer) As Integer
cs	int reboot (int secBeforeReboot)
java	int reboot (int secBeforeReboot)
uwp	async Task<int> reboot (int secBeforeReboot)
py	reboot (secBeforeReboot)
php	function reboot (\$ secBeforeReboot)
ts	async reboot (secBeforeReboot : number): Promise<number>
es	async reboot (secBeforeReboot)
dnp	int reboot (int secBeforeReboot)
cp	int reboot (int secBeforeReboot)
cmd	YModule target reboot secBeforeReboot

Paramètres :

secBeforeReboot nombre de secondes avant de redémarrer

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

module→registerBeaconCallback()**YModule**

Enregistre une fonction de callback qui sera appelée à chaque changement d'état de la balise de localisation du module.

js	function registerBeaconCallback (callback)
cpp	int registerBeaconCallback (YModuleBeaconCallback callback)
m	-(int) registerBeaconCallback : (YModuleBeaconCallback _Nullable) callback
pas	LongInt registerBeaconCallback (callback : TYModuleBeaconCallback): LongInt
vb	function registerBeaconCallback (ByVal callback As YModuleBeaconCallback) As Integer
cs	int registerBeaconCallback (BeaconCallback callback)
java	int registerBeaconCallback (BeaconCallback callback)
uwp	async Task<int> registerBeaconCallback (BeaconCallback callback)
py	registerBeaconCallback (callback)
php	function registerBeaconCallback (\$callback)
ts	async registerBeaconCallback (callback : YModuleBeaconCallback null): Promise<number>
es	async registerBeaconCallback (callback)

La fonction de callback doit accepter deux arguments: l'objet YModule dont la balise a changé, et un entier représentant l'état de la balise de localisation.

Paramètres :

callback la fonction de callback à rappeler, ou null

module→registerConfigChangeCallback()**YModule**

Enregistre une fonction de callback qui sera appelée à chaque fois qu'un réglage persistant d'un module est modifié (par exemple changement d'unité de mesure, etc.)

js	function registerConfigChangeCallback (callback)
cpp	int registerConfigChangeCallback (YModuleConfigChangeCallback callback)
m	-(int) registerConfigChangeCallback : (YModuleConfigChangeCallback _Nullable) callback
pas	LongInt registerConfigChangeCallback (callback : TYModuleConfigChangeCallback): LongInt
vb	function registerConfigChangeCallback (ByVal callback As YModuleConfigChangeCallback) As Integer
cs	int registerConfigChangeCallback (ConfigChangeCallback callback)
java	int registerConfigChangeCallback (ConfigChangeCallback callback)
uwp	async Task<int> registerConfigChangeCallback (ConfigChangeCallback callback)
py	registerConfigChangeCallback (callback)
php	function registerConfigChangeCallback (\$callback)
ts	async registerConfigChangeCallback (callback : YModuleConfigChangeCallback null): Promise<number>
es	async registerConfigChangeCallback (callback)

Paramètres :

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

module→registerLogCallback()**YModule**

Enregistre une fonction de callback qui sera appelée à chaque fois le module émet un message de log.

js	function registerLogCallback (callback)
cpp	int registerLogCallback (YModuleLogCallback callback)
m	-(int) registerLogCallback : (YModuleLogCallback _Nullable) callback
pas	LongInt registerLogCallback (callback : TYModuleLogCallback): LongInt
vb	function registerLogCallback (ByVal callback As YModuleLogCallback) As Integer
cs	int registerLogCallback (LogCallback callback)
java	int registerLogCallback (LogCallback callback)
uwp	async Task<int> registerLogCallback (LogCallback callback)
py	registerLogCallback (callback)
php	function registerLogCallback (\$callback)
ts	async registerLogCallback (callback : YModuleLogCallback null): Promise<number>
es	async registerLogCallback (callback)

Utile pour déboguer le fonctionnement d'un module Yoctopuce.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet module qui a produit un log, un chaîne de caractère qui contient le log

module→revertFromFlash()**YModule**

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

js	function revertFromFlash ()
cpp	int revertFromFlash ()
m	-(int) revertFromFlash
pas	LongInt revertFromFlash (): LongInt
vb	function revertFromFlash () As Integer
cs	int revertFromFlash ()
java	int revertFromFlash ()
uwp	async Task<int> revertFromFlash ()
py	revertFromFlash ()
php	function revertFromFlash ()
ts	async revertFromFlash (): Promise<number>
es	async revertFromFlash ()
dnp	int revertFromFlash ()
cp	int revertFromFlash ()
cmd	YModule target revertFromFlash

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

module→saveToFlash()**YModule**

Sauve les réglages courants dans la mémoire non volatile du module.

js	function saveToFlash ()
cpp	int saveToFlash ()
m	-(int) saveToFlash
pas	LongInt saveToFlash (): LongInt
vb	function saveToFlash () As Integer
cs	int saveToFlash ()
java	int saveToFlash ()
uwp	async Task<int> saveToFlash ()
py	saveToFlash ()
php	function saveToFlash ()
ts	async saveToFlash (): Promise<number>
es	async saveToFlash ()
dnp	int saveToFlash ()
cp	int saveToFlash ()
cmd	YModule target saveToFlash

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

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

module→set_allSettings()

YModule

module→setAllSettings()

Rétablit tous les paramètres du module.

js	function set_allSettings (settings)
c++	int set_allSettings (string settings)
m	-(int) setAllSettings : (NSData*) settings
pas	LongInt set_allSettings (settings : TByteArray): LongInt
vb	procedure set_allSettings (ByVal settings As Byte())
cs	int set_allSettings (byte[] settings)
java	int set_allSettings (byte[] settings)
uwp	async Task<int> set_allSettings (byte[] settings)
py	set_allSettings (settings)
php	function set_allSettings (\$ settings)
ts	async set_allSettings (settings : Uint8Array): Promise<number>
es	async set_allSettings (settings)
dnp	int set_allSettings (byte[] settings)
cp	int set_allSettings (string settings)
cmd	YModule target set_allSettings settings

Utile pour restorer les noms logiques et les calibrations du module depuis une sauvgarde. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si les réglages doivent être préservés.

Paramètres :

settings un objet binaire avec tous les paramètres

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

module→**set_allSettingsAndFiles()****YModule****module**→**setAllSettingsAndFiles()**

Rétablit tous les paramètres de configuration et fichiers sur un module.

js	function set_allSettingsAndFiles (settings)
cpp	int set_allSettingsAndFiles (string settings)
m	-(int) setAllSettingsAndFiles : (NSData*) settings
pas	LongInt set_allSettingsAndFiles (settings : TByteArray): LongInt
vb	procedure set_allSettingsAndFiles (ByVal settings As Byte())
cs	int set_allSettingsAndFiles (byte[] settings)
java	int set_allSettingsAndFiles (byte[] settings)
uwp	async Task<int> set_allSettingsAndFiles (byte[] settings)
py	set_allSettingsAndFiles (settings)
php	function set_allSettingsAndFiles (\$ settings)
ts	async set_allSettingsAndFiles (settings : Uint8Array): Promise<number>
es	async set_allSettingsAndFiles (settings)
dnp	int set_allSettingsAndFiles (byte[] settings)
cp	int set_allSettingsAndFiles (string settings)
cmd	YModule target set_allSettingsAndFiles settings

Cette méthode est utile pour récupérer les noms logiques, les calibrations, les fichiers uploadés, etc. du module depuis une sauvegarde. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si les réglages doivent être préservés.

Paramètres :

settings un buffer binaire avec tous les paramètres

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

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

Allume ou éteint la balise de localisation du module.

js	function set_beacon (newval)
c++	int set_beacon (Y_BEACON_enum newval)
m	-(int) setBeacon : (Y_BEACON_enum) newval
pas	integer set_beacon (newval : Integer): integer
vb	function set_beacon (ByVal newval As Integer) As Integer
cs	int set_beacon (int newval)
java	int set_beacon (int newval)
uwp	async Task<int> set_beacon (int newval)
py	set_beacon (newval)
php	function set_beacon (\$ newval)
ts	async set_beacon (newval : YModule_Beacon): Promise<number>
es	async set_beacon (newval)
dnp	int set_beacon (int newval)
cp	int set_beacon (int newval)
cmd	YModule target set_beacon newval

Paramètres :

newval soit YModule.BEACON_OFF, soit YModule.BEACON_ON

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

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

Change le nom logique du module.

js	function set_logicalName (newval)
cpp	int set_logicalName (string newval)
m	-(int) setLogicalName : (NSString*) newval
pas	integer set_logicalName (newval : string): integer
vb	function set_logicalName (ByVal newval As String) As Integer
cs	int set_logicalName (string newval)
java	int set_logicalName (String newval)
uwp	async Task<int> set_logicalName (string newval)
py	set_logicalName (newval)
php	function set_logicalName (\$ newval)
ts	async set_logicalName (newval : string): Promise<number>
es	async set_logicalName (newval)
dnp	int set_logicalName (string newval)
cp	int set_logicalName (string newval)
cmd	YModule target set_logicalName newval

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

Paramètres :

newval une chaîne de caractères

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

module→set_luminosity()

YModule

module→setLuminosity()

Modifie la luminosité des leds informatives du module.

js	function set_luminosity (newval)
c++	int set_luminosity (int newval)
m	-(int) setLuminosity : (int) newval
pas	integer set_luminosity (newval : LongInt): integer
vb	function set_luminosity (ByVal newval As Integer) As Integer
cs	int set_luminosity (int newval)
java	int set_luminosity (int newval)
uwp	async Task<int> set_luminosity (int newval)
py	set_luminosity (newval)
php	function set_luminosity (\$newval)
ts	async set_luminosity (newval : number): Promise<number>
es	async set_luminosity (newval)
dnp	int set_luminosity (int newval)
cp	int set_luminosity (int newval)
cmd	YModule target set_luminosity newval

Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la luminosité des leds informatives du module

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

module→**set_userdata()****YModule****module**→**setUserData()**

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

js	<code>function set_userdata(data)</code>
cpp	<code>void set_userdata(void * data)</code>
m	<code>-(void) setUserData : (id) data</code>
pas	<code>set_userdata(data: Tobject)</code>
vb	<code>procedure set_userdata(ByVal data As Object)</code>
cs	<code>void set_userdata(object data)</code>
java	<code>void set_userdata(Object data)</code>
py	<code>set_userdata(data)</code>
php	<code>function set_userdata(\$data)</code>
ts	<code>async set_userdata(data: object null): Promise<void></code>
es	<code>async set_userdata(data)</code>

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

Paramètres :

data objet quelconque à mémoriser

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

Stocke une valeur 32 bits dans la mémoire volatile du module.

js	function set_userVar (newval)
c++	int set_userVar (int newval)
m	-(int) setUserVar : (int) newval
pas	integer set_userVar (newval : LongInt): integer
vb	function set_userVar (ByVal newval As Integer) As Integer
cs	int set_userVar (int newval)
java	int set_userVar (int newval)
uwp	async Task<int> set_userVar (int newval)
py	set_userVar (newval)
php	function set_userVar (\$newval)
ts	async set_userVar (newval : number): Promise<number>
es	async set_userVar (newval)
dnp	int set_userVar (int newval)
cp	int set_userVar (int newval)
cmd	YModule target set_userVar newval

Cet attribut est à la disposition du programmeur pour y stocker par exemple une variable d'état. Au démarrage du module (ou après un redémarrage), la valeur est toujours zéro.

Paramètres :

newval un entier

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

module→triggerConfigChangeCallback()**YModule**

Force le déclenchement d'un callback de changement de configuration, afin de vérifier si ils sont disponibles ou pas.

js	function triggerConfigChangeCallback ()
cpp	int triggerConfigChangeCallback ()
m	-(int) triggerConfigChangeCallback
pas	LongInt triggerConfigChangeCallback (): LongInt
vb	function triggerConfigChangeCallback () As Integer
cs	int triggerConfigChangeCallback ()
java	int triggerConfigChangeCallback ()
uwp	async Task<int> triggerConfigChangeCallback ()
py	triggerConfigChangeCallback ()
php	function triggerConfigChangeCallback ()
ts	async triggerConfigChangeCallback (): Promise<number>
es	async triggerConfigChangeCallback ()
dnf	int triggerConfigChangeCallback ()
cp	int triggerConfigChangeCallback ()
cmd	YModule target triggerConfigChangeCallback

module→triggerFirmwareUpdate()

YModule

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

js	function triggerFirmwareUpdate (secBeforeReboot)
c++	int triggerFirmwareUpdate (int secBeforeReboot)
m	-(int) triggerFirmwareUpdate : (int) secBeforeReboot
pas	LongInt triggerFirmwareUpdate (secBeforeReboot : LongInt): LongInt
vb	function triggerFirmwareUpdate (ByVal secBeforeReboot As Integer) As Integer
cs	int triggerFirmwareUpdate (int secBeforeReboot)
java	int triggerFirmwareUpdate (int secBeforeReboot)
uwp	async Task<int> triggerFirmwareUpdate (int secBeforeReboot)
py	triggerFirmwareUpdate (secBeforeReboot)
php	function triggerFirmwareUpdate (\$secBeforeReboot)
ts	async triggerFirmwareUpdate (secBeforeReboot : number): Promise<number>
es	async triggerFirmwareUpdate (secBeforeReboot)
dnp	int triggerFirmwareUpdate (int secBeforeReboot)
cp	int triggerFirmwareUpdate (int secBeforeReboot)
cmd	YModule target triggerFirmwareUpdate secBeforeReboot

Paramètres :

secBeforeReboot nombre de secondes avant de redémarrer

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

module→updateFirmware()**YModule**

Prepare une mise à jour de firmware du module.

js	function updateFirmware (path)
cpp	YFirmwareUpdate updateFirmware (string path)
m	-(YFirmwareUpdate*) updateFirmware : (NSString*) path
pas	TYFirmwareUpdate updateFirmware (path : string): TYFirmwareUpdate
vb	function updateFirmware (ByVal path As String) As YFirmwareUpdate
cs	YFirmwareUpdate updateFirmware (string path)
java	YFirmwareUpdate updateFirmware (String path)
uwp	async Task<YFirmwareUpdate> updateFirmware (string path)
py	updateFirmware (path)
php	function updateFirmware (\$path)
ts	async updateFirmware (path : string): Promise<YFirmwareUpdate>
es	async updateFirmware (path)
dnp	YFirmwareUpdateProxy updateFirmware (string path)
cp	YFirmwareUpdateProxy* updateFirmware (string path)
cmd	YModule target updateFirmware path

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

Paramètres :

path le path du fichier .byn à utiliser

Retourne :

un objet YFirmwareUpdate ou NULL en cas d'erreur

module→updateFirmwareEx()

YModule

Prepare une mise à jour de firmware du module.

js	function updateFirmwareEx (path , force)
c++	YFirmwareUpdate updateFirmwareEx (string path , bool force)
m	-(YFirmwareUpdate*) updateFirmwareEx : (NSString*) path : (bool) force
pas	TYFirmwareUpdate updateFirmwareEx (path : string, force : boolean): TYFirmwareUpdate
vb	function updateFirmwareEx (ByVal path As String, ByVal force As Boolean) As YFirmwareUpdate
cs	YFirmwareUpdate updateFirmwareEx (string path , bool force)
java	YFirmwareUpdate updateFirmwareEx (String path , boolean force)
uwp	async Task<YFirmwareUpdate> updateFirmwareEx (string path , bool force)
py	updateFirmwareEx (path , force)
php	function updateFirmwareEx (\$path , \$force)
ts	async updateFirmwareEx (path : string, force : boolean): Promise<YFirmwareUpdate>
es	async updateFirmwareEx (path , force)
dnp	YFirmwareUpdateProxy updateFirmwareEx (string path , bool force)
cp	YFirmwareUpdateProxy* updateFirmwareEx (string path , bool force)
cmd	YModule target updateFirmwareEx path force

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

Paramètres :

path le path du fichier .byn à utiliser

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

Retourne :

un objet YFirmwareUpdate ou NULL en cas d'erreur

module→**wait_async()****YModule**

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

js	<code>function wait_async(callback, context)</code>
ts	<code>wait_async(callback: Function, context: object)</code>
es	<code>wait_async(callback, context)</code>

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées. La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

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

Retourne :

rien du tout.

24.3. La classe YDisplay

Interface pour interagir avec les écrans, disponibles par exemple dans le Yocto-Display, le Yocto-MaxiDisplay, le Yocto-MaxiDisplay-G et le Yocto-MiniDisplay

La classe `YDisplay` permet de piloter les écrans Yoctopuce. L'interface de contrôle des écrans Yoctopuce est conçue pour afficher facilement des informations et des images. Le module est capable de gérer seul la superposition de plusieurs couches graphiques, qui peuvent être dessinées individuellement, sans affichage immédiat, puis librement positionnées sur l'écran. Il est aussi capable de rejouer des séquences de commandes pré-enregistrées (animations).

Pour afficher du contenu sur l'écran, il faut utiliser la méthode `display.get_displayLayer` pour récupérer la (ou les) couche(s) graphique(s) dans lesquelles vous voulez dessiner, puis écrire dedans à l'aide des méthodes de la classe `YDisplayLayer`.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_display.js'></script></code>
cpp	<code>#include "yocto_display.h"</code>
m	<code>#import "yocto_display.h"</code>
pas	<code>uses yocto_display;</code>
vb	<code>yocto_display.vb</code>
cs	<code>yocto_display.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YDisplay;</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YDisplay;</code>
py	<code>from yocto_display import *</code>
php	<code>require_once('yocto_display.php');</code>
ts	in HTML: <code>import { YDisplay } from '../dist/esm/yocto_display.js';</code> in Node.js: <code>import { YDisplay } from 'yoctolib-cjs/yocto_display.js';</code>
es	in HTML: <code><script src='../lib/yocto_display.js'></script></code> in node.js: <code>require('yoctolib-es2017/yocto_display.js');</code>
dnp	<code>import YoctoProxyAPI.YDisplayProxy</code>
cp	<code>#include "yocto_display_proxy.h"</code>
vi	<code>YDisplay.vi</code>
ml	<code>import YoctoProxyAPI.YDisplayProxy</code>

Fonction globales

YDisplay.FindDisplay(func)

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

YDisplay.FindDisplayInContext(yctx, func)

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

YDisplay.FirstDisplay()

Commence l'énumération des écrans accessibles par la librairie.

YDisplay.FirstDisplayInContext(yctx)

Commence l'énumération des écrans accessibles par la librairie.

YDisplay.GetSimilarFunctions()

Enumère toutes les fonctions de type Display disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

Propriétés des objets YDisplayProxy

display→AdvertisedValue *[lecture seule]*

Courte chaîne de caractères représentant l'état courant de la fonction.	
display→Brightness <i>[modifiable]</i> Luminosité des leds informatives du module (valeur entre 0 et 100).	
display→DisplayHeight <i>[lecture seule]</i> Hauteur de l'écran, en pixels.	
display→DisplayType <i>[lecture seule]</i> Type de l'écran: monochrome, niveaux de gris ou couleur.	
display→DisplayWidth <i>[lecture seule]</i> Largeur de l'écran, en pixels.	
display→FriendlyName <i>[lecture seule]</i> Identifiant global de la fonction au format NOM_MODULE . NOM_FONCTION.	
display→FunctionId <i>[lecture seule]</i> Identifiant matériel de l'ecran, sans référence au module.	
display→HardwareId <i>[lecture seule]</i> Identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.	
display→IsOnline <i>[lecture seule]</i> Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.	
display→LayerCount <i>[lecture seule]</i> Nombre des couches affichables disponibles.	
display→LayerHeight <i>[lecture seule]</i> Hauteur des couches affichables, en pixels.	
display→LayerWidth <i>[lecture seule]</i> Largeur des couches affichables, en pixels.	
display→LogicalName <i>[modifiable]</i> Nom logique de la fonction.	
display→Orientation <i>[modifiable]</i> Orientation sélectionnée pour l'écran.	
display→SerialNumber <i>[lecture seule]</i> Numéro de série du module, préprogrammé en usine.	
display→StartupSeq <i>[modifiable]</i> Nom de la séquence à jouer à la mise sous tension de l'écran.	
Méthodes des objets YDisplay	
display→clearCache() Invalide le cache.	
display→copyLayerContent(srcLayerId, dstLayerId) Copie le contenu d'un couche d'affichage vers une autre couche.	
display→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de l'ecran au format TYPE (NAME) = SERIAL . FUNCTIONID.	
display→fade(brightness, duration) Change la luminosité de l'écran en douceur, pour produire un effet de fade-in ou fade-out.	
display→get_advertisedValue() Retourne la valeur courante de l'ecran (pas plus de 6 caractères).	
display→get_brightness() Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).	

display→get_displayHeight()

Retourne la hauteur de l'écran, en pixels.

display→get_displayLayer(layerId)

Retourne un objet YDisplayLayer utilisable pour dessiner sur la couche d'affichage correspondante.

display→get_displayType()

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

display→get_displayWidth()

Retourne la largeur de l'écran, en pixels.

display→get_enabled()

Retourne vrai si le l'écran est alimenté, faux sinon.

display→get_errorMessage()

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

display→get_errorType()

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

display→get_friendlyName()

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

display→get_functionDescriptor()

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

display→get_functionId()

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

display→get_hardwareId()

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

display→get_layerCount()

Retourne le nombre des couches affichables disponibles.

display→get_layerHeight()

Retourne la hauteur des couches affichables, en pixels.

display→get_layerWidth()

Retourne la largeur des couches affichables, en pixels.

display→get_logicalName()

Retourne le nom logique de l'écran.

display→get_module()

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

display→get_module_async(callback, context)

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

display→get_orientation()

Retourne l'orientation sélectionnée pour l'écran.

display→get_serialNumber()

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

display→get_startupSeq()

Retourne le nom de la séquence à jouer à la mise sous tension de l'écran.

display→get_userData()

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

display→isOnline()

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

display→isOnline_async(callback, context)

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

display→isReadOnly()

Test si la fonction est en lecture seule.

display→load(msValidity)

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

display→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.

display→load_async(msValidity, callback, context)

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

display→muteValueCallbacks()

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

display→newSequence()

Enclanche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

display→nextDisplay()

Continue l'énumération des écrans commencée à l'aide de `yFirstDisplay()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les écrans sont retournés.

display→pauseSequence(delay_ms)

Attend pour la durée spécifiée (en millisecondes) avant de jouer les commandes suivantes de la séquence active.

display→playSequence(sequenceName)

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes `newSequence()` et `saveSequence()`.

display→registerValueCallback(callback)

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

display→resetAll()

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

display→saveSequence(sequenceName)

Termine l'enregistrement d'une séquence et la sauvegarde sur la mémoire interne de l'écran, sous le nom choisi.

display→set_brightness(newval)

Modifie la luminosité de l'écran.

display→set_enabled(newval)

Modifie l'état d'activité de l'écran.

display→set_logicalName(newval)

Modifie le nom logique de l'écran.

display→set_orientation(newval)

Modifie l'orientation de l'écran.

display→set_startupSeq(newval)

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

display→set_userData(data)

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

display→stopSequence(sequenceName)

Arrête immédiatement la séquence d'affichage actuellement jouée sur l'écran.

display→swapLayerContent(layerIdA, layerIdB)

Permute le contenu de deux couches d'affichage.

display→unmuteValueCallbacks()

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

display→upload(pathname, content)

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

display→wait_async(callback, context)

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

YDisplay.FindDisplay()**YDisplay****YDisplay.FindDisplay()**

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

js	function yFindDisplay (func)
c++	YDisplay* FindDisplay (string func)
m	+(YDisplay*) FindDisplay : (NSString*) func
pas	TYDisplay yFindDisplay (func : string): TYDisplay
vb	function FindDisplay (ByVal func As String) As YDisplay
cs	static YDisplay FindDisplay (string func)
java	static YDisplay FindDisplay (String func)
uwp	static YDisplay FindDisplay (string func)
py	FindDisplay (func)
php	function FindDisplay (\$func)
ts	static FindDisplay (func : string): YDisplay
es	static FindDisplay (func)
dnp	static YDisplayProxy FindDisplay (string func)
cp	static YDisplayProxy * FindDisplay (string func)

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Si un appel à la méthode `is_online()` de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler `registerHub()` à l'initialisation de l'application.

Paramètres :

func une chaîne de caractères qui référence l'écran sans ambiguïté, par exemple `YD128X32.display`.

Retourne :

un objet de classe `YDisplay` qui permet ensuite de contrôler l'écran.

YDisplay.FindDisplayInContext()

YDisplay.FindDisplayInContext()

YDisplay

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

java	<code>static YDisplay FindDisplayInContext(YAPIContext yctx, String func)</code>
uwp	<code>static YDisplay FindDisplayInContext(YAPIContext yctx, string func)</code>
ts	<code>static FindDisplayInContext(yctx: YAPIContext, func: string): YDisplay</code>
es	<code>static FindDisplayInContext(yctx, func)</code>

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'écran sans ambiguïté, par exemple `YD128X32.display`.

Retourne :

un objet de classe `YDisplay` qui permet ensuite de contrôler l'écran.

YDisplay.FirstDisplay()

YDisplay.FirstDisplay()

YDisplay

Commence l'énumération des écrans accessibles par la librairie.

js	function yFirstDisplay ()
c++	YDisplay * FirstDisplay ()
m	+(YDisplay*) FirstDisplay
pas	TYDisplay yFirstDisplay (): TYDisplay
vb	function FirstDisplay () As YDisplay
cs	static YDisplay FirstDisplay ()
java	static YDisplay FirstDisplay ()
uwp	static YDisplay FirstDisplay ()
py	FirstDisplay ()
php	function FirstDisplay ()
ts	static FirstDisplay (): YDisplay null
es	static FirstDisplay ()

Utiliser la fonction `YDisplay.nextDisplay()` pour itérer sur les autres écrans.

Retourne :

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

YDisplay.FirstDisplayInContext()

YDisplay.FirstDisplayInContext()

YDisplay

Commence l'énumération des écrans accessibles par la librairie.

java	static YDisplay FirstDisplayInContext (YAPIContext yctx)
uwp	static YDisplay FirstDisplayInContext (YAPIContext yctx)
ts	static FirstDisplayInContext (yctx : YAPIContext): YDisplay null
es	static FirstDisplayInContext (yctx)

Utiliser la fonction `YDisplay.nextDisplay()` pour itérer sur les autres écrans.

Paramètres :

yctx un contexte YAPI.

Retourne :

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

YDisplay.GetSimilarFunctions()**YDisplay****YDisplay.GetSimilarFunctions()**

Enumère toutes les fonctions de type Display disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

dnp	static new string[] GetSimilarFunctions ()
-----	--

cp	static vector<string> GetSimilarFunctions ()
----	--

Chaque chaîne retournée peut être passée en argument à la méthode `YDisplay.FindDisplay` pour obtenir un objet permettant d'interagir avec le module correspondant.

Retourne :

un tableau de chaînes de caractères, contenant les identifiants matériels de chaque fonction disponible trouvée.

display→AdvertisedValue

YDisplay

Courte chaîne de caractères représentant l'état courant de la fonction.

dnp string **AdvertisedValue**

display→Brightness**YDisplay**

Luminosité des leds informatives du module (valeur entre 0 et 100).

dnsp **int Brightness**

Modifiable. Modifie la luminosité de l'écran. Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

display→DisplayHeight

YDisplay

Hauteur de l'écran, en pixels.

dnp `int DisplayHeight`

display→**DisplayType****YDisplay**

Type de l'écran: monochrome, niveaux de gris ou couleur.

`dnsp` [int DisplayType](#)

display→DisplayWidth

YDisplay

Largeur de l'écran, en pixels.

dnp `int DisplayWidth`

display→FriendlyName**YDisplay**

Identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

`dnf` `string FriendlyName`

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

display→**FunctionId****YDisplay**

Identifiant matériel de l'écran, sans référence au module.

dnf

`string` **FunctionId**

Par exemple `relay1`.

display→**HardwareId****YDisplay**

Identifiant matériel unique de la fonction au format `SERIAL.FUNCTIONID`.

dnf[string **HardwareId**](#)

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

display→IsOnline**YDisplay**

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

dnp

bool IsOnline

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

display→LayerCount**YDisplay**

Nombre des couches affichables disponibles.

`int` **LayerCount**

display→LayerHeight**YDisplay**

Hauteur des couches affichables, en pixels.

dnp

int LayerHeight

display→**LayerWidth****YDisplay**

Largeur des couches affichables, en pixels.

dnp

int LayerWidth

display→LogicalName**YDisplay**

Nom logique de la fonction.

dnp

`string LogicalName`

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

display→Orientation**YDisplay**

Orientation sélectionnée pour l'écran.

dnp

int Orientation

Modifiable. Modifie l'orientation de l'écran. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

display→**SerialNumber**

YDisplay

Numéro de série du module, préprogrammé en usine.

`dnsp` string **SerialNumber**

display→**StartupSeq****YDisplay**

Nom de la séquence à jouer à la mise sous tension de l'écran.

dnf `string` **StartupSeq**

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

display→clearCache()**YDisplay**

Invalide le cache.

js	function clearCache ()
cpp	void clearCache ()
m	-(void) clearCache
pas	clearCache ()
vb	procedure clearCache ()
cs	void clearCache ()
java	void clearCache ()
py	clearCache ()
php	function clearCache ()
ts	async clearCache (): Promise<void>
es	async clearCache ()

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

display→copyLayerContent()**YDisplay**

Copie le contenu d'une couche d'affichage vers une autre couche.

js	function copyLayerContent (srcLayerId , dstLayerId)
cpp	int copyLayerContent (int srcLayerId , int dstLayerId)
m	-(int) copyLayerContent : (int) srcLayerId : (int) dstLayerId
pas	LongInt copyLayerContent (srcLayerId : LongInt, dstLayerId : LongInt): LongInt
vb	function copyLayerContent (ByVal srcLayerId As Integer, ByVal dstLayerId As Integer) As Integer
cs	int copyLayerContent (int srcLayerId , int dstLayerId)
java	int copyLayerContent (int srcLayerId , int dstLayerId)
uwp	async Task<int> copyLayerContent (int srcLayerId , int dstLayerId)
py	copyLayerContent (srcLayerId , dstLayerId)
php	function copyLayerContent (\$ srcLayerId , \$ dstLayerId)
ts	async copyLayerContent (srcLayerId : number, dstLayerId : number): Promise<number>
es	async copyLayerContent (srcLayerId , dstLayerId)
dnp	int copyLayerContent (int srcLayerId , int dstLayerId)
cp	int copyLayerContent (int srcLayerId , int dstLayerId)
cmd	YDisplay target copyLayerContent srcLayerId dstLayerId

La couleur et la transparence de tous les pixels de la couche de destination sont changés pour correspondre à la couche source. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

Paramètres :

srcLayerId l'identifiant de la couche d'origine (un chiffre parmi 0..layerCount-1)

dstLayerId l'identifiant de la couche de destination (un chiffre parmi 0..layerCount-1)

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→describe()**YDisplay**

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

js	function describe ()
cpp	string describe ()
m	-(NSString*) describe
pas	string describe (): string
vb	function describe () As String
cs	string describe ()
java	String describe ()
py	describe ()
php	function describe ()
ts	async describe (): Promise<string>
es	async describe ()

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

Retourne :

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

display→fade()**YDisplay**

Change la luminosité de l'écran en douceur, pour produire un effet de fade-in ou fade-out.

js	function fade (brightness , duration)
cpp	int fade (int brightness , int duration)
m	-(int) fade : (int) brightness : (int) duration
pas	LongInt fade (brightness : LongInt, duration : LongInt): LongInt
vb	function fade (ByVal brightness As Integer, ByVal duration As Integer) As Integer
cs	int fade (int brightness , int duration)
java	int fade (int brightness , int duration)
uwp	async Task<int> fade (int brightness , int duration)
py	fade (brightness , duration)
php	function fade (\$brightness , \$duration)
ts	async fade (brightness : number, duration : number): Promise<number>
es	async fade (brightness , duration)
dnp	int fade (int brightness , int duration)
cp	int fade (int brightness , int duration)
cmd	YDisplay target fade brightness duration

Paramètres :

brightness nouvelle valeur de luminosité de l'écran

duration durée en millisecondes de la transition.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→**get_advertisedValue()****YDisplay****display**→**advertisedValue()**

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

js	function get_advertisedValue ()
cpp	string get_advertisedValue ()
m	-(NSString*) advertisedValue
pas	string get_advertisedValue (): string
vb	function get_advertisedValue () As String
cs	string get_advertisedValue ()
java	String get_advertisedValue ()
uwp	async Task<string> get_advertisedValue ()
py	get_advertisedValue ()
php	function get_advertisedValue ()
ts	async get_advertisedValue (): Promise<string>
es	async get_advertisedValue ()
dnp	string get_advertisedValue ()
cp	string get_advertisedValue ()
cmd	YDisplay target get_advertisedValue

Retourne :

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

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

display→**get_brightness()****YDisplay****display**→**brightness()**

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

js	function get_brightness ()
cpp	int get_brightness ()
m	-(int) brightness
pas	LongInt get_brightness (): LongInt
vb	function get_brightness () As Integer
cs	int get_brightness ()
java	int get_brightness ()
uwp	async Task<int> get_brightness ()
py	get_brightness ()
php	function get_brightness ()
ts	async get_brightness (): Promise<number>
es	async get_brightness ()
dnp	int get_brightness ()
cp	int get_brightness ()
cmd	YDisplay target get_brightness

Retourne :

un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

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

display→**get_displayHeight()****YDisplay****display**→**displayHeight()**

Retourne la hauteur de l'écran, en pixels.

js	function get_displayHeight ()
cpp	int get_displayHeight ()
m	-(int) displayHeight
pas	LongInt get_displayHeight (): LongInt
vb	function get_displayHeight () As Integer
cs	int get_displayHeight ()
java	int get_displayHeight ()
uwp	async Task<int> get_displayHeight ()
py	get_displayHeight ()
php	function get_displayHeight ()
ts	async get_displayHeight (): Promise<number>
es	async get_displayHeight ()
dnp	int get_displayHeight ()
cp	int get_displayHeight ()
cmd	YDisplay target get_displayHeight

Retourne :

un entier représentant la hauteur de l'écran, en pixels

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

display→**get_displayLayer()****YDisplay****display**→**displayLayer()**

Retourne un objet YDisplayLayer utilisable pour dessiner sur la couche d'affichage correspondante.

js	function get_displayLayer (layerId)
cpp	YDisplayLayer* get_displayLayer (int layerId)
m	-(YDisplayLayer*) displayLayer : (int) layerId
pas	TYDisplayLayer get_displayLayer (layerId : LongInt): TYDisplayLayer
vb	function get_displayLayer (ByVal layerId As Integer) As YDisplayLayer
cs	YDisplayLayer get_displayLayer (int layerId)
java	YDisplayLayer get_displayLayer (int layerId)
uwp	async Task<YDisplayLayer> get_displayLayer (int layerId)
py	get_displayLayer (layerId)
php	function get_displayLayer (\$layerId)
ts	async get_displayLayer (layerId : number): Promise<YDisplayLayer>
es	async get_displayLayer (layerId)
dnp	YDisplayLayerProxy get_displayLayer (int layerId)
cp	YDisplayLayerProxy* get_displayLayer (int layerId)

Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

Paramètres :

layerId l'identifiant de la couche d'affichage désirée (un chiffre parmi 0..layerCount-1)

Retourne :

un objet YDisplayLayer

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

display→**get_displayType()****display**→**displayType()**

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

js	function get_displayType ()
cpp	Y_DISPLAYTYPE_enum get_displayType ()
m	-(Y_DISPLAYTYPE_enum) displayType
pas	Integer get_displayType (): Integer
vb	function get_displayType () As Integer
cs	int get_displayType ()
java	int get_displayType ()
uwp	async Task<int> get_displayType ()
py	get_displayType ()
php	function get_displayType ()
ts	async get_displayType (): Promise<YDisplay_DisplayType>
es	async get_displayType ()
dnp	int get_displayType ()
cp	int get_displayType ()
cmd	YDisplay target get_displayType

Retourne :

une valeur parmi YDisplay.DISPLAYTYPE_MONO, YDisplay.DISPLAYTYPE_GRAY et YDisplay.DISPLAYTYPE_RGB représentant le type de l'écran: monochrome, niveaux de gris ou couleur

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

display→**get_displayWidth()****YDisplay****display**→**displayWidth()**

Retourne la largeur de l'écran, en pixels.

js	function get_displayWidth ()
cpp	int get_displayWidth ()
m	-(int) displayWidth
pas	LongInt get_displayWidth (): LongInt
vb	function get_displayWidth () As Integer
cs	int get_displayWidth ()
java	int get_displayWidth ()
uwp	async Task<int> get_displayWidth ()
py	get_displayWidth ()
php	function get_displayWidth ()
ts	async get_displayWidth (): Promise<number>
es	async get_displayWidth ()
dnp	int get_displayWidth ()
cp	int get_displayWidth ()
cmd	YDisplay target get_displayWidth

Retourne :

un entier représentant la largeur de l'écran, en pixels

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

display→**get_enabled()****display**→**enabled()**

Retourne vrai si le l'ecran est alimenté, faux sinon.

js	function get_enabled ()
cpp	Y_ENABLED_enum get_enabled ()
m	-(Y_ENABLED_enum) enabled
pas	Integer get_enabled (): Integer
vb	function get_enabled () As Integer
cs	int get_enabled ()
java	int get_enabled ()
uwp	async Task<int> get_enabled ()
py	get_enabled ()
php	function get_enabled ()
ts	async get_enabled (): Promise<YDisplay_Enabled>
es	async get_enabled ()
dnp	int get_enabled ()
cp	int get_enabled ()
cmd	YDisplay target get_enabled

Retourne :

soit YDisplay.ENABLED_FALSE, soit YDisplay.ENABLED_TRUE, selon vrai si le l'ecran est alimenté, faux sinon

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

display→**get_errorMessage()****YDisplay****display**→**errorMessage()**

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

js	<code>function get_errorMessage()</code>
cpp	<code>string get_errorMessage()</code>
m	<code>-(NSString*) errorMessage</code>
pas	<code>string get_errorMessage(): string</code>
vb	<code>function get_errorMessage() As String</code>
cs	<code>string get_errorMessage()</code>
java	<code>String get_errorMessage()</code>
py	<code>get_errorMessage()</code>
php	<code>function get_errorMessage()</code>
ts	<code>get_errorMessage(): string</code>
es	<code>get_errorMessage()</code>

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

Retourne :

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

display→get_errorType()**display→errorType()**

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

js	function get_errorType ()
cpp	YRETCODE get_errorType ()
m	-(YRETCODE) errorType
pas	YRETCODE get_errorType (): YRETCODE
vb	function get_errorType () As YRETCODE
cs	YRETCODE get_errorType ()
java	int get_errorType ()
py	get_errorType ()
php	function get_errorType ()
ts	get_errorType (): number
es	get_errorType ()

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

Retourne :

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

display→**get_friendlyName()****YDisplay****display**→**friendlyName()**

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

js	function get_friendlyName ()
cpp	string get_friendlyName ()
m	-(NSString*) friendlyName
cs	string get_friendlyName ()
java	String get_friendlyName ()
py	get_friendlyName ()
php	function get_friendlyName ()
ts	async get_friendlyName (): Promise<string>
es	async get_friendlyName ()
dnp	string get_friendlyName ()
cp	string get_friendlyName ()

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

Retourne :

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

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

display→get_functionDescriptor()**YDisplay****display→functionDescriptor()**

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

js	function get_functionDescriptor ()
cpp	YFUN_DESCR get_functionDescriptor ()
m	-(YFUN_DESCR) functionDescriptor
pas	YFUN_DESCR get_functionDescriptor (): YFUN_DESCR
vb	function get_functionDescriptor () As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor ()
java	String get_functionDescriptor ()
py	get_functionDescriptor ()
php	function get_functionDescriptor ()
ts	async get_functionDescriptor (): Promise<string>
es	async get_functionDescriptor ()

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

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y\$CLASSNAME\$.FUNCTIONDESCRIPTOR_INVALID

display→**get_functionId()****YDisplay****display**→**functionId()**

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

js	function get_functionId ()
cpp	string get_functionId ()
m	-(NSString*) functionId
vb	function get_functionId () As String
cs	string get_functionId ()
java	String get_functionId ()
py	get_functionId ()
php	function get_functionId ()
ts	async get_functionId (): Promise<string>
es	async get_functionId ()
dnp	string get_functionId ()
cp	string get_functionId ()

Par exemple `relay1`.

Retourne :

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

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

display→**get_hardwareId()****YDisplay****display**→**hardwareId()**

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

js	function get_hardwareId ()
cpp	string get_hardwareId ()
m	-(NSString*) hardwareId
vb	function get_hardwareId () As String
cs	string get_hardwareId ()
java	String get_hardwareId ()
py	get_hardwareId ()
php	function get_hardwareId ()
ts	async get_hardwareId (): Promise<string>
es	async get_hardwareId ()
dnp	string get_hardwareId ()
cp	string get_hardwareId ()

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

Retourne :

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

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

display→**get_layerCount()****YDisplay****display**→**layerCount()**

Retourne le nombre des couches affichables disponibles.

js	function get_layerCount ()
cpp	int get_layerCount ()
m	-(int) layerCount
pas	LongInt get_layerCount (): LongInt
vb	function get_layerCount () As Integer
cs	int get_layerCount ()
java	int get_layerCount ()
uwp	async Task<int> get_layerCount ()
py	get_layerCount ()
php	function get_layerCount ()
ts	async get_layerCount (): Promise<number>
es	async get_layerCount ()
dnp	int get_layerCount ()
cp	int get_layerCount ()
cmd	YDisplay target get_layerCount

Retourne :

un entier représentant le nombre des couches affichables disponibles

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

display→**get_layerHeight()****YDisplay****display**→**layerHeight()**

Retourne la hauteur des couches affichables, en pixels.

js	function get_layerHeight ()
cpp	int get_layerHeight ()
m	-(int) layerHeight
pas	LongInt get_layerHeight (): LongInt
vb	function get_layerHeight () As Integer
cs	int get_layerHeight ()
java	int get_layerHeight ()
uwp	async Task<int> get_layerHeight ()
py	get_layerHeight ()
php	function get_layerHeight ()
ts	async get_layerHeight (): Promise<number>
es	async get_layerHeight ()
dnp	int get_layerHeight ()
cp	int get_layerHeight ()
cmd	YDisplay target get_layerHeight

Retourne :

un entier représentant la hauteur des couches affichables, en pixels

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

display→**get_layerWidth()****YDisplay****display**→**layerWidth()**

Retourne la largeur des couches affichables, en pixels.

js	function get_layerWidth ()
cpp	int get_layerWidth ()
m	-(int) layerWidth
pas	LongInt get_layerWidth (): LongInt
vb	function get_layerWidth () As Integer
cs	int get_layerWidth ()
java	int get_layerWidth ()
uwp	async Task<int> get_layerWidth ()
py	get_layerWidth ()
php	function get_layerWidth ()
ts	async get_layerWidth (): Promise<number>
es	async get_layerWidth ()
dnp	int get_layerWidth ()
cp	int get_layerWidth ()
cmd	YDisplay target get_layerWidth

Retourne :

un entier représentant la largeur des couches affichables, en pixels

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

display→**get_logicalName()****display**→**logicalName()**

Retourne le nom logique de l'écran.

js	function get_logicalName ()
cpp	string get_logicalName ()
m	-(NSString*) logicalName
pas	string get_logicalName (): string
vb	function get_logicalName () As String
cs	string get_logicalName ()
java	String get_logicalName ()
uwp	async Task<string> get_logicalName ()
py	get_logicalName ()
php	function get_logicalName ()
ts	async get_logicalName (): Promise<string>
es	async get_logicalName ()
dnp	string get_logicalName ()
cp	string get_logicalName ()
cmd	YDisplay target get_logicalName

Retourne :

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

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

display→**get_module()****YDisplay****display**→**module()**

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

js	function get_module ()
cpp	<code>YModule *</code> get_module ()
m	-(<code>YModule*</code>) module
pas	<code>TYModule</code> get_module (): <code>TYModule</code>
vb	function get_module () As <code>YModule</code>
cs	<code>YModule</code> get_module ()
java	<code>YModule</code> get_module ()
py	get_module ()
php	function get_module ()
ts	async get_module (): <code>Promise<YModule></code>
es	async get_module ()
dnp	<code>YModuleProxy</code> get_module ()
cp	<code>YModuleProxy *</code> get_module ()

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

Retourne :

une instance de `YModule`

display→**get_module_async()****YDisplay****display**→**module_async()**

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

```
js function get_module_async( callback, context)
```

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

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

Paramètres :

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

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

Retourne :

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

display→**get_orientation()****YDisplay****display**→**orientation()**

Retourne l'orientation sélectionnée pour l'écran.

js	function get_orientation ()
cpp	Y_ORIENTATION_enum get_orientation ()
m	-(Y_ORIENTATION_enum) orientation
pas	Integer get_orientation (): Integer
vb	function get_orientation () As Integer
cs	int get_orientation ()
java	int get_orientation ()
uwp	async Task<int> get_orientation ()
py	get_orientation ()
php	function get_orientation ()
ts	async get_orientation (): Promise<YDisplay_Orientation>
es	async get_orientation ()
dnp	int get_orientation ()
cp	int get_orientation ()
cmd	YDisplay target get_orientation

Retourne :

une valeur parmi YDisplay.ORIENTATION_LEFT, YDisplay.ORIENTATION_UP, YDisplay.ORIENTATION_RIGHT et YDisplay.ORIENTATION_DOWN représentant l'orientation sélectionnée pour l'écran

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

display→**get_serialNumber()****YDisplay****display**→**serialNumber()**

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

js	function get_serialNumber ()
cpp	string get_serialNumber ()
m	-(NSString*) serialNumber
pas	string get_serialNumber (): string
vb	function get_serialNumber () As String
cs	string get_serialNumber ()
java	String get_serialNumber ()
uwp	async Task<string> get_serialNumber ()
py	get_serialNumber ()
php	function get_serialNumber ()
ts	async get_serialNumber (): Promise<string>
es	async get_serialNumber ()
dnp	string get_serialNumber ()
cp	string get_serialNumber ()
cmd	YDisplay target get_serialNumber

Retourne :

: une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine.

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

display→**get_startupSeq()****YDisplay****display**→**startupSeq()**

Retourne le nom de la séquence à jouer à la mise sous tension de l'écran.

js	function get_startupSeq ()
cpp	string get_startupSeq ()
m	-(NSString*) startupSeq
pas	string get_startupSeq (): string
vb	function get_startupSeq () As String
cs	string get_startupSeq ()
java	String get_startupSeq ()
uwp	async Task<string> get_startupSeq ()
py	get_startupSeq ()
php	function get_startupSeq ()
ts	async get_startupSeq (): Promise<string>
es	async get_startupSeq ()
dnp	string get_startupSeq ()
cp	string get_startupSeq ()
cmd	YDisplay target get_startupSeq

Retourne :

une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

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

display→get_userData()**YDisplay****display→userData()**

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

js	function get_userdata ()
cpp	void * get_userdata ()
m	-(id) userData
pas	Tobject get_userdata (): Tobject
vb	function get_userdata () As Object
cs	object get_userdata ()
java	Object get_userdata ()
py	get_userdata ()
php	function get_userdata ()
ts	async get_userdata (): Promise<object null>
es	async get_userdata ()

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

Retourne :

l'objet stocké précédemment par l'appelant.

display→isOnline()**YDisplay**

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

js	function isOnline ()
cpp	bool isOnline ()
m	-(BOOL) isOnline
pas	boolean isOnline (): boolean
vb	function isOnline () As Boolean
cs	bool isOnline ()
java	boolean isOnline ()
py	isOnline ()
php	function isOnline ()
ts	async isOnline (): Promise<boolean>
es	async isOnline ()
dnp	bool isOnline ()
cp	bool isOnline ()

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

Retourne :

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

display→isOnline_async()**YDisplay**

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

```
js function isOnline_async( callback, context)
```

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

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

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

Retourne :

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

display→isReadOnly()**YDisplay**

Test si la fonction est en lecture seule.

cpp	bool isReadOnly ()
m	-(bool) isReadOnly
pas	boolean isReadOnly (): boolean
vb	function isReadOnly () As Boolean
cs	bool isReadOnly ()
java	boolean isReadOnly ()
uwp	async Task<bool> isReadOnly ()
py	isReadOnly ()
php	function isReadOnly ()
ts	async isReadOnly (): Promise<boolean>
es	async isReadOnly ()
dnp	bool isReadOnly ()
cp	bool isReadOnly ()
cmd	YDisplay target isReadOnly

Retourne vrais si la fonction est protégé en ecriture ou que la fontion n'est pas disponible.

Retourne :

`true` si la fonction est protégé en ecriture ou que la fontion n'est pas disponible

display→load()**YDisplay**

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

js	function load (msValidity)
c++	YRETCODE load (int msValidity)
m	-(YRETCODE) load : (u64) msValidity
pas	YRETCODE load (msValidity : u64): YRETCODE
vb	function load (ByVal msValidity As Long) As YRETCODE
cs	YRETCODE load (ulong msValidity)
java	int load (long msValidity)
py	load (msValidity)
php	function load (\$msValidity)
ts	async load (msValidity : number): Promise<number>
es	async load (msValidity)

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

Paramètres :

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

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→loadAttribute()**YDisplay**

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

js	function loadAttribute(attrName)
cpp	string loadAttribute(string attrName)
m	-(NSString*) loadAttribute : (NSString*) attrName
pas	string loadAttribute(attrName: string): string
vb	function loadAttribute(ByVal attrName As String) As String
cs	string loadAttribute(string attrName)
java	String loadAttribute(String attrName)
uwp	async Task<string> loadAttribute(string attrName)
py	loadAttribute(attrName)
php	function loadAttribute(\$attrName)
ts	async loadAttribute(attrName: string): Promise<string>
es	async loadAttribute(attrName)
dnf	string loadAttribute(string attrName)
cp	string loadAttribute(string attrName)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

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

display→load_async()**YDisplay**

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

```
js function load_async( msValidity, callback, context)
```

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI . SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

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

display→muteValueCallbacks()**YDisplay**

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

js	function muteValueCallbacks ()
cpp	int muteValueCallbacks ()
m	-(int) muteValueCallbacks
pas	LongInt muteValueCallbacks (): LongInt
vb	function muteValueCallbacks () As Integer
cs	int muteValueCallbacks ()
java	int muteValueCallbacks ()
uwp	async Task<int> muteValueCallbacks ()
py	muteValueCallbacks ()
php	function muteValueCallbacks ()
ts	async muteValueCallbacks (): Promise<number>
es	async muteValueCallbacks ()
dnp	int muteValueCallbacks ()
cp	int muteValueCallbacks ()
cmd	YDisplay target muteValueCallbacks

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclenchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→newSequence()**YDisplay**

Enclanche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

js	function newSequence ()
cpp	int newSequence ()
m	-(int) newSequence
pas	LongInt newSequence (): LongInt
vb	function newSequence () As Integer
cs	int newSequence ()
java	int newSequence ()
uwp	async Task<int> newSequence ()
py	newSequence ()
php	function newSequence ()
ts	async newSequence (): Promise<number>
es	async newSequence ()
dnp	int newSequence ()
cp	int newSequence ()
cmd	YDisplay target newSequence

Le nom de la séquence sera donné au moment de l'appel à `saveSequence()`, une fois la séquence terminée.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→**nextDisplay()****YDisplay**

Continue l'énumération des écrans commencée à l'aide de `yFirstDisplay()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les écrans sont retournés.

js	function nextDisplay ()
cpp	YDisplay * nextDisplay ()
m	-(nullable YDisplay*) nextDisplay
pas	TYDisplay nextDisplay (): TYDisplay
vb	function nextDisplay () As YDisplay
cs	YDisplay nextDisplay ()
java	YDisplay nextDisplay ()
uwp	YDisplay nextDisplay ()
py	nextDisplay ()
php	function nextDisplay ()
ts	nextDisplay (): YDisplay null
es	nextDisplay ()

Si vous souhaitez retrouver un écran spécifique, utilisez `Display.findDisplay()` avec un `hardwareID` ou un nom logique.

Retourne :

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

display→pauseSequence()**YDisplay**

Attend pour la durée spécifiée (en millisecondes) avant de jouer les commandes suivantes de la séquence active.

js	function pauseSequence (delay_ms)
cpp	int pauseSequence (int delay_ms)
m	-(int) pauseSequence : (int) delay_ms
pas	LongInt pauseSequence (delay_ms : LongInt): LongInt
vb	function pauseSequence (ByVal delay_ms As Integer) As Integer
cs	int pauseSequence (int delay_ms)
java	int pauseSequence (int delay_ms)
uwp	async Task<int> pauseSequence (int delay_ms)
py	pauseSequence (delay_ms)
php	function pauseSequence (\$delay_ms)
ts	async pauseSequence (delay_ms : number): Promise<number>
es	async pauseSequence (delay_ms)
dnp	int pauseSequence (int delay_ms)
cp	int pauseSequence (int delay_ms)
cmd	YDisplay target pauseSequence delay_ms

Cette méthode peut être utilisée lors de l'enregistrement d'une séquence d'affichage, pour insérer une attente mesurée lors de l'exécution (mais sans effet immédiat). Cette méthode peut aussi être appelée dynamiquement pendant l'exécution d'une séquence enregistrée, pour suspendre temporairement ou reprendre l'exécution. Pour annuler une attente, appelez simplement la méthode avec une attente de zéro.

Paramètres :

delay_ms la durée de l'attente, en millisecondes

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→playSequence()**YDisplay**

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes `newSequence()` et `saveSequence()`.

js	<code>function playSequence(sequenceName)</code>
cpp	<code>int playSequence(string sequenceName)</code>
m	<code>-(int) playSequence : (NSString*) sequenceName</code>
pas	<code>LongInt playSequence(sequenceName: string): LongInt</code>
vb	<code>function playSequence(ByVal sequenceName As String) As Integer</code>
cs	<code>int playSequence(string sequenceName)</code>
java	<code>int playSequence(String sequenceName)</code>
uwp	<code>async Task<int> playSequence(string sequenceName)</code>
py	<code>playSequence(sequenceName)</code>
php	<code>function playSequence(\$sequenceName)</code>
ts	<code>async playSequence(sequenceName: string): Promise<number></code>
es	<code>async playSequence(sequenceName)</code>
dnp	<code>int playSequence(string sequenceName)</code>
cp	<code>int playSequence(string sequenceName)</code>
cmd	<code>YDisplay target playSequence sequenceName</code>

Paramètres :

sequenceName le nom de la nouvelle séquence créée

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→registerValueCallback()**YDisplay**

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

js	function registerValueCallback (callback)
cpp	int registerValueCallback (YDisplayValueCallback callback)
m	-(int) registerValueCallback : (YDisplayValueCallback _Nullable) callback
pas	LongInt registerValueCallback (callback : TYDisplayValueCallback): LongInt
vb	function registerValueCallback (ByVal callback As YDisplayValueCallback) As Integer
cs	int registerValueCallback (ValueCallback callback)
java	int registerValueCallback (UpdateCallback callback)
uwp	async Task<int> registerValueCallback (ValueCallback callback)
py	registerValueCallback (callback)
php	function registerValueCallback (\$callback)
ts	async registerValueCallback (callback : YDisplayValueCallback null): Promise<number>
es	async registerValueCallback (callback)

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

Paramètres :

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

display→resetAll()**YDisplay**

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

js	function resetAll ()
cpp	int resetAll ()
m	-(int) resetAll
pas	LongInt resetAll (): LongInt
vb	function resetAll () As Integer
cs	int resetAll ()
java	int resetAll ()
uwp	async Task<int> resetAll ()
py	resetAll ()
php	function resetAll ()
ts	async resetAll (): Promise<number>
es	async resetAll ()
dnp	int resetAll ()
cp	int resetAll ()
cmd	YDisplay target resetAll

Utiliser cette fonction dans une sequence va tuer stopper l'affichage de la sequence: ne pas utiliser cette fonction pour réinitialiser l'écran au début d'une séquence.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→saveSequence()**YDisplay**

Termine l'enregistrement d'une séquence et la sauvegarde sur la mémoire interne de l'écran, sous le nom choisi.

js	function saveSequence (sequenceName)
cpp	int saveSequence (string sequenceName)
m	-(int) saveSequence : (NSString*) sequenceName
pas	LongInt saveSequence (sequenceName : string): LongInt
vb	function saveSequence (ByVal sequenceName As String) As Integer
cs	int saveSequence (string sequenceName)
java	int saveSequence (String sequenceName)
uwp	async Task<int> saveSequence (string sequenceName)
py	saveSequence (sequenceName)
php	function saveSequence (\$sequenceName)
ts	async saveSequence (sequenceName : string): Promise<number>
es	async saveSequence (sequenceName)
dnp	int saveSequence (string sequenceName)
cp	int saveSequence (string sequenceName)
cmd	YDisplay target saveSequence sequenceName

La séquence peut être rejouée ultérieurement à l'aide de la méthode `playSequence()`.

Paramètres :

sequenceName le nom de la nouvelle séquence créée

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→**set_brightness()****YDisplay****display**→**setBrightness()**

Modifie la luminosité de l'écran.

js	function set_brightness (newval)
cpp	int set_brightness (int newval)
m	-(int) setBrightness : (int) newval
pas	integer set_brightness (newval : LongInt): integer
vb	function set_brightness (ByVal newval As Integer) As Integer
cs	int set_brightness (int newval)
java	int set_brightness (int newval)
uwp	async Task<int> set_brightness (int newval)
py	set_brightness (newval)
php	function set_brightness (\$ newval)
ts	async set_brightness (newval : number): Promise<number>
es	async set_brightness (newval)
dnp	int set_brightness (int newval)
cp	int set_brightness (int newval)
cmd	YDisplay target set_brightness newval

Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la luminosité de l'écran

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→**set_enabled()****YDisplay****display**→**setEnabled()**

Modifie l'état d'activité de l'écran.

js	function set_enabled (newval)
cpp	int set_enabled (Y_ENABLED_enum newval)
m	-(int) setEnabled : (Y_ENABLED_enum) newval
pas	integer set_enabled (newval : Integer): integer
vb	function set_enabled (ByVal newval As Integer) As Integer
cs	int set_enabled (int newval)
java	int set_enabled (int newval)
uwp	async Task<int> set_enabled (int newval)
py	set_enabled (newval)
php	function set_enabled (\$ newval)
ts	async set_enabled (newval : YDisplay_Enabled): Promise<number>
es	async set_enabled (newval)
dnp	int set_enabled (int newval)
cp	int set_enabled (int newval)
cmd	YDisplay target set_enabled newval

Paramètres :

newval soit YDisplay.ENABLED_FALSE, soit YDisplay.ENABLED_TRUE, selon l'état d'activité de l'écran

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→**set_logicalName()****YDisplay****display**→**setLogicalName()**

Modifie le nom logique de l'écran.

js	function set_logicalName (newval)
cpp	int set_logicalName (string newval)
m	-(int) setLogicalName : (NSString*) newval
pas	integer set_logicalName (newval : string): integer
vb	function set_logicalName (ByVal newval As String) As Integer
cs	int set_logicalName (string newval)
java	int set_logicalName (String newval)
uwp	async Task<int> set_logicalName (string newval)
py	set_logicalName (newval)
php	function set_logicalName (\$ newval)
ts	async set_logicalName (newval : string): Promise<number>
es	async set_logicalName (newval)
dnp	int set_logicalName (string newval)
cp	int set_logicalName (string newval)
cmd	YDisplay target set_logicalName newval

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

Paramètres :

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

Retourne :

YAPI . SUCCESS si l'appel se déroule sans erreur.

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

display→**set_orientation()****YDisplay****display**→**setOrientation()**

Modifie l'orientation de l'écran.

js	function set_orientation (newval)
cpp	int set_orientation (Y_ORIENTATION_enum newval)
m	-(int) setOrientation : (Y_ORIENTATION_enum) newval
pas	integer set_orientation (newval : Integer): integer
vb	function set_orientation (ByVal newval As Integer) As Integer
cs	int set_orientation (int newval)
java	int set_orientation (int newval)
uwp	async Task<int> set_orientation (int newval)
py	set_orientation (newval)
php	function set_orientation (\$ newval)
ts	async set_orientation (newval : YDisplay_Orientation): Promise<number>
es	async set_orientation (newval)
dnp	int set_orientation (int newval)
cp	int set_orientation (int newval)
cmd	YDisplay target set_orientation newval

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

Paramètres :

newval une valeur parmi `YDisplay.ORIENTATION_LEFT`, `YDisplay.ORIENTATION_UP`, `YDisplay.ORIENTATION_RIGHT` et `YDisplay.ORIENTATION_DOWN` représentant l'orientation de l'écran

Retourne :

`YAPI . SUCCESS` si l'opération se déroule sans erreur.

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

display→**set_startupSeq()****YDisplay****display**→**setStartupSeq()**

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

js	function set_startupSeq (newval)
cpp	int set_startupSeq (string newval)
m	-(int) setStartupSeq : (NSString*) newval
pas	integer set_startupSeq (newval : string): integer
vb	function set_startupSeq (ByVal newval As String) As Integer
cs	int set_startupSeq (string newval)
java	int set_startupSeq (String newval)
uwp	async Task<int> set_startupSeq (string newval)
py	set_startupSeq (newval)
php	function set_startupSeq (\$ newval)
ts	async set_startupSeq (newval : string): Promise<number>
es	async set_startupSeq (newval)
dnp	int set_startupSeq (string newval)
cp	int set_startupSeq (string newval)
cmd	YDisplay target set_startupSeq newval

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

Paramètres :

newval une chaîne de caractères représentant le nom de la séquence à jouer à la mise sous tension de l'écran

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→**set_userData()****YDisplay****display**→**setUserData()**

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

js	function set_userData (data)
cpp	void set_userData (void * data)
m	-(void) setUserData : (id) data
pas	set_userData (data : Tobject)
vb	procedure set_userData (ByVal data As Object)
cs	void set_userData (object data)
java	void set_userData (Object data)
py	set_userData (data)
php	function set_userData (\$data)
ts	async set_userData (data : object null): Promise<void>
es	async set_userData (data)

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

Paramètres :

data objet quelconque à mémoriser

display→stopSequence()**YDisplay**

Arrête immédiatement la séquence d'affichage actuellement jouée sur l'écran.

js	function stopSequence ()
cpp	int stopSequence ()
m	-(int) stopSequence
pas	LongInt stopSequence (): LongInt
vb	function stopSequence () As Integer
cs	int stopSequence ()
java	int stopSequence ()
uwp	async Task<int> stopSequence ()
py	stopSequence ()
php	function stopSequence ()
ts	async stopSequence (): Promise<number>
es	async stopSequence ()
dnp	int stopSequence ()
cp	int stopSequence ()
cmd	YDisplay target stopSequence

L'affichage est laissé tel quel.

Paramètres :

sequenceName le nom de la nouvelle séquence créée

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

Permute le contenu de deux couches d'affichage.

js	function swapLayerContent (layerIdA , layerIdB)
cpp	int swapLayerContent (int layerIdA , int layerIdB)
m	-(int) swapLayerContent : (int) layerIdA : (int) layerIdB
pas	LongInt swapLayerContent (layerIdA : LongInt, layerIdB : LongInt): LongInt
vb	function swapLayerContent (ByVal layerIdA As Integer, ByVal layerIdB As Integer) As Integer
cs	int swapLayerContent (int layerIdA , int layerIdB)
java	int swapLayerContent (int layerIdA , int layerIdB)
uwp	async Task<int> swapLayerContent (int layerIdA , int layerIdB)
py	swapLayerContent (layerIdA , layerIdB)
php	function swapLayerContent (\$layerIdA , \$layerIdB)
ts	async swapLayerContent (layerIdA : number, layerIdB : number): Promise<number>
es	async swapLayerContent (layerIdA , layerIdB)
dnp	int swapLayerContent (int layerIdA , int layerIdB)
cp	int swapLayerContent (int layerIdA , int layerIdB)
cmd	YDisplay target swapLayerContent layerIdA layerIdB

La couleur et la transparence de tous les pixels des deux couches sont permutées. Cette méthode modifie le contenu affiché, mais n'a aucun effet sur les propriétés de l'objet layer lui-même. En particulier, la visibilité des deux couches reste inchangée. Cela permet d'implémenter très efficacement un affichage par double-buffering, en utilisant une couche cachée et une couche visible. Notez que la couche zéro n'a pas de transparence (elle est toujours opaque).

Paramètres :

layerIdA l'identifiant de la première couche (un chiffre parmi 0..layerCount-1)

layerIdB l'identifiant de la deuxième couche (un chiffre parmi 0..layerCount-1)

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→unmuteValueCallbacks()**YDisplay**

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

js	function unmuteValueCallbacks ()
cpp	int unmuteValueCallbacks ()
m	-(int) unmuteValueCallbacks
pas	LongInt unmuteValueCallbacks (): LongInt
vb	function unmuteValueCallbacks () As Integer
cs	int unmuteValueCallbacks ()
java	int unmuteValueCallbacks ()
uwp	async Task<int> unmuteValueCallbacks ()
py	unmuteValueCallbacks ()
php	function unmuteValueCallbacks ()
ts	async unmuteValueCallbacks (): Promise<number>
es	async unmuteValueCallbacks ()
dnp	int unmuteValueCallbacks ()
cp	int unmuteValueCallbacks ()
cmd	YDisplay target unmuteValueCallbacks

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→upload()**YDisplay**

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

js	function upload (pathname , content)
cpp	int upload (string pathname , string content)
m	-(int) upload : (NSString*) pathname : (NSData*) content
pas	LongInt upload (pathname : string, content : TByteArray): LongInt
vb	procedure upload (ByVal pathname As String, ByVal content As Byte())
cs	int upload (string pathname , byte[] content)
java	int upload (String pathname , byte[] content)
uwp	async Task<int> upload (string pathname , byte[] content)
py	upload (pathname , content)
php	function upload (\$ pathname , \$ content)
ts	async upload (pathname : string, content : Uint8Array): Promise<number>
es	async upload (pathname , content)
dnp	int upload (string pathname , byte[] content)
cp	int upload (string pathname , string content)
cmd	YDisplay target upload pathname content

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

Paramètres :

pathname nom complet du fichier, y compris le chemin d'accès.

content contenu du fichier à télécharger

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

display→wait_async()**YDisplay**

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

js	<code>function wait_async(callback, context)</code>
ts	<code>wait_async(callback: Function, context: object)</code>
es	<code>wait_async(callback, context)</code>

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées. La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

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

Retourne :

rien du tout.

24.4. La classe YDisplayLayer

Interface pour dessiner dans les couches graphiques des écrans, obtenue par l'appel à `display.get_displayLayer`.

Chaque `DisplayLayer` sert à contrôler une couche de contenu affichable (images, texte, etc.). Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code><script type='text/javascript' src='yocto_display.js'></script></code>
cpp	<code>#include "yocto_display.h"</code>
m	<code>#import "yocto_display.h"</code>
pas	<code>uses yocto_display;</code>
vb	<code>yocto_display.vb</code>
cs	<code>yocto_display.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YDisplayLayer;</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YDisplayLayer;</code>
py	<code>from yocto_display import *</code>
php	<code>require_once('yocto_display.php');</code>
ts	in HTML: <code>import { YDisplayLayer } from '../dist/esm/yocto_display.js';</code> in Node.js: <code>import { YDisplayLayer } from 'yoctolib-cjs/yocto_display.js';</code>
es	in HTML: <code><script src='../lib/yocto_display.js'></script></code> in node.js: <code>require('yoctolib-es2017/yocto_display.js');</code>
dnf	<code>import YoctoProxyAPI.YDisplayLayerProxy</code>
cp	<code>#include "yocto_display_proxy.h"</code>
ml	<code>import YoctoProxyAPI.YDisplayLayerProxy</code>

Méthodes des objets YDisplayLayer

`displaylayer→clear()`

Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.

`displaylayer→clearConsole()`

Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.

`displaylayer→consoleOut(text)`

Affiche un message dans la zone de console, et déplace le curseur de la console à la fin du texte.

`displaylayer→drawBar(x1, y1, x2, y2)`

Dessine un rectangle plein à une position spécifiée.

`displaylayer→drawBitmap(x, y, w, bitmap, bgcolor)`

Dessine un bitmap à la position spécifiée de la couche.

`displaylayer→drawCircle(x, y, r)`

Dessine un cercle vide à une position spécifiée.

`displaylayer→drawDisc(x, y, r)`

Dessine un disque plein à une position spécifiée.

`displaylayer→drawImage(x, y, imagename)`

Dessine une image GIF à la position spécifiée de la couche.

`displaylayer→drawPixel(x, y)`

Dessine un pixel unique à une position spécifiée.

displaylayer→drawRect(x1, y1, x2, y2)

Dessine un rectangle vide à une position spécifiée.

displaylayer→drawText(x, y, anchor, text)

Affiche un texte à la position spécifiée de la couche.

displaylayer→get_display()

Retourne l'YDisplay parent.

displaylayer→get_displayHeight()

Retourne la hauteur de l'écran, en pixels.

displaylayer→get_displayWidth()

Retourne la largeur de l'écran, en pixels.

displaylayer→get_layerHeight()

Retourne la hauteur des couches affichables, en pixels.

displaylayer→get_layerWidth()

Retourne la largeur des couches affichables, en pixels.

displaylayer→hide()

Cache la couche de dessin.

displaylayer→lineTo(x, y)

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

displaylayer→moveTo(x, y)

Déplace le point de dessin courant de cette couche à la position spécifiée.

displaylayer→reset()

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

displaylayer→selectColorPen(color)

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

displaylayer→selectEraser()

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de copie d'images bitmaps.

displaylayer→selectFont(fontname)

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

displaylayer→selectGrayPen(graylevel)

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

displaylayer→setAntialiasingMode(mode)

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

displaylayer→setConsoleBackground(bgcol)

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

displaylayer→setConsoleMargins(x1, y1, x2, y2)

Configure les marges d'affichage pour la fonction `consoleOut`.

displaylayer→setConsoleWordWrap(wordwrap)

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

displaylayer→setLayerPosition(x, y, scrollTime)

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

displaylayer→unhide()

Affiche la couche.

displaylayer→clear()**YDisplayLayer**

Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.

js	function clear ()
cpp	int clear ()
m	-(int) clear
pas	LongInt clear (): LongInt
vb	function clear () As Integer
cs	int clear ()
java	int clear ()
uwp	async Task<int> clear ()
py	clear ()
php	function clear ()
ts	async clear (): Promise<number>
es	async clear ()
dnp	int clear ()
cp	int clear ()
cmd	YDisplay target [-layer layerId] clear

Cette méthode ne change pas les réglages de la couche. Si vous désirez remettre la couche dans son état initial, utilisez plutôt la méthode `reset()`.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→clearConsole()**YDisplayLayer**

Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.

js	function clearConsole ()
cpp	int clearConsole ()
m	-(int) clearConsole
pas	LongInt clearConsole (): LongInt
vb	function clearConsole () As Integer
cs	int clearConsole ()
java	int clearConsole ()
uwp	async Task<int> clearConsole ()
py	clearConsole ()
php	function clearConsole ()
ts	async clearConsole (): Promise<number>
es	async clearConsole ()
dnp	int clearConsole ()
cp	int clearConsole ()
cmd	YDisplay target [-layer layerId] clearConsole

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→consoleOut()**YDisplayLayer**

Affiche un message dans la zone de console, et déplace le curseur de la console à la fin du texte.

js	function consoleOut (text)
c++	int consoleOut (string text)
m	-(int) consoleOut : (NSString*) text
pas	LongInt consoleOut (text : string): LongInt
vb	function consoleOut (ByVal text As String) As Integer
cs	int consoleOut (string text)
java	int consoleOut (String text)
uwp	async Task<int> consoleOut (string text)
py	consoleOut (text)
php	function consoleOut (\$ text)
ts	async consoleOut (text : string): Promise<number>
es	async consoleOut (text)
dnp	int consoleOut (string text)
cp	int consoleOut (string text)
cmd	YDisplay target [-layer layerId] consoleOut text

Le curseur revient automatiquement en début de ligne suivante lorsqu'un saut de ligne est rencontré, ou lorsque la marge droite est atteinte. Lorsque le texte à afficher s'apprête à dépasser la marge inférieure, le contenu de la zone de console est automatiquement décalé vers le haut afin de laisser la place à la nouvelle ligne de texte.

Paramètres :

text le message à afficher

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→drawBar()**YDisplayLayer**

Dessine un rectangle plein à une position spécifiée.

js	function drawBar (x1 , y1 , x2 , y2)
cpp	int drawBar (int x1 , int y1 , int x2 , int y2)
m	-(int) drawBar : (int) x1 : (int) y1 : (int) x2 : (int) y2
pas	LongInt drawBar (x1 : LongInt, y1 : LongInt, x2 : LongInt, y2 : LongInt): LongInt
vb	function drawBar (ByVal x1 As Integer, ByVal y1 As Integer, ByVal x2 As Integer, ByVal y2 As Integer) As Integer
cs	int drawBar (int x1 , int y1 , int x2 , int y2)
java	int drawBar (int x1 , int y1 , int x2 , int y2)
uwp	async Task<int> drawBar (int x1 , int y1 , int x2 , int y2)
py	drawBar (x1 , y1 , x2 , y2)
php	function drawBar (\$x1 , \$y1 , \$x2 , \$y2)
ts	async drawBar (x1 : number, y1 : number, x2 : number, y2 : number): Promise<number>
es	async drawBar (x1 , y1 , x2 , y2)
dnp	int drawBar (int x1 , int y1 , int x2 , int y2)
cp	int drawBar (int x1 , int y1 , int x2 , int y2)
cmd	YDisplay target [-layer layerId] drawBar x1 y1 x2 y2

Paramètres :

- x1** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle
- y1** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle
- x2** la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle
- y2** la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→drawBitmap()**YDisplayLayer**

Dessine un bitmap à la position spécifiée de la couche.

```

js function drawBitmap( x, y, w, bitmap, bgcol)
cpp int drawBitmap( int x, int y, int w, string bitmap, int bgcol)
m -(int) drawBitmap : (int) x
                        : (int) y
                        : (int) w
                        : (NSData*) bitmap
                        : (int) bgcol
pas LongInt drawBitmap( x: LongInt,
                        y: LongInt,
                        w: LongInt,
                        bitmap: TByteArray,
                        bgcol: LongInt): LongInt
vb procedure drawBitmap( ByVal x As Integer,
                        ByVal y As Integer,
                        ByVal w As Integer,
                        ByVal bitmap As Byte())
cs int drawBitmap( int x, int y, int w, byte[] bitmap, int bgcol)
java int drawBitmap( int x, int y, int w, byte[] bitmap, int bgcol)
uwp async Task<int> drawBitmap( int x,
                                int y,
                                int w,
                                byte[] bitmap,
                                int bgcol)
py drawBitmap( x, y, w, bitmap, bgcol)
php function drawBitmap( $x, $y, $w, $bitmap, $bgcol)
ts async drawBitmap( x: number, y: number, w: number, bitmap: Uint8Array, bgcol: number):
    Promise<number>
es async drawBitmap( x, y, w, bitmap, bgcol)
dnp int drawBitmap( int x, int y, int w, byte[] bitmap, int bgcol)
cp int drawBitmap( int x, int y, int w, string bitmap, int bgcol)
cmd YDisplay target [-layer layerId] drawBitmap x y w bitmap bgcol

```

Le bitmap est passé sous forme d'un objet binaire, où chaque bit correspond à un pixel, de gauche à droite et de haut en bas. Le bit de poids fort de chaque octet correspond au pixel de gauche, et le bit de poids faible au pixel le plus à droite. Les bits à 1 sont dessinés avec la couleur active de la couche. Les bits à 0 avec la couleur de fond spécifiée, sauf si la valeur -1 a été choisie, auquel cas ils ne sont pas dessinés (ils sont considérés comme transparents). Chaque ligne commence sur un nouvel octet. La hauteur du bitmap est donnée implicitement par la taille de l'objet binaire.

Paramètres :

- x** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du bitmap
- y** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du bitmap
- w** la largeur du bitmap, en pixels
- bitmap** l'objet binaire contenant le bitmap
- bgcol** le niveau de gris à utiliser pour les bits à zéro (0 = noir, 255 = blanc), ou -1 pour laisser les pixels inchangés

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

Dessine un cercle vide à une position spécifiée.

js	function drawCircle (x , y , r)
cpp	int drawCircle (int x , int y , int r)
m	-(int) drawCircle : (int) x : (int) y : (int) r
pas	LongInt drawCircle (x : LongInt, y : LongInt, r : LongInt): LongInt
vb	function drawCircle (ByVal x As Integer, ByVal y As Integer, ByVal r As Integer) As Integer
cs	int drawCircle (int x , int y , int r)
java	int drawCircle (int x , int y , int r)
uwp	async Task<int> drawCircle (int x , int y , int r)
py	drawCircle (x , y , r)
php	function drawCircle (\$x , \$y , \$r)
ts	async drawCircle (x : number, y : number, r : number): Promise<number>
es	async drawCircle (x , y , r)
dnp	int drawCircle (int x , int y , int r)
cp	int drawCircle (int x , int y , int r)
cmd	YDisplay target [-layer layerId] drawCircle x y r

Paramètres :

- x** la distance en pixels depuis la gauche de la couche jusqu'au centre du cercle
- y** la distance en pixels depuis le haut de la couche jusqu'au centre du cercle
- r** le rayon du cercle, en pixels

Retourne :

YAPI .SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→drawImage()

YDisplayLayer

Dessine une image GIF à la position spécifiée de la couche.

js	function drawImage (x , y , imageName)
cpp	int drawImage (int x , int y , string imageName)
m	-(int) drawImage : (int) x : (int) y : (NSString*) imageName
pas	LongInt drawImage (x : LongInt, y : LongInt, imageName : string): LongInt
vb	function drawImage (ByVal x As Integer, ByVal y As Integer, ByVal imageName As String) As Integer
cs	int drawImage (int x , int y , string imageName)
java	int drawImage (int x , int y , String imageName)
uwp	async Task<int> drawImage (int x , int y , string imageName)
py	drawImage (x , y , imageName)
php	function drawImage (\$x , \$y , \$imageName)
ts	async drawImage (x : number, y : number, imageName : string): Promise<number>
es	async drawImage (x , y , imageName)
dnp	int drawImage (int x , int y , string imageName)
cp	int drawImage (int x , int y , string imageName)
cmd	YDisplay target [-layer layerId] drawImage x y imageName

L'image GIF doit avoir été préalablement préchargée dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une image bitmap, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier d'image manquant ou d'un format de fichier invalide.

Paramètres :

x	la distance en pixels depuis la gauche de la couche jusqu'au bord gauche de l'image
y	la distance en pixels depuis le haut de la couche jusqu'au bord supérieur de l'image
imagename	le nom du fichier GIF à afficher

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→drawRect()**YDisplayLayer**

Dessine un rectangle vide à une position spécifiée.

js	<code>function drawRect(x1, y1, x2, y2)</code>
cpp	<code>int drawRect(int x1, int y1, int x2, int y2)</code>
m	<code>-(int) drawRect : (int) x1 : (int) y1 : (int) x2 : (int) y2</code>
pas	<code>LongInt drawRect(x1: LongInt, y1: LongInt, x2: LongInt, y2: LongInt): LongInt</code>
vb	<code>function drawRect(ByVal x1 As Integer, ByVal y1 As Integer, ByVal x2 As Integer, ByVal y2 As Integer) As Integer</code>
cs	<code>int drawRect(int x1, int y1, int x2, int y2)</code>
java	<code>int drawRect(int x1, int y1, int x2, int y2)</code>
uwp	<code>async Task<int> drawRect(int x1, int y1, int x2, int y2)</code>
py	<code>drawRect(x1, y1, x2, y2)</code>
php	<code>function drawRect(\$x1, \$y1, \$x2, \$y2)</code>
ts	<code>async drawRect(x1: number, y1: number, x2: number, y2: number): Promise<number></code>
es	<code>async drawRect(x1, y1, x2, y2)</code>
dnp	<code>int drawRect(int x1, int y1, int x2, int y2)</code>
cp	<code>int drawRect(int x1, int y1, int x2, int y2)</code>
cmd	<code>YDisplay target [-layer layerId] drawRect x1 y1 x2 y2</code>

Paramètres :

- x1** la distance en pixels depuis la gauche de la couche jusqu'au bord gauche du rectangle
- y1** la distance en pixels depuis le haut de la couche jusqu'au bord supérieur du rectangle
- x2** la distance en pixels depuis la gauche de la couche jusqu'au bord droit du rectangle
- y2** la distance en pixels depuis le haut de la couche jusqu'au bord inférieur du rectangle

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→drawText()

YDisplayLayer

Affiche un texte à la position spécifiée de la couche.

js	function drawText (x , y , anchor , text)
cpp	int drawText (int x , int y , Y_ALIGN anchor , string text)
m	-(int) drawText : (int) x : (int) y : (Y_ALIGN) anchor : (NSString*) text
pas	LongInt drawText (x : LongInt, y : LongInt, anchor : TYALIGN, text : string): LongInt
vb	function drawText (ByVal x As Integer, ByVal y As Integer, ByVal anchor As Y_ALIGN, ByVal text As String) As Integer
cs	int drawText (int x , int y , ALIGN anchor , string text)
java	int drawText (int x , int y , ALIGN anchor , String text)
uwp	async Task<int> drawText (int x , int y , ALIGN anchor , string text)
py	drawText (x , y , anchor , text)
php	function drawText (\$x , \$y , \$anchor , \$text)
ts	async drawText (x : number, y : number, anchor : YDisplayLayer_Align, text : string): Promise<number>
es	async drawText (x , y , anchor , text)
dnp	int drawText (int x , int y , int anchor , string text)
cp	int drawText (int x , int y , int anchor , string text)
cmd	YDisplay target [-layer layerId] drawText x y anchor text

Le point du texte qui sera aligné sur la position spécifiée est appelé point d'ancrage, et peut être choisi parmi plusieurs options.

Paramètres :

```
x      la distance en pixels depuis la gauche de la couche jusqu'au point d'ancrage du texte
y      la distance en pixels depuis le haut de la couche jusqu'au point d'ancrage du texte
anchor le point d'ancrage du texte, choisi parmi l'énumération YDisplayLayer.ALIGN:
YDisplayLayer.ALIGN_TOP_LEFT, YDisplayLayer.ALIGN_CENTER_LEFT,
YDisplayLayer.ALIGN_BASELINE_LEFT,
YDisplayLayer.ALIGN_BOTTOM_LEFT,
YDisplayLayer.ALIGN_TOP_CENTER,  YDisplayLayer.ALIGN_CENTER,
YDisplayLayer.ALIGN_BASELINE_CENTER,
YDisplayLayer.ALIGN_BOTTOM_CENTER,
YDisplayLayer.ALIGN_TOP_DECIMAL,
YDisplayLayer.ALIGN_CENTER_DECIMAL,
YDisplayLayer.ALIGN_BASELINE_DECIMAL,
YDisplayLayer.ALIGN_BOTTOM_DECIMAL,
YDisplayLayer.ALIGN_TOP_RIGHT,
YDisplayLayer.ALIGN_CENTER_RIGHT,
YDisplayLayer.ALIGN_BASELINE_RIGHT,
YDisplayLayer.ALIGN_BOTTOM_RIGHT.
text le texte à afficher
```

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→**get_display()****YDisplayLayer****displaylayer**→**display()**

Retourne l'YDisplay parent.

js	function get_display ()
cpp	YDisplay* get_display ()
m	-(YDisplay*) display
pas	TYDisplay get_display (): TYDisplay
vb	function get_display () As YDisplay
cs	YDisplay get_display ()
java	YDisplay get_display ()
uwp	async Task<YDisplay> get_display ()
py	get_display ()
php	function get_display ()
ts	async get_display (): Promise<YDisplay>
es	async get_display ()
dnp	YDisplayProxy get_display ()
cp	YDisplayProxy* get_display ()

Retourne l'objet YDisplay parent du YDisplayLayer courant.

Retourne :

un objet YDisplay

displaylayer→**get_displayHeight()****YDisplayLayer****displaylayer**→**displayHeight()**

Retourne la hauteur de l'écran, en pixels.

js	function get_displayHeight ()
c++	int get_displayHeight ()
m	-(int) displayHeight
pas	LongInt get_displayHeight (): LongInt
vb	function get_displayHeight () As Integer
cs	int get_displayHeight ()
java	int get_displayHeight ()
uwp	async Task<int> get_displayHeight ()
py	get_displayHeight ()
php	function get_displayHeight ()
ts	async get_displayHeight (): Promise<number>
es	async get_displayHeight ()
dnp	int get_displayHeight ()
cp	int get_displayHeight ()
cmd	YDisplay target [-layer layerId] get_displayHeight

Retourne :

un entier représentant la hauteur de l'écran, en pixels

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

displaylayer→**get_displayWidth()****YDisplayLayer****displaylayer**→**displayWidth()**

Retourne la largeur de l'écran, en pixels.

js	function get_displayWidth ()
cpp	int get_displayWidth ()
m	-(int) displayWidth
pas	LongInt get_displayWidth (): LongInt
vb	function get_displayWidth () As Integer
cs	int get_displayWidth ()
java	int get_displayWidth ()
uwp	async Task<int> get_displayWidth ()
py	get_displayWidth ()
php	function get_displayWidth ()
ts	async get_displayWidth (): Promise<number>
es	async get_displayWidth ()
dnp	int get_displayWidth ()
cp	int get_displayWidth ()
cmd	YDisplay target [-layer layerId] get_displayWidth

Retourne :

un entier représentant la largeur de l'écran, en pixels

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

displaylayer→get_layerHeight()**YDisplayLayer****displaylayer→layerHeight()**

Retourne la hauteur des couches affichables, en pixels.

js	function get_layerHeight ()
c++	int get_layerHeight ()
m	-(int) layerHeight
pas	LongInt get_layerHeight (): LongInt
vb	function get_layerHeight () As Integer
cs	int get_layerHeight ()
java	int get_layerHeight ()
uwp	async Task<int> get_layerHeight ()
py	get_layerHeight ()
php	function get_layerHeight ()
ts	async get_layerHeight (): Promise<number>
es	async get_layerHeight ()
dnp	int get_layerHeight ()
cp	int get_layerHeight ()
cmd	YDisplay target [-layer layerId] get_layerHeight

Retourne :

un entier représentant la hauteur des couches affichables, en pixels.

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

displaylayer→**get_layerWidth()****YDisplayLayer****displaylayer**→**layerWidth()**

Retourne la largeur des couches affichables, en pixels.

js	function get_layerWidth ()
cpp	int get_layerWidth ()
m	-(int) layerWidth
pas	LongInt get_layerWidth (): LongInt
vb	function get_layerWidth () As Integer
cs	int get_layerWidth ()
java	int get_layerWidth ()
uwp	async Task<int> get_layerWidth ()
py	get_layerWidth ()
php	function get_layerWidth ()
ts	async get_layerWidth (): Promise<number>
es	async get_layerWidth ()
dnp	int get_layerWidth ()
cp	int get_layerWidth ()
cmd	YDisplay target [-layer layerId] get_layerWidth

Retourne :

un entier représentant la largeur des couches affichables, en pixels

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

displaylayer→hide()**YDisplayLayer**

Cache la couche de dessin.

js	function hide ()
c++	int hide ()
m	-(int) hide
pas	LongInt hide (): LongInt
vb	function hide () As Integer
cs	int hide ()
java	int hide ()
uwp	async Task<int> hide ()
py	hide ()
php	function hide ()
ts	async hide (): Promise<number>
es	async hide ()
dnp	int hide ()
cp	int hide ()
cmd	YDisplay target [-layer layerId] hide

L'état de la couche est préservé, mais la couche ne sera plus plus affichés à l'écran jusqu'au prochain appel à `unhide()`. Le fait de cacher la couche améliore les performances de toutes les primitives d'affichage, car il évite de consacrer inutilement des cycles de calcul à afficher les états intermédiaires (technique de double-buffering).

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→lineTo()

YDisplayLayer

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

js	function lineTo (x , y)
cpp	int lineTo (int x , int y)
m	-(int) lineTo : (int) x : (int) y
pas	LongInt lineTo (x : LongInt, y : LongInt): LongInt
vb	function lineTo (ByVal x As Integer, ByVal y As Integer) As Integer
cs	int lineTo (int x , int y)
java	int lineTo (int x , int y)
uwp	async Task<int> lineTo (int x , int y)
py	lineTo (x , y)
php	function lineTo (\$x , \$y)
ts	async lineTo (x : number, y : number): Promise<number>
es	async lineTo (x , y)
dnp	int lineTo (int x , int y)
cp	int lineTo (int x , int y)
cml	YDisplay target [-layer layerId] lineTo x y

Le pixel final spécifié est inclus dans la ligne dessinée. Le point de dessin courant est déplacé à au point final de la ligne.

Paramètres :

x la distance en pixels depuis la gauche de la couche jusqu'au point final

y la distance en pixels depuis le haut de la couche jusqu'au point final

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

Déplace le point de dessin courant de cette couche à la position spécifiée.

js	function moveTo (x , y)
c++	int moveTo (int x , int y)
m	-(int) moveTo : (int) x : (int) y
pas	LongInt moveTo (x : LongInt, y : LongInt): LongInt
vb	function moveTo (ByVal x As Integer, ByVal y As Integer) As Integer
cs	int moveTo (int x , int y)
java	int moveTo (int x , int y)
uwp	async Task<int> moveTo (int x , int y)
py	moveTo (x , y)
php	function moveTo (\$x , \$y)
ts	async moveTo (x : number, y : number): Promise<number>
es	async moveTo (x , y)
dnp	int moveTo (int x , int y)
cp	int moveTo (int x , int y)
cmd	YDisplay target [-layer layerId] moveTo x y

Paramètres :

x la distance en pixels depuis la gauche de la couche de dessin

y la distance en pixels depuis le haut de la couche de dessin

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→reset()**YDisplayLayer**

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

js	function reset ()
cpp	int reset ()
m	-(int) reset
pas	LongInt reset (): LongInt
vb	function reset () As Integer
cs	int reset ()
java	int reset ()
uwp	async Task<int> reset ()
py	reset ()
php	function reset ()
ts	async reset (): Promise<number>
es	async reset ()
dnp	int reset ()
cp	int reset ()
cmd	YDisplay target [-layer layerId] reset

Réinitialise la position du point de dessin courant au coin supérieur gauche, et la couleur de dessin à la valeur la plus lumineuse. Si vous désirez simplement effacer le contenu de la couche, utilisez plutôt la méthode `clear()`.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→selectColorPen()**YDisplayLayer**

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

js	function selectColorPen (color)
cpp	int selectColorPen (int color)
m	-(int) selectColorPen : (int) color
pas	LongInt selectColorPen (color : LongInt): LongInt
vb	function selectColorPen (ByVal color As Integer) As Integer
cs	int selectColorPen (int color)
java	int selectColorPen (int color)
uwp	async Task<int> selectColorPen (int color)
py	selectColorPen (color)
php	function selectColorPen (\$color)
ts	async selectColorPen (color : number): Promise<number>
es	async selectColorPen (color)
dnp	int selectColorPen (int color)
cp	int selectColorPen (int color)
cmd	YDisplay target [-layer layerId] selectColorPen color

La couleur est fournie sous forme de couleur RGB. Pour les écrans monochromes ou en niveaux de gris, la couleur est automatiquement ramenée dans les valeurs permises.

Paramètres :

color la couleur RGB désirée (sous forme d'entier 24 bits)

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→selectEraser()**YDisplayLayer**

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de copie d'images bitmaps.

js	function selectEraser ()
cpp	int selectEraser ()
m	-(int) selectEraser
pas	LongInt selectEraser (): LongInt
vb	function selectEraser () As Integer
cs	int selectEraser ()
java	int selectEraser ()
uwp	async Task<int> selectEraser ()
py	selectEraser ()
php	function selectEraser ()
ts	async selectEraser (): Promise<number>
es	async selectEraser ()
dnp	int selectEraser ()
cp	int selectEraser ()
cmd	YDisplay target [-layer layerId] selectEraser

Tous les points dessinés à la gomme redeviennent transparents (comme ils l'étaient lorsque la couche était vide), rendant ainsi visibles les couches inférieures.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→selectFont()**YDisplayLayer**

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

js	function selectFont (fontname)
c++	int selectFont (string fontname)
m	-(int) selectFont : (NSString*) fontname
pas	LongInt selectFont (fontname : string): LongInt
vb	function selectFont (ByVal fontname As String) As Integer
cs	int selectFont (string fontname)
java	int selectFont (String fontname)
uwp	async Task<int> selectFont (string fontname)
py	selectFont (fontname)
php	function selectFont (\$fontname)
ts	async selectFont (fontname : string): Promise<number>
es	async selectFont (fontname)
dnp	int selectFont (string fontname)
cp	int selectFont (string fontname)
cmd	YDisplay target [-layer layerId] selectFont fontname

La police est spécifiée par le nom de son fichier. Vous pouvez utiliser l'une des polices prédéfinies dans le module, ou une autre police que vous avez préalablement préchargé dans la mémoire du module. Si vous rencontrez des problèmes à l'utilisation d'une police de caractères, consultez les logs du module pour voir si vous n'y trouvez pas un message à propos d'un fichier de police manquant ou d'un format de fichier invalide.

Paramètres :

fontname le nom du fichier définissant la police de caractères. Les polices intégrées sont 8x8.yfm, Small.yfm, Medium.yfm, Large.yfm (sauf Yocto-MiniDisplay).

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→selectGrayPen()**YDisplayLayer**

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

js	function selectGrayPen (graylevel)
c++	int selectGrayPen (int graylevel)
m	-(int) selectGrayPen : (int) graylevel
pas	LongInt selectGrayPen (graylevel : LongInt): LongInt
vb	function selectGrayPen (ByVal graylevel As Integer) As Integer
cs	int selectGrayPen (int graylevel)
java	int selectGrayPen (int graylevel)
uwp	async Task<int> selectGrayPen (int graylevel)
py	selectGrayPen (graylevel)
php	function selectGrayPen (\$graylevel)
ts	async selectGrayPen (graylevel : number): Promise<number>
es	async selectGrayPen (graylevel)
dnp	int selectGrayPen (int graylevel)
cp	int selectGrayPen (int graylevel)
cmd	YDisplay target [-layer layerId] selectGrayPen graylevel

Le niveau de gris est fourni sous forme d'un chiffre allant de 0 (noir) à 255 (blanc, ou la couleur la plus claire de l'écran, quelle qu'elle soit). Pour les écrans monochromes (sans niveaux de gris), toute valeur inférieure à 128 conduit à un point noir, et toute valeur supérieure ou égale à 128 devient un point lumineux.

Paramètres :

graylevel le niveau de gris désiré, de 0 à 255

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→setAntialiasingMode()**YDisplayLayer**

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

js	function setAntialiasingMode (mode)
cpp	int setAntialiasingMode (bool mode)
m	-(int) setAntialiasingMode : (bool) mode
pas	LongInt setAntialiasingMode (mode : boolean): LongInt
vb	function setAntialiasingMode (ByVal mode As Boolean) As Integer
cs	int setAntialiasingMode (bool mode)
java	int setAntialiasingMode (boolean mode)
uwp	async Task<int> setAntialiasingMode (bool mode)
py	setAntialiasingMode (mode)
php	function setAntialiasingMode (\$mode)
ts	async setAntialiasingMode (mode : boolean): Promise<number>
es	async setAntialiasingMode (mode)
dnp	int setAntialiasingMode (bool mode)
cp	int setAntialiasingMode (bool mode)
cmd	YDisplay target [-layer layerId] setAntialiasingMode mode

L'anti-aliasing est atténue la pixelisation des images lorsqu'on regarde l'écran depuis une distance suffisante, mais peut aussi donner parfois une impression de flou lorsque l'écran est regardé de très près. Au final, c'est un choix esthétique qui vous revient. L'anti-aliasing est activé par défaut pour les écrans en niveaux de gris et les écrans couleurs, mais vous pouvez le désactiver si vous préférez. Ce réglage n'a pas d'effet sur les écrans monochromes.

Paramètres :

mode true pour activer l'antialiasing, false pour le désactiver.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→setConsoleBackground()**YDisplayLayer**

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

js	function setConsoleBackground (bgcol)
cpp	int setConsoleBackground (int bgcol)
m	-(int) setConsoleBackground : (int) bgcol
pas	LongInt setConsoleBackground (bgcol : LongInt): LongInt
vb	function setConsoleBackground (ByVal bgcol As Integer) As Integer
cs	int setConsoleBackground (int bgcol)
java	int setConsoleBackground (int bgcol)
uwp	async Task<int> setConsoleBackground (int bgcol)
py	setConsoleBackground (bgcol)
php	function setConsoleBackground (\$bgcol)
ts	async setConsoleBackground (bgcol : number): Promise<number>
es	async setConsoleBackground (bgcol)
dnp	int setConsoleBackground (int bgcol)
cp	int setConsoleBackground (int bgcol)
cmd	YDisplay target [-layer layerId] setConsoleBackground bgcol

Paramètres :

bgcol le niveau de gris à utiliser pour le fond lors de défilement (0 = noir, 255 = blanc), ou -1 pour un fond transparent

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→setConsoleMargins()

YDisplayLayer

Configure les marges d'affichage pour la fonction `consoleOut`.

js	function setConsoleMargins(x1, y1, x2, y2)
cpp	int setConsoleMargins(int x1, int y1, int x2, int y2)
m	-(int) setConsoleMargins : (int) x1 : (int) y1 : (int) x2 : (int) y2
pas	LongInt setConsoleMargins(x1: LongInt, y1: LongInt, x2: LongInt, y2: LongInt): LongInt
vb	function setConsoleMargins(ByVal x1 As Integer, ByVal y1 As Integer, ByVal x2 As Integer, ByVal y2 As Integer) As Integer
cs	int setConsoleMargins(int x1, int y1, int x2, int y2)
java	int setConsoleMargins(int x1, int y1, int x2, int y2)
uwp	async Task<int> setConsoleMargins(int x1, int y1, int x2, int y2)
py	setConsoleMargins(x1, y1, x2, y2)
php	function setConsoleMargins(\$x1, \$y1, \$x2, \$y2)
ts	async setConsoleMargins(x1: number, y1: number, x2: number, y2: number): Promise<number>
es	async setConsoleMargins(x1, y1, x2, y2)
dnp	int setConsoleMargins(int x1, int y1, int x2, int y2)
cp	int setConsoleMargins(int x1, int y1, int x2, int y2)
cmd	YDisplay target [-layer layerId] setConsoleMargins x1 y1 x2 y2

Paramètres :

- x1** la distance en pixels depuis la gauche de la couche jusqu'à la marge gauche
- y1** la distance en pixels depuis le haut de la couche jusqu'à la marge supérieure
- x2** la distance en pixels depuis la gauche de la couche jusqu'à la marge droite
- y2** la distance en pixels depuis le haut de la couche jusqu'à la marge inférieure

Retourne :

YAPI .SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→setConsoleWordWrap()**YDisplayLayer**

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

js	function setConsoleWordWrap (wordwrap)
cpp	int setConsoleWordWrap (bool wordwrap)
m	-(int) setConsoleWordWrap : (bool) wordwrap
pas	LongInt setConsoleWordWrap (wordwrap : boolean): LongInt
vb	function setConsoleWordWrap (ByVal wordwrap As Boolean) As Integer
cs	int setConsoleWordWrap (bool wordwrap)
java	int setConsoleWordWrap (boolean wordwrap)
uwp	async Task<int> setConsoleWordWrap (bool wordwrap)
py	setConsoleWordWrap (wordwrap)
php	function setConsoleWordWrap (\$ wordwrap)
ts	async setConsoleWordWrap (wordwrap : boolean): Promise<number>
es	async setConsoleWordWrap (wordwrap)
dnp	int setConsoleWordWrap (bool wordwrap)
cp	int setConsoleWordWrap (bool wordwrap)
cmd	YDisplay target [-layer layerId] setConsoleWordWrap wordwrap

Paramètres :

wordwrap `true` pour retourner à la ligne entre les mots seulements, `false` pour retourner à l'extrême droite de chaque ligne.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→setLayerPosition()

YDisplayLayer

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

js	function setLayerPosition (x , y , scrollTime)
cpp	int setLayerPosition (int x , int y , int scrollTime)
m	-(int) setLayerPosition : (int) x : (int) y : (int) scrollTime
pas	LongInt setLayerPosition (x : LongInt, y : LongInt, scrollTime : LongInt): LongInt
vb	function setLayerPosition (ByVal x As Integer, ByVal y As Integer, ByVal scrollTime As Integer) As Integer
cs	int setLayerPosition (int x , int y , int scrollTime)
java	int setLayerPosition (int x , int y , int scrollTime)
uwp	async Task<int> setLayerPosition (int x , int y , int scrollTime)
py	setLayerPosition (x , y , scrollTime)
php	function setLayerPosition (\$x , \$y , \$scrollTime)
ts	async setLayerPosition (x : number, y : number, scrollTime : number): Promise<number>
es	async setLayerPosition (x , y , scrollTime)
dnp	int setLayerPosition (int x , int y , int scrollTime)
cp	int setLayerPosition (int x , int y , int scrollTime)
cmd	YDisplay target [-layer layerId] setLayerPosition x y scrollTime

Lorsqu'une durée de défilement est configurée, la position d'affichage de la couche est automatiquement mise à jour durant les millisecondes suivantes pour animer le déplacement.

Paramètres :

x	la distance en pixels depuis la gauche de l'écran jusqu'à l'origine de la couche.
y	la distance en pixels depuis le haut de l'écran jusqu'à l'origine de la couche.
scrollTime	durée en millisecondes du déplacement, ou 0 si le déplacement doit être immédiat.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

displaylayer→unhide()**YDisplayLayer**

Affiche la couche.

js	function unhide ()
cpp	int unhide ()
m	-(int) unhide
pas	LongInt unhide (): LongInt
vb	function unhide () As Integer
cs	int unhide ()
java	int unhide ()
uwp	async Task<int> unhide ()
py	unhide ()
php	function unhide ()
ts	async unhide (): Promise<number>
es	async unhide ()
dnp	int unhide ()
cp	int unhide ()
cmd	YDisplay target [-layer layerId] unhide

Affiche a nouveau la couche après la command hide.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

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

24.5. La classe YAnButton

Interface pour interagir avec les entrées analogiques, disponibles par exemple dans le Yocto-Buzzer, le Yocto-Knob, le Yocto-MaxiBuzzer et le Yocto-MaxiDisplay

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

es	in HTML: <code><script src="../../lib/yocto_anbutton.js"></script></code> in node.js: <code>require('yoctolib-es2017/yocto_anbutton.js');</code>
js	<code><script type='text/javascript' src='yocto_anbutton.js'></script></code>
cpp	<code>#include "yocto_anbutton.h"</code>
m	<code>#import "yocto_anbutton.h"</code>
pas	<code>uses yocto_anbutton;</code>
vb	<code>yocto_anbutton.vb</code>
cs	<code>yocto_anbutton.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YAnButton;</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YAnButton;</code>
py	<code>from yocto_anbutton import *</code>
php	<code>require_once('yocto_anbutton.php');</code>
ts	in HTML: <code>import { YAnButton } from '../../dist/esm/yocto_anbutton.js';</code> in Node.js: <code>import { YAnButton } from 'yoctolib-cjs/yocto_anbutton.js';</code>
dnf	<code>import YoctoProxyAPI.YAnButtonProxy</code>
cp	<code>#include "yocto_anbutton_proxy.h"</code>
vi	<code>YAnButton.vi</code>
ml	<code>import YoctoProxyAPI.YAnButtonProxy</code>

Fonction globales

YAnButton.FindAnButton(func)

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

YAnButton.FindAnButtonInContext(yctx, func)

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

YAnButton.FirstAnButton()

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

YAnButton.FirstAnButtonInContext(yctx)

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

YAnButton.GetSimilarFunctions()

Enumère toutes les fonctions de type `AnButton` disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (`hardwareId`).

Propriétés des objets `YAnButtonProxy`

anbutton→AdvertisedValue [lecture seule]

Courte chaîne de caractères représentant l'état courant de la fonction.

anbutton→AnalogCalibration [modifiable]

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

anbutton→CalibratedValue [lecture seule]

Valeur calibrée de l'entrée (entre 0 et 1000 inclus).

anbutton→CalibrationMax [modifiable]

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

anbutton→CalibrationMin [modifiable]

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

anbutton→FriendlyName [lecture seule]

Identifiant global de la fonction au format `NOM_MODULE . NOM_FONCTION`.

anbutton→FunctionId [lecture seule]

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

anbutton→HardwareId [lecture seule]

Identifiant matériel unique de la fonction au format `SERIAL . FUNCTIONID`.

anbutton→InputType [modifiable]

Type de décodage appliqué à l'entrée (entrée analogique ou entrées binaires multiplexées).

anbutton→IsOnline [lecture seule]

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

anbutton→IsPressed [lecture seule]

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

anbutton→LogicalName [modifiable]

Nom logique de la fonction.

anbutton→Sensitivity [modifiable]

Sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks.

anbutton→SerialNumber [lecture seule]

Numéro de série du module, préprogrammé en usine.

Méthodes des objets `YAnButton`

anbutton→clearCache()

Invalide le cache.

anbutton→describe()

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

anbutton→get_advertisedValue()

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

anbutton→get_analogCalibration()

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

anbutton→get_calibratedValue()

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

anbutton→get_calibrationMax()

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

anbutton→get_calibrationMin()

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

anbutton→get_errorMessage()

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

anbutton→get_errorType()

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

anbutton→get_friendlyName()

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

`anbutton→get_functionDescriptor()`

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

`anbutton→get_functionId()`

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

`anbutton→get_hardwareId()`

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

`anbutton→get_inputType()`

Retourne le type de décodage appliqué à l'entrée (entrée analogique ou entrées binaires multiplexées).

`anbutton→get_isPressed()`

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

`anbutton→get_lastTimePressed()`

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

`anbutton→get_lastTimeReleased()`

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

`anbutton→get_logicalName()`

Retourne le nom logique de l'entrée analogique.

`anbutton→get_module()`

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

`anbutton→get_module_async(callback, context)`

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

`anbutton→get_pulseCounter()`

Retourne la valeur du compteur d'impulsions.

`anbutton→get_pulseTimer()`

Retourne le timer du compteur d'impulsions (ms).

`anbutton→get_rawValue()`

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

`anbutton→get_sensitivity()`

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

`anbutton→get_serialNumber()`

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

`anbutton→get_userData()`

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

`anbutton→isOnline()`

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

`anbutton→isOnline_async(callback, context)`

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

`anbutton→isReadOnly()`

Test si la fonction est en lecture seule.

`anbutton→load(msValidity)`

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

`anbutton→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.

anbutton→load_async(msValidity, callback, context)

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

anbutton→muteValueCallbacks()

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

anbutton→nextAnButton()

Continue l'énumération des entrées analogiques commencée à l'aide de `yFirstAnButton()`. Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les entrées analogiques sont retournés.

anbutton→registerValueCallback(callback)

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

anbutton→resetCounter()

Réinitialise le compteur d'impulsions et son timer.

anbutton→set_analogCalibration(newval)

Enclenche ou déclenche le procédure de calibration.

anbutton→set_calibrationMax(newval)

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

anbutton→set_calibrationMin(newval)

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

anbutton→set_inputType(newval)

Modifie le type de décodage appliqué à l'entrée (entrée analogique ou entrées binaires multiplexées).

anbutton→set_logicalName(newval)

Modifie le nom logique de l'entrée analogique.

anbutton→set_sensitivity(newval)

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

anbutton→set_userData(data)

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

anbutton→unmuteValueCallbacks()

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

anbutton→wait_async(callback, context)

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

YAnButton.FindAnButton()

YAnButton.FindAnButton()

YAnButton

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

js	function yFindAnButton (func)
cpp	YAnButton* FindAnButton (string func)
m	+(YAnButton*) FindAnButton : (NSString*) func
pas	TYAnButton yFindAnButton (func : string): TYAnButton
vb	function FindAnButton (ByVal func As String) As YAnButton
cs	static YAnButton FindAnButton (string func)
java	static YAnButton FindAnButton (String func)
uwp	static YAnButton FindAnButton (string func)
py	FindAnButton (func)
php	function FindAnButton (\$func)
ts	static FindAnButton (func : string): YAnButton
es	static FindAnButton (func)
dnp	static YAnButtonProxy FindAnButton (string func)
cp	static YAnButtonProxy * FindAnButton (string func)

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Si un appel à la méthode `is_online()` de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler `registerHub()` à l'initialisation de l'application.

Paramètres :

func une chaîne de caractères qui référence l'entrée analogique sans ambiguïté, par exemple `YBUZZER2.anButton1`.

Retourne :

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

YAnButton.FindAnButtonInContext()

YAnButton.FindAnButtonInContext()

YAnButton

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

java	static YAnButton FindAnButtonInContext (YAPIContext yctx , String func)
uwp	static YAnButton FindAnButtonInContext (YAPIContext yctx , string func)
ts	static FindAnButtonInContext (yctx : YAPIContext, func : string): YAnButton
es	static FindAnButtonInContext (yctx , func)

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence l'entrée analogique sans ambiguïté, par exemple `YBUZZER2.anButton1`.

Retourne :

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

YAnButton.FirstAnButton()**YAnButton****YAnButton.FirstAnButton()**

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

js	function yFirstAnButton ()
c++	YAnButton * FirstAnButton ()
m	+(YAnButton*) FirstAnButton
pas	TYAnButton yFirstAnButton (): TYAnButton
vb	function FirstAnButton () As YAnButton
cs	static YAnButton FirstAnButton ()
java	static YAnButton FirstAnButton ()
uwp	static YAnButton FirstAnButton ()
py	FirstAnButton ()
php	function FirstAnButton ()
ts	static FirstAnButton (): YAnButton null
es	static FirstAnButton ()

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

Retourne :

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

YAnButton.FirstAnButtonInContext() YAnButton.FirstAnButtonInContext()

YAnButton

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

java	static YAnButton FirstAnButtonInContext (YAPIContext yctx)
uwp	static YAnButton FirstAnButtonInContext (YAPIContext yctx)
ts	static FirstAnButtonInContext (yctx : YAPIContext): YAnButton null
es	static FirstAnButtonInContext (yctx)

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

Paramètres :

yctx un contexte YAPI.

Retourne :

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

YAnButton.GetSimilarFunctions()**YAnButton****YAnButton.GetSimilarFunctions()**

Enumère toutes les fonctions de type AnButton disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

dnp

```
static new string[] GetSimilarFunctions( )
```

cp

```
static vector<string> GetSimilarFunctions( )
```

Chaque chaîne retournée peut être passée en argument à la méthode `YAnButton.FindAnButton` pour obtenir un objet permettant d'interagir avec le module correspondant.

Retourne :

un tableau de chaînes de caractères, contenant les identifiants matériels de chaque fonction disponible trouvée.

anbutton→AdvertisedValue**YAnButton**

Courte chaîne de caractères représentant l'état courant de la fonction.

dnp

 string **AdvertisedValue**

anbutton→**AnalogCalibration****YAnButton**

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

dnp **int** **AnalogCalibration**

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

anbutton→**CalibratedValue****YAnButton**

Valeur calibrée de l'entrée (entre 0 et 1000 inclus).

dnp

int CalibratedValue

anbutton→CalibrationMax**YAnButton**

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

dnf **int CalibrationMax**

Modifiable. Modifie la valeur maximale de calibration pour l'entrée (entre 0 et 4095 inclus), sans lancer la calibration automatique. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

anbutton→CalibrationMin**YAnButton**

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

dnf **int CalibrationMin**

Modifiable. Modifie la valeur minimale de calibration pour l'entrée (entre 0 et 4095 inclus), sans lancer la calibration automatique. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

anbutton→FriendlyName**YAnButton**

Identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

`dnf` `string` **FriendlyName**

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

anbutton→FunctionId**YAnButton**

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

`dn` `string` **FunctionId**

Par exemple `relay1`.

anbutton→**HardwareId****YAnButton**

Identifiant matériel unique de la fonction au format `SERIAL.FUNCTIONID`.

`dnf` `string` **HardwareId**

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

anbutton→InputType**YAnButton**

Type de décodage appliqué à l'entrée (entrée analogique ou entrées binaires multiplexées).

dnf **int InputType**

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

anbutton→IsOnline**YAnButton**

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

dnp

bool IsOnline

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

anbutton→IsPressed**YAnButton**

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

dnp

int IsPressed

anbutton→LogicalName**YAnButton**

Nom logique de la fonction.

dnp `string LogicalName`

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

anbutton→Sensitivity**YAnButton**

Sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks.

dnf **int Sensitivity**

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

anbutton→SerialNumber

YAnButton

Numéro de série du module, préprogrammé en usine.

dnf string **SerialNumber**

anbutton→clearCache()**YAnButton**

Invalide le cache.

js	function clearCache ()
cpp	void clearCache ()
m	-(void) clearCache
pas	clearCache ()
vb	procedure clearCache ()
cs	void clearCache ()
java	void clearCache ()
py	clearCache ()
php	function clearCache ()
ts	async clearCache (): Promise<void>
es	async clearCache ()

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

anbutton→describe()**YAnButton**

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

js	function describe ()
cpp	string describe ()
m	-(NSString*) describe
pas	string describe (): string
vb	function describe () As String
cs	string describe ()
java	String describe ()
py	describe ()
php	function describe ()
ts	async describe (): Promise<string>
es	async describe ()

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

Retourne :

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

anbutton→get_advertisedValue()**YAnButton****anbutton→advertisedValue()**

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

js	function get_advertisedValue ()
cpp	string get_advertisedValue ()
m	-(NSString*) advertisedValue
pas	string get_advertisedValue (): string
vb	function get_advertisedValue () As String
cs	string get_advertisedValue ()
java	String get_advertisedValue ()
uwp	async Task<string> get_advertisedValue ()
py	get_advertisedValue ()
php	function get_advertisedValue ()
ts	async get_advertisedValue (): Promise<string>
es	async get_advertisedValue ()
dnp	string get_advertisedValue ()
cp	string get_advertisedValue ()
cmd	YAnButton target get_advertisedValue

Retourne :

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

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

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

Permet de savoir si une procédure de calibration est actuellement en cours.

js	function get_analogCalibration ()
cpp	Y_ANALOGCALIBRATION_enum get_analogCalibration ()
m	-(Y_ANALOGCALIBRATION_enum) analogCalibration
pas	Integer get_analogCalibration (): Integer
vb	function get_analogCalibration () As Integer
cs	int get_analogCalibration ()
java	int get_analogCalibration ()
uwp	async Task<int> get_analogCalibration ()
py	get_analogCalibration ()
php	function get_analogCalibration ()
ts	async get_analogCalibration (): Promise<YAnButton_AnalogCalibration>
es	async get_analogCalibration ()
dnp	int get_analogCalibration ()
cp	int get_analogCalibration ()
cmd	YAnButton target get_analogCalibration

Retourne :

soit YAnButton.ANALOGCALIBRATION_OFF, soit YAnButton.ANALOGCALIBRATION_ON

En cas d'erreur, déclenche une exception ou retourne YAnButton.ANALOGCALIBRATION_INVALID.

anbutton→get_calibratedValue()**YAnButton****anbutton→calibratedValue()**

Retourne la valeur calibrée de l'entrée (entre 0 et 1000 inclus).

js	function get_calibratedValue ()
cpp	int get_calibratedValue ()
m	-(int) calibratedValue
pas	LongInt get_calibratedValue (): LongInt
vb	function get_calibratedValue () As Integer
cs	int get_calibratedValue ()
java	int get_calibratedValue ()
uwp	async Task<int> get_calibratedValue ()
py	get_calibratedValue ()
php	function get_calibratedValue ()
ts	async get_calibratedValue (): Promise<number>
es	async get_calibratedValue ()
dnp	int get_calibratedValue ()
cp	int get_calibratedValue ()
cmd	YAnButton target get_calibratedValue

Retourne :

un entier représentant la valeur calibrée de l'entrée (entre 0 et 1000 inclus)

En cas d'erreur, déclenche une exception ou retourne `YAnButton.CALIBRATEDVALUE_INVALID`.

anbutton→**get_calibrationMax()****YAnButton****anbutton**→**calibrationMax()**

Retourne la valeur maximale observée durant la calibration (entre 0 et 4095 inclus).

js	function get_calibrationMax ()
cpp	int get_calibrationMax ()
m	-(int) calibrationMax
pas	LongInt get_calibrationMax (): LongInt
vb	function get_calibrationMax () As Integer
cs	int get_calibrationMax ()
java	int get_calibrationMax ()
uwp	async Task<int> get_calibrationMax ()
py	get_calibrationMax ()
php	function get_calibrationMax ()
ts	async get_calibrationMax (): Promise<number>
es	async get_calibrationMax ()
dnp	int get_calibrationMax ()
cp	int get_calibrationMax ()
cmd	YAnButton target get_calibrationMax

Retourne :

un entier représentant la valeur maximale observée durant la calibration (entre 0 et 4095 inclus)

En cas d'erreur, déclenche une exception ou retourne `YAnButton.CALIBRATIONMAX_INVALID`.

anbutton→get_calibrationMin()**YAnButton****anbutton→calibrationMin()**

Retourne la valeur minimale observée durant la calibration (entre 0 et 4095 inclus).

js	function get_calibrationMin ()
cpp	int get_calibrationMin ()
m	-(int) calibrationMin
pas	LongInt get_calibrationMin (): LongInt
vb	function get_calibrationMin () As Integer
cs	int get_calibrationMin ()
java	int get_calibrationMin ()
uwp	async Task<int> get_calibrationMin ()
py	get_calibrationMin ()
php	function get_calibrationMin ()
ts	async get_calibrationMin (): Promise<number>
es	async get_calibrationMin ()
dnp	int get_calibrationMin ()
cp	int get_calibrationMin ()
cmd	YAnButton target get_calibrationMin

Retourne :

un entier représentant la valeur minimale observée durant la calibration (entre 0 et 4095 inclus)

En cas d'erreur, déclenche une exception ou retourne `YAnButton.CALIBRATIONMIN_INVALID`.

anbutton→get_errorMessage()**YAnButton****anbutton→errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'entrée analogique.

js	function get_errorMessage ()
cpp	string get_errorMessage ()
m	-(NSString*) errorMessage
pas	string get_errorMessage (): string
vb	function get_errorMessage () As String
cs	string get_errorMessage ()
java	String get_errorMessage ()
py	get_errorMessage ()
php	function get_errorMessage ()
ts	get_errorMessage (): string
es	get_errorMessage ()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de l'entrée analogique.

anbutton→get_errorType()**YAnButton****anbutton→errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'entrée analogique.

js	function get_errorType ()
cpp	YRETCODE get_errorType ()
m	-(YRETCODE) errorType
pas	YRETCODE get_errorType (): YRETCODE
vb	function get_errorType () As YRETCODE
cs	YRETCODE get_errorType ()
java	int get_errorType ()
py	get_errorType ()
php	function get_errorType ()
ts	get_errorType (): number
es	get_errorType ()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de l'entrée analogique.

anbutton→**get_friendlyName()****YAnButton****anbutton**→**friendlyName()**

Retourne un identifiant global de l'entrée analogique au format `NOM_MODULE.NOM_FONCTION`.

js	function get_friendlyName ()
cpp	string get_friendlyName ()
m	-(NSString*) friendlyName
cs	string get_friendlyName ()
java	String get_friendlyName ()
py	get_friendlyName ()
php	function get_friendlyName ()
ts	async get_friendlyName (): Promise<string>
es	async get_friendlyName ()
dnp	string get_friendlyName ()
cp	string get_friendlyName ()

Le chaîne retournée utilise soit les noms logiques du module et de l'entrée analogique si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de l'entrée analogique (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant l'entrée analogique en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `YAnButton.FRIENDLYNAME_INVALID`.

anbutton→get_functionDescriptor()**YAnButton****anbutton→functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

js	function get_functionDescriptor ()
cpp	YFUN_DESCR get_functionDescriptor ()
m	-(YFUN_DESCR) functionDescriptor
pas	YFUN_DESCR get_functionDescriptor (): YFUN_DESCR
vb	function get_functionDescriptor () As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor ()
java	String get_functionDescriptor ()
py	get_functionDescriptor ()
php	function get_functionDescriptor ()
ts	async get_functionDescriptor (): Promise<string>
es	async get_functionDescriptor ()

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y\$CLASSNAME\$.FUNCTIONDESCRIPTOR_INVALID

anbutton→get_functionId()**YAnButton****anbutton→functionId()**

Retourne l'identifiant matériel de l'entrée analogique, sans référence au module.

js	function get_functionId ()
cpp	string get_functionId ()
m	-(NSString*) functionId
vb	function get_functionId () As String
cs	string get_functionId ()
java	String get_functionId ()
py	get_functionId ()
php	function get_functionId ()
ts	async get_functionId (): Promise<string>
es	async get_functionId ()
dnp	string get_functionId ()
cp	string get_functionId ()

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant l'entrée analogique (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `YAnButton.FUNCTIONID_INVALID`.

anbutton→**get_hardwareId()****YAnButton****anbutton**→**hardwareId()**

Retourne l'identifiant matériel unique de l'entrée analogique au format `SERIAL.FUNCTIONID`.

js	function get_hardwareId ()
cpp	string get_hardwareId ()
m	-(NSString*) hardwareId
vb	function get_hardwareId () As String
cs	string get_hardwareId ()
java	String get_hardwareId ()
py	get_hardwareId ()
php	function get_hardwareId ()
ts	async get_hardwareId (): Promise<string>
es	async get_hardwareId ()
dnp	string get_hardwareId ()
cp	string get_hardwareId ()

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de l'entrée analogique (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant l'entrée analogique (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `YAnButton.HARDWAREID_INVALID`.

anbutton→get_inputType()**YAnButton****anbutton→inputType()**

Retourne le type de décodage appliqué à l'entrée (entrée analogique ou entrées binaires multiplexées).

js	function get_inputType ()
cpp	Y_INPUTTYPE_enum get_inputType ()
m	-(Y_INPUTTYPE_enum) inputType
pas	Integer get_inputType (): Integer
vb	function get_inputType () As Integer
cs	int get_inputType ()
java	int get_inputType ()
uwp	async Task<int> get_inputType ()
py	get_inputType ()
php	function get_inputType ()
ts	async get_inputType (): Promise<YAnButton_InputType>
es	async get_inputType ()
dnp	int get_inputType ()
cp	int get_inputType ()
cmd	YAnButton target get_inputType

Retourne :

une valeur parmi `YAnButton.INPUTTYPE_ANALOG_FAST`, `YAnButton.INPUTTYPE_DIGITAL4` et `YAnButton.INPUTTYPE_ANALOG_SMOOTH` représentant le type de décodage appliqué à l'entrée (entrée analogique ou entrées binaires multiplexées)

En cas d'erreur, déclenche une exception ou retourne `YAnButton.INPUTTYPE_INVALID`.

anbutton→get_isPressed()**YAnButton****anbutton→isPressed()**

Retourne vrai si l'entrée (considérée comme binaire) est active (contact fermé), et faux sinon.

js	function get_isPressed ()
cpp	Y_ISPRESSED_enum get_isPressed ()
m	-(Y_ISPRESSED_enum) isPressed
pas	Integer get_isPressed (): Integer
vb	function get_isPressed () As Integer
cs	int get_isPressed ()
java	int get_isPressed ()
uwp	async Task<int> get_isPressed ()
py	get_isPressed ()
php	function get_isPressed ()
ts	async get_isPressed (): Promise<YAnButton_IsPressed>
es	async get_isPressed ()
dnp	int get_isPressed ()
cp	int get_isPressed ()
cmd	YAnButton target get_isPressed

Retourne :

soit `YAnButton.ISPRESSED_FALSE`, soit `YAnButton.ISPRESSED_TRUE`, selon vrai si l'entrée (considérée comme binaire) est active (contact fermé), et faux sinon

En cas d'erreur, déclenche une exception ou retourne `YAnButton.ISPRESSED_INVALID`.

anbutton→get_lastTimePressed()**YAnButton****anbutton→lastTimePressed()**

Retourne le temps absolu (nombre de millisecondes) entre la mise sous tension du module et la dernière pression observée du bouton à l'entrée (transition du contact de ouvert à fermé).

js	function get_lastTimePressed ()
cpp	s64 get_lastTimePressed ()
m	-(s64) lastTimePressed
pas	int64 get_lastTimePressed (): int64
vb	function get_lastTimePressed () As Long
cs	long get_lastTimePressed ()
java	long get_lastTimePressed ()
uwp	async Task<long> get_lastTimePressed ()
py	get_lastTimePressed ()
php	function get_lastTimePressed ()
ts	async get_lastTimePressed (): Promise<number>
es	async get_lastTimePressed ()
dnp	long get_lastTimePressed ()
cp	s64 get_lastTimePressed ()
cmd	YAnButton target get_lastTimePressed

Retourne :

un entier représentant le temps absolu (nombre de millisecondes) entre la mise sous tension du module et la dernière pression observée du bouton à l'entrée (transition du contact de ouvert à fermé)

En cas d'erreur, déclenche une exception ou retourne `YAnButton.LASTTIMEPRESSED_INVALID`.

anbutton→get_lastTimeReleased()**YAnButton****anbutton→lastTimeReleased()**

Retourne le temps absolu (nombre de millisecondes) entre la mise sous tension du module et le dernier relâchement observée du bouton à l'entrée (transition du contact de fermé à ouvert).

js	function get_lastTimeReleased ()
cpp	s64 get_lastTimeReleased ()
m	-(s64) lastTimeReleased
pas	int64 get_lastTimeReleased (): int64
vb	function get_lastTimeReleased () As Long
cs	long get_lastTimeReleased ()
java	long get_lastTimeReleased ()
uwp	async Task<long> get_lastTimeReleased ()
py	get_lastTimeReleased ()
php	function get_lastTimeReleased ()
ts	async get_lastTimeReleased (): Promise<number>
es	async get_lastTimeReleased ()
dnp	long get_lastTimeReleased ()
cp	s64 get_lastTimeReleased ()
cmd	YAnButton target get_lastTimeReleased

Retourne :

un entier représentant le temps absolu (nombre de millisecondes) entre la mise sous tension du module et le dernier relâchement observée du bouton à l'entrée (transition du contact de fermé à ouvert)

En cas d'erreur, déclenche une exception ou retourne `YAnButton.LASTTIMERELASED_INVALID`.

anbutton→get_logicalName()**YAnButton****anbutton→logicalName()**

Retourne le nom logique de l'entrée analogique.

js	function get_logicalName ()
cpp	string get_logicalName ()
m	-(NSString*) logicalName
pas	string get_logicalName (): string
vb	function get_logicalName () As String
cs	string get_logicalName ()
java	String get_logicalName ()
uwp	async Task<string> get_logicalName ()
py	get_logicalName ()
php	function get_logicalName ()
ts	async get_logicalName (): Promise<string>
es	async get_logicalName ()
dnp	string get_logicalName ()
cp	string get_logicalName ()
cmd	YAnButton target get_logicalName

Retourne :

une chaîne de caractères représentant le nom logique de l'entrée analogique.

En cas d'erreur, déclenche une exception ou retourne `YAnButton.LOGICALNAME_INVALID`.

anbutton→get_module()**YAnButton****anbutton→module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

js	function get_module ()
cpp	<code>YModule *</code> get_module ()
m	-(<code>YModule*</code>) module
pas	<code>TYModule</code> get_module (): <code>TYModule</code>
vb	function get_module () As <code>YModule</code>
cs	<code>YModule</code> get_module ()
java	<code>YModule</code> get_module ()
py	get_module ()
php	function get_module ()
ts	async get_module (): <code>Promise<YModule></code>
es	async get_module ()
dnp	<code>YModuleProxy</code> get_module ()
cp	<code>YModuleProxy *</code> get_module ()

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

anbutton→**get_module_async()****YAnButton****anbutton**→**module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

anbutton→get_pulseCounter()**YAnButton****anbutton→pulseCounter()**

Retourne la valeur du compteur d'impulsions.

js	function get_pulseCounter ()
cpp	s64 get_pulseCounter ()
m	-(s64) pulseCounter
pas	int64 get_pulseCounter (): int64
vb	function get_pulseCounter () As Long
cs	long get_pulseCounter ()
java	long get_pulseCounter ()
uwp	async Task<long> get_pulseCounter ()
py	get_pulseCounter ()
php	function get_pulseCounter ()
ts	async get_pulseCounter (): Promise<number>
es	async get_pulseCounter ()
dnp	long get_pulseCounter ()
cp	s64 get_pulseCounter ()
cmd	YAnButton target get_pulseCounter

La valeur est codée sur 32 bits. En cas de dépassement de capacité ($\geq 2^{32}$), le compteur repart à zéro. Le compteur peut être réinitialisé en appelant la méthode `resetCounter()`.

Retourne :

un entier représentant la valeur du compteur d'impulsions

En cas d'erreur, déclenche une exception ou retourne `YAnButton.PULSECOUNTER_INVALID`.

anbutton→get_pulseTimer()**YAnButton****anbutton→pulseTimer()**

Retourne le timer du compteur d'impulsions (ms).

js	function get_pulseTimer ()
cpp	s64 get_pulseTimer ()
m	-(s64) pulseTimer
pas	int64 get_pulseTimer (): int64
vb	function get_pulseTimer () As Long
cs	long get_pulseTimer ()
java	long get_pulseTimer ()
uwp	async Task<long> get_pulseTimer ()
py	get_pulseTimer ()
php	function get_pulseTimer ()
ts	async get_pulseTimer (): Promise<number>
es	async get_pulseTimer ()
dnp	long get_pulseTimer ()
cp	s64 get_pulseTimer ()
cmd	YAnButton target get_pulseTimer

Retourne :

un entier représentant le timer du compteur d'impulsions (ms)

En cas d'erreur, déclenche une exception ou retourne `YAnButton.PULSETIMER_INVALID`.

anbutton→get_rawValue()**YAnButton****anbutton→rawValue()**

Retourne la valeur mesurée de l'entrée telle-quelle (entre 0 et 4095 inclus).

js	function get_rawValue ()
cpp	int get_rawValue ()
m	-(int) rawValue
pas	LongInt get_rawValue (): LongInt
vb	function get_rawValue () As Integer
cs	int get_rawValue ()
java	int get_rawValue ()
uwp	async Task<int> get_rawValue ()
py	get_rawValue ()
php	function get_rawValue ()
ts	async get_rawValue (): Promise<number>
es	async get_rawValue ()
dnp	int get_rawValue ()
cp	int get_rawValue ()
cmd	YAnButton target get_rawValue

Retourne :

un entier représentant la valeur mesurée de l'entrée telle-quelle (entre 0 et 4095 inclus)

En cas d'erreur, déclenche une exception ou retourne `YAnButton.RAWVALUE_INVALID`.

anbutton→get_sensitivity()**YAnButton****anbutton→sensitivity()**

Retourne la sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks.

js	function get_sensitivity ()
cpp	int get_sensitivity ()
m	-(int) sensitivity
pas	LongInt get_sensitivity (): LongInt
vb	function get_sensitivity () As Integer
cs	int get_sensitivity ()
java	int get_sensitivity ()
uwp	async Task<int> get_sensitivity ()
py	get_sensitivity ()
php	function get_sensitivity ()
ts	async get_sensitivity (): Promise<number>
es	async get_sensitivity ()
dnp	int get_sensitivity ()
cp	int get_sensitivity ()
cmd	YAnButton target get_sensitivity

Retourne :

un entier représentant la sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks

En cas d'erreur, déclenche une exception ou retourne `YAnButton.SENSITIVITY_INVALID`.

anbutton→get_serialNumber()**YAnButton****anbutton→serialNumber()**

Retourne le numéro de série du module, préprogrammé en usine.

js	function get_serialNumber ()
cpp	string get_serialNumber ()
m	-(NSString*) serialNumber
pas	string get_serialNumber (): string
vb	function get_serialNumber () As String
cs	string get_serialNumber ()
java	String get_serialNumber ()
uwp	async Task<string> get_serialNumber ()
py	get_serialNumber ()
php	function get_serialNumber ()
ts	async get_serialNumber (): Promise<string>
es	async get_serialNumber ()
dnp	string get_serialNumber ()
cp	string get_serialNumber ()
cmd	YAnButton target get_serialNumber

Retourne :

: une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine.

En cas d'erreur, déclenche une exception ou retourne YFunction.SERIALNUMBER_INVALID.

anbutton→get_userdata()**YAnButton****anbutton→userdata()**

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userdata`.

js	<code>function get_userdata()</code>
cpp	<code>void * get_userdata()</code>
m	<code>-(id) userData</code>
pas	<code>Tobject get_userdata(): Tobject</code>
vb	<code>function get_userdata() As Object</code>
cs	<code>object get_userdata()</code>
java	<code>Object get_userdata()</code>
py	<code>get_userdata()</code>
php	<code>function get_userdata()</code>
ts	<code>async get_userdata(): Promise<object null></code>
es	<code>async get_userdata()</code>

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

anbutton→isOnline()**YAnButton**

Vérifie si le module hébergeant l'entrée analogique est joignable, sans déclencher d'erreur.

js	function isOnline ()
cpp	bool isOnline ()
m	-(BOOL) isOnline
pas	boolean isOnline (): boolean
vb	function isOnline () As Boolean
cs	bool isOnline ()
java	boolean isOnline ()
py	isOnline ()
php	function isOnline ()
ts	async isOnline (): Promise<boolean>
es	async isOnline ()
dnp	bool isOnline ()
cp	bool isOnline ()

Si les valeurs des attributs en cache de l'entrée analogique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si l'entrée analogique est joignable, `false` sinon

anbutton→isOnline_async()**YAnButton**

Vérifie si le module hébergeant l'entrée analogique est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache de l'entrée analogique sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

anbutton→isReadOnly()**YAnButton**

Test si la fonction est en lecture seule.

cpp	bool isReadOnly ()
m	-(bool) isReadOnly
pas	boolean isReadOnly (): boolean
vb	function isReadOnly () As Boolean
cs	bool isReadOnly ()
java	boolean isReadOnly ()
uwp	async Task<bool> isReadOnly ()
py	isReadOnly ()
php	function isReadOnly ()
ts	async isReadOnly (): Promise<boolean>
es	async isReadOnly ()
dnp	bool isReadOnly ()
cp	bool isReadOnly ()
cmd	YAnButton target isReadOnly

Retourne vrais si la fonction est protégé en ecriture ou que la fontion n'est pas disponible.

Retourne :

`true` si la fonction est protégé en ecriture ou que la fontion n'est pas disponible

anbutton→load()**YAnButton**

Met en cache les valeurs courantes de l'entrée analogique, avec une durée de validité spécifiée.

js	function load (msValidity)
c++	YRETCODE load (int msValidity)
m	-(YRETCODE) load : (u64) msValidity
pas	YRETCODE load (msValidity : u64): YRETCODE
vb	function load (ByVal msValidity As Long) As YRETCODE
cs	YRETCODE load (ulong msValidity)
java	int load (long msValidity)
py	load (msValidity)
php	function load (\$msValidity)
ts	async load (msValidity : number): Promise<number>
es	async load (msValidity)

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

anbutton→loadAttribute()**YAnButton**

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

js	function loadAttribute(attrName)
cpp	string loadAttribute(string attrName)
m	-(NSString*) loadAttribute : (NSString*) attrName
pas	string loadAttribute(attrName: string): string
vb	function loadAttribute(ByVal attrName As String) As String
cs	string loadAttribute(string attrName)
java	String loadAttribute(String attrName)
uwp	async Task<string> loadAttribute(string attrName)
py	loadAttribute(attrName)
php	function loadAttribute(\$attrName)
ts	async loadAttribute(attrName: string): Promise<string>
es	async loadAttribute(attrName)
dnf	string loadAttribute(string attrName)
cp	string loadAttribute(string attrName)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

anbutton→load_async()**YAnButton**

Met en cache les valeurs courantes de l'entrée analogique, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI . SUCCESS`)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

anbutton→muteValueCallbacks()**YAnButton**

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

js	function muteValueCallbacks ()
cpp	int muteValueCallbacks ()
m	-(int) muteValueCallbacks
pas	LongInt muteValueCallbacks (): LongInt
vb	function muteValueCallbacks () As Integer
cs	int muteValueCallbacks ()
java	int muteValueCallbacks ()
uwp	async Task<int> muteValueCallbacks ()
py	muteValueCallbacks ()
php	function muteValueCallbacks ()
ts	async muteValueCallbacks (): Promise<number>
es	async muteValueCallbacks ()
dnp	int muteValueCallbacks ()
cp	int muteValueCallbacks ()
cmd	YAnButton target muteValueCallbacks

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclenchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

anbutton→nextAnButton()**YAnButton**

Continue l'énumération des entrées analogiques commencée à l'aide de `yFirstAnButton()`. Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les entrées analogiques sont retournés.

js	function nextAnButton ()
cpp	YAnButton * nextAnButton ()
m	-(nullable YAnButton*) nextAnButton
pas	TYAnButton nextAnButton (): TYAnButton
vb	function nextAnButton () As YAnButton
cs	YAnButton nextAnButton ()
java	YAnButton nextAnButton ()
uwp	YAnButton nextAnButton ()
py	nextAnButton ()
php	function nextAnButton ()
ts	nextAnButton (): YAnButton null
es	nextAnButton ()

Si vous souhaitez retrouver une entrée analogique spécifique, utilisez `AnButton.findAnButton()` avec un `hardwareID` ou un nom logique.

Retourne :

un pointeur sur un objet `YAnButton` accessible en ligne, ou `null` lorsque l'énumération est terminée.

anbutton→registerValueCallback()**YAnButton**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

js	function registerValueCallback (callback)
cpp	int registerValueCallback (YAnButtonValueCallback callback)
m	-(int) registerValueCallback : (YAnButtonValueCallback _Nullable) callback
pas	LongInt registerValueCallback (callback : TYAnButtonValueCallback): LongInt
vb	function registerValueCallback (ByVal callback As YAnButtonValueCallback) As Integer
cs	int registerValueCallback (ValueCallback callback)
java	int registerValueCallback (UpdateCallback callback)
uwp	async Task<int> registerValueCallback (ValueCallback callback)
py	registerValueCallback (callback)
php	function registerValueCallback (\$callback)
ts	async registerValueCallback (callback : YAnButtonValueCallback null): Promise<number>
es	async registerValueCallback (callback)

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

anbutton→resetCounter()**YAnButton**

Réinitialise le compteur d'impulsions et son timer.

js	function resetCounter ()
cpp	int resetCounter ()
m	-(int) resetCounter
pas	LongInt resetCounter (): LongInt
vb	function resetCounter () As Integer
cs	int resetCounter ()
java	int resetCounter ()
uwp	async Task<int> resetCounter ()
py	resetCounter ()
php	function resetCounter ()
ts	async resetCounter (): Promise<number>
es	async resetCounter ()
dnp	int resetCounter ()
cp	int resetCounter ()
cmd	YAnButton target resetCounter

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

anbutton→set_analogCalibration()**YAnButton****anbutton→setAnalogCalibration()**

Enclenche ou déclenche le procédure de calibration.

js	function set_analogCalibration (newval)
cpp	int set_analogCalibration (Y_ANALOGCALIBRATION_enum newval)
m	-(int) setAnalogCalibration : (Y_ANALOGCALIBRATION_enum) newval
pas	integer set_analogCalibration (newval : Integer): integer
vb	function set_analogCalibration (ByVal newval As Integer) As Integer
cs	int set_analogCalibration (int newval)
java	int set_analogCalibration (int newval)
uwp	async Task<int> set_analogCalibration (int newval)
py	set_analogCalibration (newval)
php	function set_analogCalibration (\$ newval)
ts	async set_analogCalibration (newval : YAnButton_AnalogCalibration): Promise<number>
es	async set_analogCalibration (newval)
dnp	int set_analogCalibration (int newval)
cp	int set_analogCalibration (int newval)
cmd	YAnButton target set_analogCalibration newval

N'oubliez pas d'appeler la méthode `saveToFlash()` du module à la fin de la calibration si le réglage doit être préservé.

Paramètres :

newval soit `YAnButton.ANALOGCALIBRATION_OFF`, soit `YAnButton.ANALOGCALIBRATION_ON`

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

anbutton→set_calibrationMax()**YAnButton****anbutton→setCalibrationMax()**

Modifie la valeur maximale de calibration pour l'entrée (entre 0 et 4095 inclus), sans lancer la calibration automatique.

js	function set_calibrationMax (newval)
cpp	int set_calibrationMax (int newval)
m	-(int) setCalibrationMax : (int) newval
pas	integer set_calibrationMax (newval : LongInt): integer
vb	function set_calibrationMax (ByVal newval As Integer) As Integer
cs	int set_calibrationMax (int newval)
java	int set_calibrationMax (int newval)
uwp	async Task<int> set_calibrationMax (int newval)
py	set_calibrationMax (newval)
php	function set_calibrationMax (\$newval)
ts	async set_calibrationMax (newval : number): Promise<number>
es	async set_calibrationMax (newval)
dnp	int set_calibrationMax (int newval)
cp	int set_calibrationMax (int newval)
cmd	YAnButton target set_calibrationMax newval

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la valeur maximale de calibration pour l'entrée (entre 0 et 4095 inclus), sans lancer la calibration automatique

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

anbutton→set_calibrationMin()**YAnButton****anbutton→setCalibrationMin()**

Modifie la valeur minimale de calibration pour l'entrée (entre 0 et 4095 inclus), sans lancer la calibration automatique.

js	function set_calibrationMin (newval)
cpp	int set_calibrationMin (int newval)
m	-(int) setCalibrationMin : (int) newval
pas	integer set_calibrationMin (newval : LongInt): integer
vb	function set_calibrationMin (ByVal newval As Integer) As Integer
cs	int set_calibrationMin (int newval)
java	int set_calibrationMin (int newval)
uwp	async Task<int> set_calibrationMin (int newval)
py	set_calibrationMin (newval)
php	function set_calibrationMin (\$newval)
ts	async set_calibrationMin (newval : number): Promise<number>
es	async set_calibrationMin (newval)
dnp	int set_calibrationMin (int newval)
cp	int set_calibrationMin (int newval)
cmd	YAnButton target set_calibrationMin newval

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la valeur minimale de calibration pour l'entrée (entre 0 et 4095 inclus), sans lancer la calibration automatique

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

anbutton→set_inputType()**YAnButton****anbutton→setInputType()**

Modifie le type de décodage appliqué à l'entrée (entrée analogique ou entrées binaires multiplexées).

js	function set_inputType (newval)
cpp	int set_inputType (Y_INPUTTYPE_enum newval)
m	-(int) setInputType : (Y_INPUTTYPE_enum) newval
pas	integer set_inputType (newval : Integer): integer
vb	function set_inputType (ByVal newval As Integer) As Integer
cs	int set_inputType (int newval)
java	int set_inputType (int newval)
uwp	async Task<int> set_inputType (int newval)
py	set_inputType (newval)
php	function set_inputType (\$newval)
ts	async set_inputType (newval : YAnButton_InputType): Promise<number>
es	async set_inputType (newval)
dnp	int set_inputType (int newval)
cp	int set_inputType (int newval)
cmd	YAnButton target set_inputType newval

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une valeur parmi `YAnButton.INPUTTYPE_ANALOG_FAST`, `YAnButton.INPUTTYPE_DIGITAL4` et `YAnButton.INPUTTYPE_ANALOG_SMOOTH` représentant le type de décodage appliqué à l'entrée (entrée analogique ou entrées binaires multiplexées)

Retourne :

`YAPI.SUCCESS` si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

anbutton→set_logicalName()**YAnButton****anbutton→setLogicalName()**

Modifie le nom logique de l'entrée analogique.

js	function set_logicalName (newval)
cpp	int set_logicalName (string newval)
m	-(int) setLogicalName : (NSString*) newval
pas	integer set_logicalName (newval : string): integer
vb	function set_logicalName (ByVal newval As String) As Integer
cs	int set_logicalName (string newval)
java	int set_logicalName (String newval)
uwp	async Task<int> set_logicalName (string newval)
py	set_logicalName (newval)
php	function set_logicalName (\$ newval)
ts	async set_logicalName (newval : string): Promise<number>
es	async set_logicalName (newval)
dnp	int set_logicalName (string newval)
cp	int set_logicalName (string newval)
cmd	YAnButton target set_logicalName newval

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique de l'entrée analogique.

Retourne :

YAPI . SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

anbutton→**set_sensitivity()****YAnButton****anbutton**→**setSensitivity()**

Modifie la sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks.

js	function set_sensitivity (newval)
cpp	int set_sensitivity (int newval)
m	-(int) setSensitivity : (int) newval
pas	integer set_sensitivity (newval : LongInt): integer
vb	function set_sensitivity (ByVal newval As Integer) As Integer
cs	int set_sensitivity (int newval)
java	int set_sensitivity (int newval)
uwp	async Task<int> set_sensitivity (int newval)
py	set_sensitivity (newval)
php	function set_sensitivity (\$newval)
ts	async set_sensitivity (newval : number): Promise<number>
es	async set_sensitivity (newval)
dnp	int set_sensitivity (int newval)
cp	int set_sensitivity (int newval)
cmd	YAnButton target set_sensitivity newval

La sensibilité sert à filtrer les variations autour d'une valeur fixe, mais ne prétérte pas la transmission d'événements lorsque la valeur d'entrée évolue constamment dans la même direction. Cas particulier: lorsque la valeur 1000 est utilisée, seuls les valeurs déclenchant une commutation d'état pressé/non-pressé sont transmises. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

anbutton→set_userdata()**YAnButton****anbutton→setUserData()**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get_userdata.

js	function set_userdata (data)
cpp	void set_userdata (void * data)
m	-(void) setUserData : (id) data
pas	set_userdata (data : Tobject)
vb	procedure set_userdata (ByVal data As Object)
cs	void set_userdata (object data)
java	void set_userdata (Object data)
py	set_userdata (data)
php	function set_userdata (\$data)
ts	async set_userdata (data : object null): Promise<void>
es	async set_userdata (data)

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

anbutton→unmuteValueCallbacks()**YAnButton**

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

js	function unmuteValueCallbacks ()
c++	int unmuteValueCallbacks ()
m	-(int) unmuteValueCallbacks
pas	LongInt unmuteValueCallbacks (): LongInt
vb	function unmuteValueCallbacks () As Integer
cs	int unmuteValueCallbacks ()
java	int unmuteValueCallbacks ()
uwp	async Task<int> unmuteValueCallbacks ()
py	unmuteValueCallbacks ()
php	function unmuteValueCallbacks ()
ts	async unmuteValueCallbacks (): Promise<number>
es	async unmuteValueCallbacks ()
dnp	int unmuteValueCallbacks ()
cp	int unmuteValueCallbacks ()
cmd	YAnButton target unmuteValueCallbacks

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

anbutton→wait_async()**YAnButton**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

js	<code>function wait_async(callback, context)</code>
ts	<code>wait_async(callback: Function, context: object)</code>
es	<code>wait_async(callback, context)</code>

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées. La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

Retourne :

rien du tout.

24.6. La classe YFiles

Interface pour interagir avec les systèmes de fichier, disponibles par exemple dans le Yocto-Color-V2, le Yocto-Serial, le YoctoHub-Ethernet et le YoctoHub-Wireless-n

La class YFiles permet d'accéder au système de fichier embarqué sur certains modules Yoctopuce. Le stockage de fichiers permet par exemple de personnaliser un service web (dans le cas d'un module connecté au réseau) ou pour d'ajouter un police de caractères (dans le cas d'un module d'affichage).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<script type='text/javascript' src='yocto_files.js'></script>
cpp	#include "yocto_files.h"
m	#import "yocto_files.h"
pas	uses yocto_files;
vb	yocto_files.vb
cs	yocto_files.cs
java	import com.yoctopuce.YoctoAPI.YFiles;
uwp	import com.yoctopuce.YoctoAPI.YFiles;
py	from yocto_files import *
php	require_once('yocto_files.php');
ts	in HTML: import { YFiles } from '../dist/esm/yocto_files.js'; in Node.js: import { YFiles } from 'yoctolib-cjs/yocto_files.js';
es	in HTML: <script src='../lib/yocto_files.js'></script> in node.js: require('yoctolib-es2017/yocto_files.js');
dnf	import YoctoProxyAPI.YFilesProxy
cp	#include "yocto_files_proxy.h"
vi	YFiles.vi
ml	import YoctoProxyAPI.YFilesProxy

Fonction globales

YFiles.FindFiles(func)

Permet de retrouver un système de fichier d'après un identifiant donné.

YFiles.FindFilesInContext(yctx, func)

Permet de retrouver un système de fichier d'après un identifiant donné dans un Context YAPI.

YFiles.FirstFiles()

Commence l'énumération des systèmes de fichier accessibles par la librairie.

YFiles.FirstFilesInContext(yctx)

Commence l'énumération des systèmes de fichier accessibles par la librairie.

YFiles.GetSimilarFunctions()

Enumère toutes les fonctions de type Files disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

Propriétés des objets YFilesProxy

files→AdvertisedValue *[lecture seule]*

Courte chaîne de caractères représentant l'état courant de la fonction.

files→FilesCount *[lecture seule]*

Nombre de fichiers présents dans le système de fichier.

files→FriendlyName *[lecture seule]*

Identifiant global de la fonction au format NOM_MODULE . NOM_FONCTION.

files→FunctionId [lecture seule]

Identifiant matériel du système de fichier, sans référence au module.

files→HardwareId [lecture seule]

Identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.

files→IsOnline [lecture seule]

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

files→LogicalName [modifiable]

Nom logique de la fonction.

files→SerialNumber [lecture seule]

Numéro de série du module, préprogrammé en usine.

Méthodes des objets YFiles

files→clearCache()

Invalide le cache.

files→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format TYPE (NAME) = SERIAL . FUNCTIONID.

files→download(pathname)

Télécharge le fichier choisi du filesystème et retourne son contenu.

files→download_async(pathname, callback, context)

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

files→fileExist(filename)

Test si un fichier esit dans le système de fichier du module.

files→format_fs()

Rétabli le système de fichier dans on état original, défragmenté.

files→get_advertisedValue()

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

files→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

files→get_errorType()

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

files→get_filesCount()

Retourne le nombre de fichiers présents dans le système de fichier.

files→get_freeSpace()

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

files→get_friendlyName()

Retourne un identifiant global du système de fichier au format NOM_MODULE . NOM_FONCTION.

files→get_functionDescriptor()

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

files→get_functionId()

Retourne l'identifiant matériel du système de fichier, sans référence au module.

files→get_hardwareId()

Retourne l'identifiant matériel unique du système de fichier au format SERIAL . FUNCTIONID.

files→get_list(pattern)

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

files→get_logicalName()

Retourne le nom logique du système de fichier.

files→get_module()

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

files→get_module_async(callback, context)

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

files→get_serialNumber()

Retourne le numéro de série du module, préprogrammé en usine.

files→get_userData()

Retourne le contenu de l'attribut `userData`, précédemment stocké à l'aide de la méthode `set_userData`.

files→isOnline()

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

files→isOnline_async(callback, context)

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

files→isReadOnly()

Test si la fonction est en lecture seule.

files→load(msValidity)

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

files→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.

files→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

files→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

files→nextFiles()

Continue l'énumération des systèmes de fichier commencée à l'aide de `yFirstFiles()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les systèmes de fichier sont retournés.

files→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

files→remove(pathname)

Efface un fichier, spécifié par son path complet, du système de fichier.

files→set_logicalName(newval)

Modifie le nom logique du système de fichier.

files→set_userData(data)

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

files→unmuteValueCallbacks()

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

files→upload(pathname, content)

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

files→wait_async(callback, context)

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

YFiles.FindFiles()**YFiles****YFiles.FindFiles()**

Permet de retrouver un système de fichier d'après un identifiant donné.

js	function yFindFiles (func)
c++	YFiles* FindFiles (string func)
m	+(YFiles*) FindFiles : (NSString*) func
pas	TYFiles yFindFiles (func : string): TYFiles
vb	function FindFiles (ByVal func As String) As YFiles
cs	static YFiles FindFiles (string func)
java	static YFiles FindFiles (String func)
uwp	static YFiles FindFiles (string func)
py	FindFiles (func)
php	function FindFiles (\$func)
ts	static FindFiles (func : string): YFiles
es	static FindFiles (func)
dnp	static YFilesProxy FindFiles (string func)
cp	static YFilesProxy * FindFiles (string func)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le système de fichier soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YFiles.isOnline()` pour tester si le système de fichier est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Si un appel à la méthode `is_online()` de cet objet renvoie FAUX alors que vous êtes sûr que le module correspondant est bien branché, vérifiez que vous n'avez pas oublié d'appeler `registerHub()` à l'initialisation de l'application.

Paramètres :

func une chaîne de caractères qui référence le système de fichier sans ambiguïté, par exemple `YRGBLED2.files`.

Retourne :

un objet de classe `YFiles` qui permet ensuite de contrôler le système de fichier.

YFiles.FindFilesInContext()**YFiles****YFiles.FindFilesInContext()**

Permet de retrouver un système de fichier d'après un identifiant donné dans un Context YAPI.

java	static YFiles FindFilesInContext (YAPIContext yctx , String func)
uwp	static YFiles FindFilesInContext (YAPIContext yctx , string func)
ts	static FindFilesInContext (yctx : YAPIContext, func : string): YFiles
es	static FindFilesInContext (yctx , func)

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

Cette fonction n'exige pas que le système de fichier soit en ligne au moment où elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YFiles.isOnline()` pour tester si le système de fichier est utilisable à un moment donné. En cas d'ambiguïté lorsqu'on fait une recherche par nom logique, aucune erreur ne sera notifiée: la première instance trouvée sera renvoyée. La recherche se fait d'abord par nom matériel, puis par nom logique.

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence le système de fichier sans ambiguïté, par exemple `YRGBLED2.files`.

Retourne :

un objet de classe `YFiles` qui permet ensuite de contrôler le système de fichier.

YFiles.FirstFiles()**YFiles****YFiles.FirstFiles()**

Commence l'énumération des systèmes de fichier accessibles par la librairie.

js	function yFirstFiles ()
cpp	YFiles * FirstFiles ()
m	+(YFiles*) FirstFiles
pas	TYFiles yFirstFiles (): TYFiles
vb	function FirstFiles () As YFiles
cs	static YFiles FirstFiles ()
java	static YFiles FirstFiles ()
uwp	static YFiles FirstFiles ()
py	FirstFiles ()
php	function FirstFiles ()
ts	static FirstFiles (): YFiles null
es	static FirstFiles ()

Utiliser la fonction `YFiles.nextFiles()` pour itérer sur les autres systèmes de fichier.

Retourne :

un pointeur sur un objet `YFiles`, correspondant au premier système de fichier accessible en ligne, ou `null` si il n'y a pas de systèmes de fichier disponibles.

YFiles.FirstFilesInContext()**YFiles****YFiles.FirstFilesInContext()**

Commence l'énumération des systèmes de fichier accessibles par la librairie.

java	static YFiles FirstFilesInContext (YAPIContext yctx)
uwp	static YFiles FirstFilesInContext (YAPIContext yctx)
ts	static FirstFilesInContext (yctx : YAPIContext): YFiles null
es	static FirstFilesInContext (yctx)

Utiliser la fonction `YFiles.nextFiles()` pour itérer sur les autres systèmes de fichier.

Paramètres :

yctx un contexte YAPI.

Retourne :

un pointeur sur un objet `YFiles`, correspondant au premier système de fichier accessible en ligne, ou `null` si il n'y a pas de systèmes de fichier disponibles.

YFiles.GetSimilarFunctions()**YFiles****YFiles.GetSimilarFunctions()**

Enumère toutes les fonctions de type Files disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

`dn` `static new string[] GetSimilarFunctions()`

`cp` `static vector<string> GetSimilarFunctions()`

Chaque chaîne retournée peut être passée en argument à la méthode `YFiles.FindFiles` pour obtenir un objet permettant d'interagir avec le module correspondant.

Retourne :

un tableau de chaînes de caractères, contenant les identifiants matériels de chaque fonction disponible trouvée.

files→AdvertisedValue**YFiles**

Courte chaîne de caractères représentant l'état courant de la fonction.

dnf

 string **AdvertisedValue**

files→FilesCount**YFiles**

Nombre de fichiers présents dans le système de fichier.

dnp

int FilesCount

files→FriendlyName**YFiles**

Identifiant global de la fonction au format `NOM_MODULE . NOM_FONCTION`.

`dnf` `string` **FriendlyName**

Le chaîne retournée utilise soit les noms logiques du module et de la fonction si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de la fonction (par exemple: `MyCustomName.relay1`)

files→FunctionId**YFiles**

Identifiant matériel du système de fichier, sans référence au module.

`dnf` `string` **FunctionId**

Par exemple `relay1`.

files→HardwareId**YFiles**

Identifiant matériel unique de la fonction au format `SERIAL.FUNCTIONID`.

`dnf` `string` **HardwareId**

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la fonction (par exemple `RELAYLO1-123456.relay1`).

files→IsOnline**YFiles**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

dnf

bool IsOnline

Si les valeurs des attributs en cache de la fonction sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

files→LogicalName**YFiles**

Nom logique de la fonction.

dnp `string LogicalName`

Modifiable. Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

files→**SerialNumber****YFiles**

Numéro de série du module, préprogrammé en usine.

dnp

 string **SerialNumber**

files→clearCache()

YFiles

Invalide le cache.

js	function clearCache ()
cpp	void clearCache ()
m	-(void) clearCache
pas	clearCache ()
vb	procedure clearCache ()
cs	void clearCache ()
java	void clearCache ()
py	clearCache ()
php	function clearCache ()
ts	async clearCache (): Promise<void>
es	async clearCache ()

Invalide le cache des valeurs courantes du système de fichier. Force le prochain appel à une méthode `get_xxx()` ou `loadxxx()` pour charger les les données depuis le module.

files→describe()**YFiles**

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format `TYPE (NAME) = SERIAL . FUNCTIONID`.

js	function describe ()
cpp	string describe ()
m	-(NSString*) describe
pas	string describe (): string
vb	function describe () As String
cs	string describe ()
java	String describe ()
py	describe ()
php	function describe ()
ts	async describe (): Promise<string>
es	async describe ()

Plus précisément, `TYPE` correspond au type de fonction, `NAME` correspond au nom utilisé lors du premier accès à la fonction, `SERIAL` correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et `FUNCTIONID` correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La méthode va retourner `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` si le module est déjà connecté ou `Relay(BadCustomName.relay1)=unresolved` si le module n'est pas déjà connecté. Cette méthode ne déclenche aucune transaction USB ou TCP et peut donc être utilisé dans un débogueur.

Retourne :

une chaîne de caractères décrivant le système de fichier (ex:
`Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

files→download()

YFiles

Télécharge le fichier choisi du filesystem et retourne son contenu.

js	function download (pathname)
c++	string download (string pathname)
m	-(NSMutableData*) download : (NSString*) pathname
pas	TByteArray download (pathname : string): TByteArray
vb	function download (ByVal pathname As String) As Byte
cs	byte[] download (string pathname)
java	byte[] download (String pathname)
uwp	async Task<byte[]> download (string pathname)
py	download (pathname)
php	function download (\$pathname)
ts	async download (pathname : string): Promise<Uint8Array>
es	async download (pathname)
dnp	byte[] download (string pathname)
cp	string download (string pathname)
cmd	YFiles target download pathname

Paramètres :

pathname nom complet du fichier à charger, y compris le chemin d'accès.

Retourne :

le contenu du fichier chargé sous forme d'objet binaire

En cas d'erreur, déclenche une exception ou retourne un contenu vide.

files→download_async()**YFiles**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

```
js function download_async( pathname, callback, context)
```

Paramètres :

pathname nom complet du fichier à charger, y compris le chemin d'accès.

callback fonction fournie par l'utilisateur, qui sera appelée lorsque la suite du chargement aura été effectué. La fonction callback doit prendre trois arguments: - la variable de contexte à disposition de l'utilisateur - l'objet YFiles dont la méthode download_async a été appelée - le contenu du fichier chargé sous forme d'objet binaire

context variable de contexte à disposition de l'utilisateur

Retourne :

rien.

files→fileExist()

YFiles

Test si un fichier existe dans le système de fichier du module.

js	function fileExist (filename)
c++	bool fileExist (string filename)
m	-(bool) fileExist : (NSString*) filename
pas	boolean fileExist (filename : string): boolean
vb	function fileExist (ByVal filename As String) As Boolean
cs	bool fileExist (string filename)
java	boolean fileExist (String filename)
uwp	async Task<bool> fileExist (string filename)
py	fileExist (filename)
php	function fileExist (\$ filename)
ts	async fileExist (filename : string): Promise<boolean>
es	async fileExist (filename)
dnp	bool fileExist (string filename)
cp	bool fileExist (string filename)
cmd	YFiles target fileExist filename

Paramètres :

filename le nom de fichier.

Retourne :

vrai si le fichier existe, et faux si le fichier n'existe pas.

En cas d'erreur, déclenche une exception.

files→format_fs()**YFiles**

Rétabli le système de fichier dans on état original, défragmenté.

js	function format_fs ()
cpp	int format_fs ()
m	-(int) format_fs
pas	LongInt format_fs (): LongInt
vb	function format_fs () As Integer
cs	int format_fs ()
java	int format_fs ()
uwp	async Task<int> format_fs ()
py	format_fs ()
php	function format_fs ()
ts	async format_fs (): Promise<number>
es	async format_fs ()
dnp	int format_fs ()
cp	int format_fs ()
cmd	YFiles target format_fs

entièrement vide. Tous les fichiers précédemment chargés sont irrémédiablement effacés.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→**get_advertisedValue()****files**→**advertisedValue()**

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

js	function get_advertisedValue ()
cpp	string get_advertisedValue ()
m	-(NSString*) advertisedValue
pas	string get_advertisedValue (): string
vb	function get_advertisedValue () As String
cs	string get_advertisedValue ()
java	String get_advertisedValue ()
uwp	async Task<string> get_advertisedValue ()
py	get_advertisedValue ()
php	function get_advertisedValue ()
ts	async get_advertisedValue (): Promise<string>
es	async get_advertisedValue ()
dnp	string get_advertisedValue ()
cp	string get_advertisedValue ()
cmd	YFiles target get_advertisedValue

Retourne :

une chaîne de caractères représentant la valeur courante du système de fichier (pas plus de 6 caractères).

En cas d'erreur, déclenche une exception ou retourne `YFiles.ADVERTISEDVALUE_INVALID`.

files→get_errorMessage()**YFiles****files→errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

js	function get_errorMessage ()
cpp	string get_errorMessage ()
m	-(NSString*) errorMessage
pas	string get_errorMessage (): string
vb	function get_errorMessage () As String
cs	string get_errorMessage ()
java	String get_errorMessage ()
py	get_errorMessage ()
php	function get_errorMessage ()
ts	get_errorMessage (): string
es	get_errorMessage ()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du système de fichier.

files→get_errorType()**files→errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

js	function get_errorType ()
cpp	YRETCODE get_errorType ()
m	-(YRETCODE) errorType
pas	YRETCODE get_errorType (): YRETCODE
vb	function get_errorType () As YRETCODE
cs	YRETCODE get_errorType ()
java	int get_errorType ()
py	get_errorType ()
php	function get_errorType ()
ts	get_errorType (): number
es	get_errorType ()

Cette méthode est principalement utile lorsque la librairie Yoctopuce est utilisée en désactivant la gestion des exceptions.

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du système de fichier.

files→get_filesCount()**YFiles****files→filesCount()**

Retourne le nombre de fichiers présents dans le système de fichier.

js	function get_filesCount ()
cpp	int get_filesCount ()
m	-(int) filesCount
pas	LongInt get_filesCount (): LongInt
vb	function get_filesCount () As Integer
cs	int get_filesCount ()
java	int get_filesCount ()
uwp	async Task<int> get_filesCount ()
py	get_filesCount ()
php	function get_filesCount ()
ts	async get_filesCount (): Promise<number>
es	async get_filesCount ()
dnp	int get_filesCount ()
cp	int get_filesCount ()
cmd	YFiles target get_filesCount

Retourne :

un entier représentant le nombre de fichiers présents dans le système de fichier

En cas d'erreur, déclenche une exception ou retourne `YFiles.FILES_COUNT_INVALID`.

files→**get_freeSpace()****files**→**freeSpace()**

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

js	function get_freeSpace ()
cpp	int get_freeSpace ()
m	-(int) freeSpace
pas	LongInt get_freeSpace (): LongInt
vb	function get_freeSpace () As Integer
cs	int get_freeSpace ()
java	int get_freeSpace ()
uwp	async Task<int> get_freeSpace ()
py	get_freeSpace ()
php	function get_freeSpace ()
ts	async get_freeSpace (): Promise<number>
es	async get_freeSpace ()
dnp	int get_freeSpace ()
cp	int get_freeSpace ()
cmd	YFiles target get_freeSpace

Retourne :

un entier représentant l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets

En cas d'erreur, déclenche une exception ou retourne `YFiles.FREESPACE_INVALID`.

files→**get_friendlyName()****YFiles****files**→**friendlyName()**

Retourne un identifiant global du système de fichier au format `NOM_MODULE.NOM_FONCTION`.

js	function get_friendlyName ()
cpp	string get_friendlyName ()
m	-(NSString*) friendlyName
cs	string get_friendlyName ()
java	String get_friendlyName ()
py	get_friendlyName ()
php	function get_friendlyName ()
ts	async get_friendlyName (): Promise<string>
es	async get_friendlyName ()
dnp	string get_friendlyName ()
cp	string get_friendlyName ()

Le chaîne retournée utilise soit les noms logiques du module et du système de fichier si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel du système de fichier (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant le système de fichier en utilisant les noms logiques (ex: `MyCustomName.relay1`)

En cas d'erreur, déclenche une exception ou retourne `YFiles.FRIENDLYNAME_INVALID`.

files→get_functionDescriptor()**files→functionDescriptor()**

Retourne un identifiant unique de type YFUN_DESCR correspondant à la fonction.

js	function get_functionDescriptor ()
cpp	YFUN_DESCR get_functionDescriptor ()
m	-(YFUN_DESCR) functionDescriptor
pas	YFUN_DESCR get_functionDescriptor (): YFUN_DESCR
vb	function get_functionDescriptor () As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor ()
java	String get_functionDescriptor ()
py	get_functionDescriptor ()
php	function get_functionDescriptor ()
ts	async get_functionDescriptor (): Promise<string>
es	async get_functionDescriptor ()

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y\$CLASSNAME\$.FUNCTIONDESCRIPTOR_INVALID

files→**get_functionId()****YFiles****files**→**functionId()**

Retourne l'identifiant matériel du système de fichier, sans référence au module.

js	function get_functionId ()
cpp	string get_functionId ()
m	-(NSString*) functionId
vb	function get_functionId () As String
cs	string get_functionId ()
java	String get_functionId ()
py	get_functionId ()
php	function get_functionId ()
ts	async get_functionId (): Promise<string>
es	async get_functionId ()
dnp	string get_functionId ()
cp	string get_functionId ()

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant le système de fichier (ex: `relay1`)

En cas d'erreur, déclenche une exception ou retourne `YFiles.FUNCTIONID_INVALID`.

files→**get_hardwareId()****files**→**hardwareId()**

Retourne l'identifiant matériel unique du système de fichier au format `SERIAL.FUNCTIONID`.

js	function get_hardwareId ()
cpp	string get_hardwareId ()
m	-(NSString*) hardwareId
vb	function get_hardwareId () As String
cs	string get_hardwareId ()
java	String get_hardwareId ()
py	get_hardwareId ()
php	function get_hardwareId ()
ts	async get_hardwareId (): Promise<string>
es	async get_hardwareId ()
dnp	string get_hardwareId ()
cp	string get_hardwareId ()

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel du système de fichier (par exemple `RELAYLO1-123456.relay1`).

Retourne :

une chaîne de caractères identifiant le système de fichier (ex: `RELAYLO1-123456.relay1`)

En cas d'erreur, déclenche une exception ou retourne `YFiles.HARDWAREID_INVALID`.

files→get_list()**YFiles****files→list()**

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

js	function get_list (pattern)
cpp	vector<YFileRecord> get_list (string pattern)
m	-(NSMutableArray*) list : (NSString*) pattern
pas	TYFileRecordArray get_list (pattern : string): TYFileRecordArray
vb	function get_list (ByVal pattern As String) As List
cs	List<YFileRecord> get_list (string pattern)
java	ArrayList<YFileRecord> get_list (String pattern)
uwp	async Task<List<YFileRecord>> get_list (string pattern)
py	get_list (pattern)
php	function get_list (\$ pattern)
ts	async get_list (pattern : string): Promise<YFileRecord[]
es	async get_list (pattern)
dnp	YFileRecordProxy[] get_list (string pattern)
cp	vector<YFileRecordProxy> get_list (string pattern)
cmd	YFiles target get_list pattern

Paramètres :

pattern un filtre optionnel sur les noms de fichiers retournés, pouvant contenir des astérisques et des points d'interrogations comme jokers. Si le pattern fourni est vide, tous les fichiers sont retournés.

Retourne :

une liste d'objets YFileRecord, contenant le nom complet (y compris le chemin d'accès), la taille en octets et le CRC 32-bit du contenu du fichier.

En cas d'erreur, déclenche une exception ou retourne une liste vide.

files→**get_logicalName()****files**→**logicalName()**

Retourne le nom logique du système de fichier.

js	function get_logicalName ()
cpp	string get_logicalName ()
m	-(NSString*) logicalName
pas	string get_logicalName (): string
vb	function get_logicalName () As String
cs	string get_logicalName ()
java	String get_logicalName ()
uwp	async Task<string> get_logicalName ()
py	get_logicalName ()
php	function get_logicalName ()
ts	async get_logicalName (): Promise<string>
es	async get_logicalName ()
dnp	string get_logicalName ()
cp	string get_logicalName ()
cmd	YFiles target get_logicalName

Retourne :

une chaîne de caractères représentant le nom logique du système de fichier.

En cas d'erreur, déclenche une exception ou retourne `YFiles.LOGICALNAME_INVALID`.

files→get_module()**YFiles****files→module()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

js	function get_module ()
cpp	<code>YModule *</code> get_module ()
m	-(<code>YModule*</code>) module
pas	<code>TYModule</code> get_module (): <code>TYModule</code>
vb	function get_module () As <code>YModule</code>
cs	<code>YModule</code> get_module ()
java	<code>YModule</code> get_module ()
py	get_module ()
php	function get_module ()
ts	async get_module (): <code>Promise<YModule></code>
es	async get_module ()
dnp	<code>YModuleProxy</code> get_module ()
cp	<code>YModuleProxy *</code> get_module ()

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Retourne :

une instance de `YModule`

files→**get_module_async()****YFiles****files**→**module_async()**

Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

```
js function get_module_async( callback, context)
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de `YModule` retournée ne sera pas joignable.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

files→get_serialNumber()**YFiles****files→serialNumber()**

Retourne le numéro de série du module, préprogrammé en usine.

js	function get_serialNumber ()
cpp	string get_serialNumber ()
m	-(NSString*) serialNumber
pas	string get_serialNumber (): string
vb	function get_serialNumber () As String
cs	string get_serialNumber ()
java	String get_serialNumber ()
uwp	async Task<string> get_serialNumber ()
py	get_serialNumber ()
php	function get_serialNumber ()
ts	async get_serialNumber (): Promise<string>
es	async get_serialNumber ()
dnp	string get_serialNumber ()
cp	string get_serialNumber ()
cmd	YFiles target get_serialNumber

Retourne :

: une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine.

En cas d'erreur, déclenche une exception ou retourne YFunction.SERIALNUMBER_INVALID.

files→get_userdata()**files→userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set_userdata.

js	function get_userdata ()
cpp	void * get_userdata ()
m	-(id) userData
pas	Tobject get_userdata (): Tobject
vb	function get_userdata () As Object
cs	object get_userdata ()
java	Object get_userdata ()
py	get_userdata ()
php	function get_userdata ()
ts	async get_userdata (): Promise<object null>
es	async get_userdata ()

Cet attribut n'est pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Retourne :

l'objet stocké précédemment par l'appelant.

files→isOnline()**YFiles**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

js	function isOnline ()
cpp	bool isOnline ()
m	-(BOOL) isOnline
pas	boolean isOnline (): boolean
vb	function isOnline () As Boolean
cs	bool isOnline ()
java	boolean isOnline ()
py	isOnline ()
php	function isOnline ()
ts	async isOnline (): Promise<boolean>
es	async isOnline ()
dnp	bool isOnline ()
cp	bool isOnline ()

Si les valeurs des attributs en cache du système de fichier sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si le système de fichier est joignable, `false` sinon

files→isOnline_async()**YFiles**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

```
js function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache du système de fichier sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

files→isReadOnly()**YFiles**

Test si la fonction est en lecture seule.

cpp	bool isReadOnly ()
m	-(bool) isReadOnly
pas	boolean isReadOnly (): boolean
vb	function isReadOnly () As Boolean
cs	bool isReadOnly ()
java	boolean isReadOnly ()
uwp	async Task<bool> isReadOnly ()
py	isReadOnly ()
php	function isReadOnly ()
ts	async isReadOnly (): Promise<boolean>
es	async isReadOnly ()
dnp	bool isReadOnly ()
cp	bool isReadOnly ()
cmd	YFiles target isReadOnly

Retourne vrai si la fonction est protégé en écriture ou que la fonction n'est pas disponible.

Retourne :

`true` si la fonction est protégé en écriture ou que la fonction n'est pas disponible

files→load()

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

js	function load (msValidity)
c++	YRETCODE load (int msValidity)
m	-(YRETCODE) load : (u64) msValidity
pas	YRETCODE load (msValidity : u64): YRETCODE
vb	function load (ByVal msValidity As Long) As YRETCODE
cs	YRETCODE load (ulong msValidity)
java	int load (long msValidity)
py	load (msValidity)
php	function load (\$msValidity)
ts	async load (msValidity : number): Promise<number>
es	async load (msValidity)

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

msValidity un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→loadAttribute()

YFiles

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

js	function loadAttribute(attrName)
cpp	string loadAttribute(string attrName)
m	-(NSString*) loadAttribute : (NSString*) attrName
pas	string loadAttribute(attrName: string): string
vb	function loadAttribute(ByVal attrName As String) As String
cs	string loadAttribute(string attrName)
java	String loadAttribute(String attrName)
uwp	async Task<string> loadAttribute(string attrName)
py	loadAttribute(attrName)
php	function loadAttribute(\$attrName)
ts	async loadAttribute(attrName: string): Promise<string>
es	async loadAttribute(attrName)
dnf	string loadAttribute(string attrName)
cp	string loadAttribute(string attrName)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

files→load_async()

YFiles

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionnellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

- msValidity** un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
- callback** fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI . SUCCESS)
- context** contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

files→muteValueCallbacks()**YFiles**

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

js	function muteValueCallbacks ()
cpp	int muteValueCallbacks ()
m	-(int) muteValueCallbacks
pas	LongInt muteValueCallbacks (): LongInt
vb	function muteValueCallbacks () As Integer
cs	int muteValueCallbacks ()
java	int muteValueCallbacks ()
uwp	async Task<int> muteValueCallbacks ()
py	muteValueCallbacks ()
php	function muteValueCallbacks ()
ts	async muteValueCallbacks (): Promise<number>
es	async muteValueCallbacks ()
dnp	int muteValueCallbacks ()
cp	int muteValueCallbacks ()
cmd	YFiles target muteValueCallbacks

Vous pouvez utiliser cette fonction pour économiser la bande passante et le CPU sur les machines de faible puissance, ou pour éviter le déclenchement de callbacks HTTP. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→**nextFiles()****YFiles**

Continue l'énumération des systèmes de fichier commencée à l'aide de `yFirstFiles()`. Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les systèmes de fichier sont retournés.

js	function nextFiles ()
cpp	YFiles * nextFiles ()
m	-(nullable YFiles*) nextFiles
pas	TYFiles nextFiles (): TYFiles
vb	function nextFiles () As YFiles
cs	YFiles nextFiles ()
java	YFiles nextFiles ()
uwp	YFiles nextFiles ()
py	nextFiles ()
php	function nextFiles ()
ts	nextFiles (): YFiles null
es	nextFiles ()

Si vous souhaitez retrouver un système de fichier spécifique, utilisez `Files.findFiles()` avec un `hardwareID` ou un nom logique.

Retourne :

un pointeur sur un objet `YFiles` accessible en ligne, ou `null` lorsque l'énumération est terminée.

files→registerValueCallback()

YFiles

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

js	function registerValueCallback (callback)
cpp	int registerValueCallback (YFilesValueCallback callback)
m	-(int) registerValueCallback : (YFilesValueCallback _Nullable) callback
pas	LongInt registerValueCallback (callback : TYFilesValueCallback): LongInt
vb	function registerValueCallback (ByVal callback As YFilesValueCallback) As Integer
cs	int registerValueCallback (ValueCallback callback)
java	int registerValueCallback (UpdateCallback callback)
uwp	async Task<int> registerValueCallback (ValueCallback callback)
py	registerValueCallback (callback)
php	function registerValueCallback (\$callback)
ts	async registerValueCallback (callback : YFilesValueCallback null): Promise<number>
es	async registerValueCallback (callback)

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

files→remove()

YFiles

Efface un fichier, spécifié par son path complet, du système de fichier.

js	function remove (pathname)
c++	int remove (string pathname)
m	-(int) remove : (NSString*) pathname
pas	LongInt remove (pathname : string): LongInt
vb	function remove (ByVal pathname As String) As Integer
cs	int remove (string pathname)
java	int remove (String pathname)
uwp	async Task<int> remove (string pathname)
py	remove (pathname)
php	function remove (\$pathname)
ts	async remove (pathname : string): Promise<number>
es	async remove (pathname)
dnp	int remove (string pathname)
cp	int remove (string pathname)
cmd	YFiles target remove pathname

A cause de la fragmentation, l'effacement d'un fichier ne libère pas toujours la totalité de l'espace qu'il occupe. Par contre, la ré-écriture d'un fichier du même nom récupérera dans tout les cas l'espace qui n'aurait éventuellement pas été libéré. Pour s'assurer de libérer la totalité de l'espace du système de fichier, utilisez la fonction `format_fs`.

Paramètres :

pathname nom complet du fichier, y compris le chemin d'accès.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→set_logicalName()**YFiles****files→setLogicalName()**

Modifie le nom logique du système de fichier.

js	function set_logicalName (newval)
cpp	int set_logicalName (string newval)
m	-(int) setLogicalName : (NSString*) newval
pas	integer set_logicalName (newval : string): integer
vb	function set_logicalName (ByVal newval As String) As Integer
cs	int set_logicalName (string newval)
java	int set_logicalName (String newval)
uwp	async Task<int> set_logicalName (string newval)
py	set_logicalName (newval)
php	function set_logicalName (\$ newval)
ts	async set_logicalName (newval : string): Promise<number>
es	async set_logicalName (newval)
dnp	int set_logicalName (string newval)
cp	int set_logicalName (string newval)
cmd	YFiles target set_logicalName newval

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du système de fichier.

Retourne :

YAPI . SUCCESS si l'appel se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→**set_userData()****files**→**setUserData()**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

js	function set_userData (data)
cpp	void set_userData (void * data)
m	-(void) setUserData : (id) data
pas	set_userData (data : Tobject)
vb	procedure set_userData (ByVal data As Object)
cs	void set_userData (object data)
java	void set_userData (Object data)
py	set_userData (data)
php	function set_userData (\$data)
ts	async set_userData (data : object null): Promise<void>
es	async set_userData (data)

Cet attribut n'es pas utilisé directement par l'API. Il est à la disposition de l'appelant pour stocker un contexte.

Paramètres :

data objet quelconque à mémoriser

files→unmuteValueCallbacks()**YFiles**

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

js	function unmuteValueCallbacks ()
cpp	int unmuteValueCallbacks ()
m	-(int) unmuteValueCallbacks
pas	LongInt unmuteValueCallbacks (): LongInt
vb	function unmuteValueCallbacks () As Integer
cs	int unmuteValueCallbacks ()
java	int unmuteValueCallbacks ()
uwp	async Task<int> unmuteValueCallbacks ()
py	unmuteValueCallbacks ()
php	function unmuteValueCallbacks ()
ts	async unmuteValueCallbacks (): Promise<number>
es	async unmuteValueCallbacks ()
dnp	int unmuteValueCallbacks ()
cp	int unmuteValueCallbacks ()
cmd	YFiles target unmuteValueCallbacks

Cette fonction annule un précédent appel à `muteValueCallbacks()`. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→upload()

YFiles

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

js	function upload (pathname , content)
c++	int upload (string pathname , string content)
m	-(int) upload : (NSString*) pathname : (NSData*) content
pas	LongInt upload (pathname : string, content : TByteArray): LongInt
vb	procedure upload (ByVal pathname As String, ByVal content As Byte())
cs	int upload (string pathname , byte[] content)
java	int upload (String pathname , byte[] content)
uwp	async Task<int> upload (string pathname , byte[] content)
py	upload (pathname , content)
php	function upload (\$pathname, \$content)
ts	async upload (pathname : string, content : Uint8Array): Promise<number>
es	async upload (pathname , content)
dnp	int upload (string pathname , byte[] content)
cp	int upload (string pathname , string content)
cmd	YFiles target upload pathname content

Si un fichier existe déjà pour le même chemin d'accès, son contenu est remplacé.

Paramètres :

- pathname** nom complet du fichier, y compris le chemin d'accès.
- content** contenu du fichier à télécharger

Retourne :

YAPI . SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

files→wait_async()**YFiles**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

js	function wait_async (callback , context)
ts	wait_async (callback : Function, context : object)
es	wait_async (callback , context)

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées. La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

context contexte fourni par l'appelant, et qui sera passé tel-qu'el à la fonction de callback

Retourne :

rien du tout.

25. Problèmes courants

25.1. Par où commencer ?

Si c'est la première fois que vous utilisez un module Yoctopuce et ne savez pas trop par où commencer, allez donc jeter un coup d'œil sur le blog de Yoctopuce. Il y a une section dédiée aux débutants ¹.

25.2. Linux et USB

Pour fonctionner correctement sous Linux la librairie a besoin d'avoir accès en écriture à tous les périphériques USB Yoctopuce. Or, par défaut, sous Linux les droits d'accès des utilisateurs non-root à USB sont limités à la lecture. Afin d'éviter de devoir lancer les exécutable en tant que root, il faut créer une nouvelle règle *udev* pour autoriser un ou plusieurs utilisateurs à accéder en écriture aux périphériques Yoctopuce.

Pour ajouter une règle *udev* à votre installation, il faut ajouter un fichier avec un nom au format "`##-nomArbitraire.rules`" dans le répertoire `/etc/udev/rules.d`. Lors du démarrage du système, *udev* va lire tous les fichiers avec l'extension `.rules` de ce répertoire en respectant l'ordre alphabétique (par exemple, le fichier `"51-custom.rules"` sera interprété APRES le fichier `"50-udev-default.rules"`).

Le fichier `"50-udev-default"` contient les règles *udev* par défaut du système. Pour modifier le comportement par défaut du système, il faut donc créer un fichier qui commence par un nombre plus grand que 50, qui définira un comportement plus spécifique que le défaut du système. Notez que pour ajouter une règle vous aurez besoin d'avoir un accès root sur le système.

Dans le répertoire `udev_conf` de l'archive du *VirtualHub*² pour Linux, vous trouverez deux exemples de règles qui vous éviteront de devoir partir de rien.

Exemple 1: 51-yoctopuce.rules

Cette règle va autoriser tous les utilisateurs à accéder en lecture et en écriture aux périphériques Yoctopuce USB. Les droits d'accès pour tous les autres périphériques ne seront pas modifiés. Si ce scénario vous convient il suffit de copier le fichier `"51-yoctopuce_all.rules"` dans le répertoire `/etc/udev/rules.d` et de redémarrer votre système.

¹ voir: http://www.yoctopuce.com/FR/blog_by_categories/pour-les-debutants

² <http://www.yoctopuce.com/EN/virtualhub.php>

```
# udev rules to allow write access to all users
# for Yoctopuce USB devices
SUBSYSTEM=="usb", ATTR{idVendor}=="24e0", MODE="0666"
```

Exemple 2: 51-yoctopuce_group.rules

Cette règle va autoriser le groupe "yoctogroup" à accéder en lecture et écriture aux périphériques Yoctopuce USB. Les droits d'accès pour tous les autres périphériques ne seront pas modifiés. Si ce scénario vous convient il suffit de copier le fichier "51-yoctopuce_group.rules" dans le répertoire "/etc/udev/rules.d" et de redémarrer votre système.

```
# udev rules to allow write access to all users of "yoctogroup"
# for Yoctopuce USB devices
SUBSYSTEM=="usb", ATTR{idVendor}=="24e0", MODE="0664", GROUP="yoctogroup"
```

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-MaxiDisplay-G 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-MaxiDisplay-G 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. RegisterHub d'un VirtualHub déconnecte le précédent

Si lorsque vous faire un `YAPI.RegisterHub` d'un VirtualHub 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.

25.9. Commandes ignorées

Si vous avez l'impression que des commandes envoyées à un module Yoctopuce sont ignorées, typiquement lorsque vous avez écrit un programme qui sert à configurer ce modules Yoctopuce et qui envoie donc beaucoup de commandes, vérifiez que vous avez bien mis un `YAPI.FreeAPI()` à la fin du programme. Les commandes sont envoyées aux modules de manière asynchrone grâce à un processus qui tourne en arrière plan. Lorsque le programme se termine, ce processus est tué, même s'il n'a pas eu le temps de tout envoyer. En revanche `API.FreeAPI()` attend que la file d'attente des commandes à envoyer soit vide avant de libérer les ressources utilisées par l'API et rendre la main.

25.10. Module endommagé

Yoctopuce s'efforce de réduire la production de déchets électroniques. Si vous avez l'impression que votre Yocto-MaxiDisplay-G ne fonctionne plus, commencez par contacter le support Yoctopuce par e-mail pour poser un diagnostic. Même si c'est suite à une mauvaise manipulation que le module a été endommagé, il se peut que Yoctopuce puisse le réparer, et ainsi éviter de créer un déchet électronique.



Déchets d'équipements électriques et électroniques (DEEE) Si voulez vraiment vous débarrasser de votre Yocto-MaxiDisplay-G, 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>

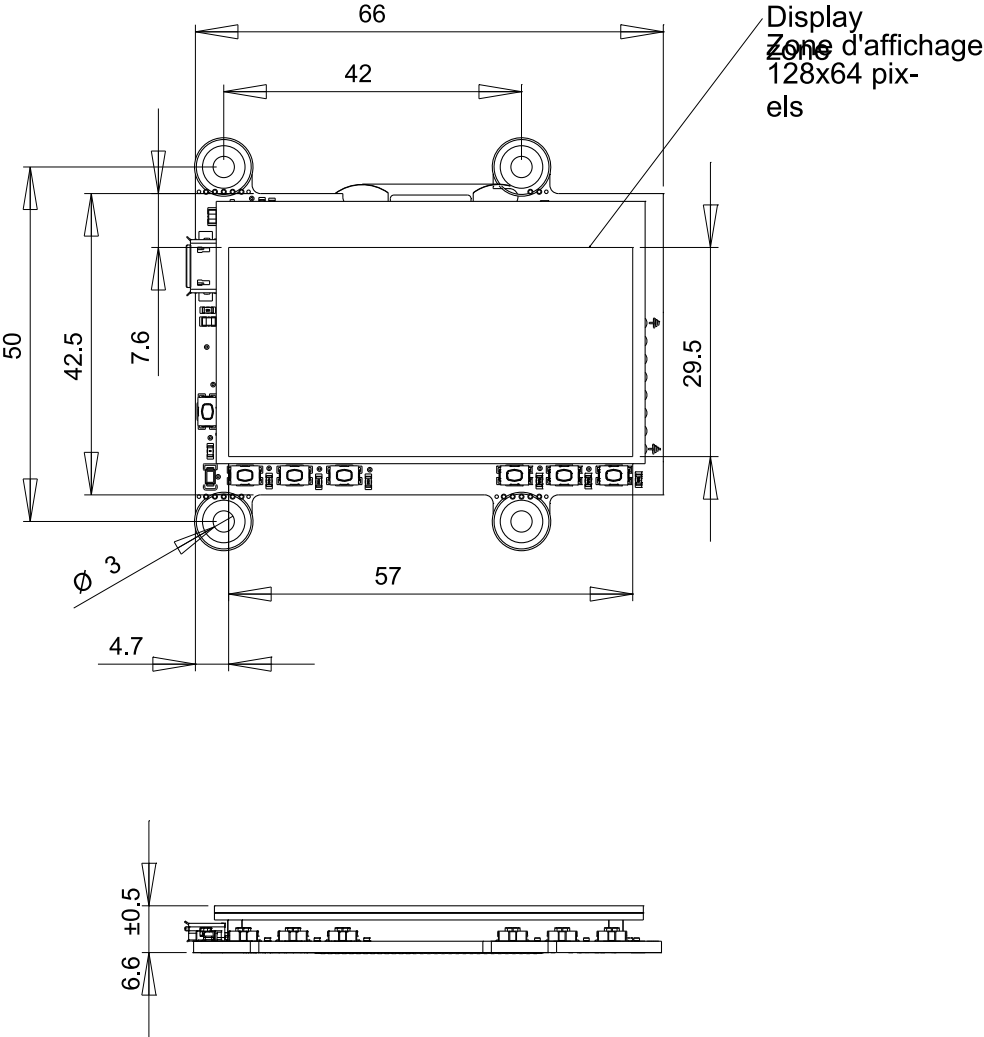
26. Caractéristiques

Vous trouverez résumées ci-dessous les principales caractéristiques techniques de votre module Yocto-MaxiDisplay-G

Identifiant produit	YD128G64
Révision matérielle [†]	
Zone affichable	57x29.5 mm
Connecteur USB	micro-B
Largeur	58 mm
Longueur	66 mm
Poids	23 g
Resolution	128x64 px
Temp. de fonctionnement normale	5...40 °C
Temp. de fonctionnement étendue [‡]	-25...70 °C
Conformité RoHS	RoHS III (2011/65/UE+2015/863)
USB Vendor ID	0x24E0
USB Device ID	0x004A
Boîtier recommandé	YoctoBox-MaxiDisplay
Code tarifaire harmonisé	9032.9000
Fabriqué en	Suisse

[†] Ces spécifications correspondent à la révision matérielle actuelle du produit. Les spécifications des versions antérieures peuvent être inférieures.

[‡] La plage de température étendue est définie d'après les spécifications des composants et testée sur une durée limitée (1h). En cas d'utilisation prolongée hors de la plage de température standard, il est recommandé procéder à des tests extensifs avant la mise en production.



All dimensions are in mm
Toutes les dimensions sont en mm

Yocto-MaxiDisplay

A4
Scale
1:1
Echelle