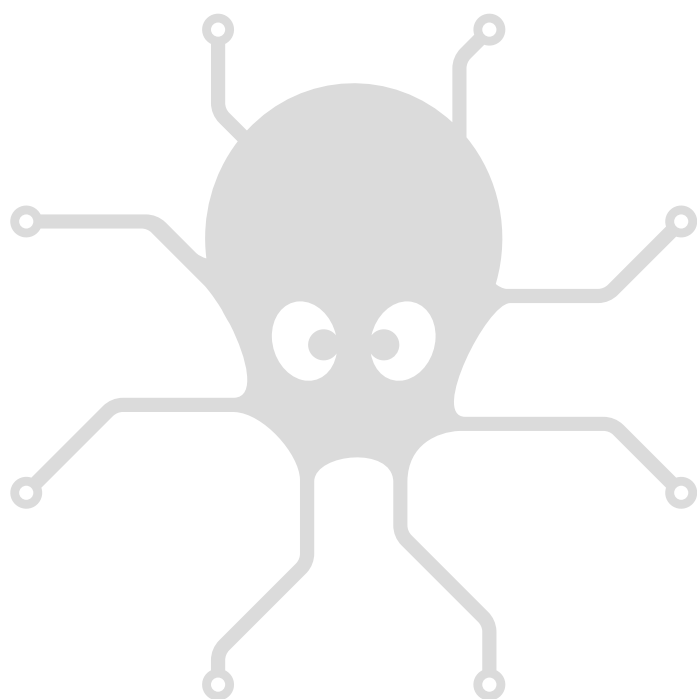




Yocto-Color, Mode d'emploi

Table des matières

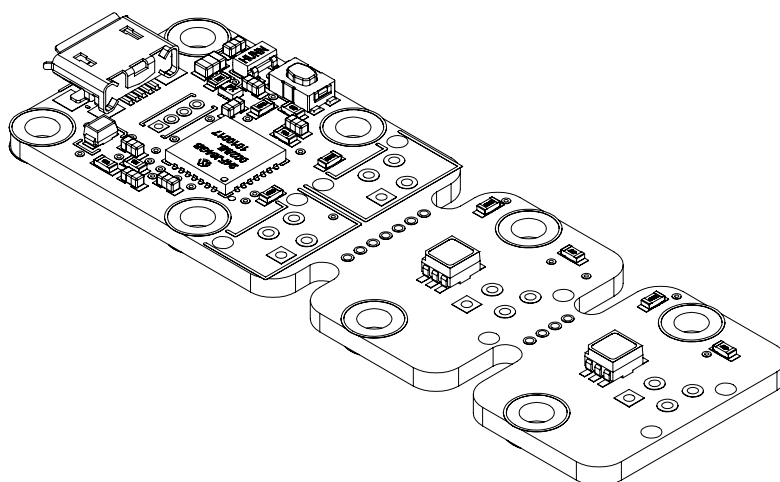
1. Introduction	1
1.1. Prérequis	2
1.2. Accessoires optionnels	3
2. Présentation	5
2.1. Les éléments communs	5
2.2. Les éléments spécifiques	6
3. Premiers pas	7
3.1. Localisation	7
3.2. Test du module	7
3.3. Configuration	8
4. Montage et connectique	11
4.1. Fixation	11
4.2. Déporter les Leds	12
4.3. Contraintes d'alimentation par USB	14
5. Programmation, concepts généraux	15
5.1. Paradigme de programmation	15
5.2. Le module Yocto-Color	17
5.3. Interface de contrôle du module	17
5.4. Interface de la fonction ColorLed	19
5.5. Les espaces de couleurs: RGB et HSL	20
5.6. Quelle interface: Native, DLL ou Service?	21
5.7. Programmation, par où commencer?	23
6. Utilisation du Yocto-Color en ligne de commande	25
6.1. Installation	25
6.2. Utilisation: description générale	25
6.3. Contrôle de la fonction ColorLed	26
6.4. Contrôle de la partie module	27
6.5. Limitations	27
7. Utilisation du Yocto-Color en JavaScript / EcmaScript	29

7.1. Fonctions bloquantes et fonctions asynchrones en JavaScript	30
7.2. Utiliser la librairie Yoctopuce pour JavaScript / EcmaScript 2017	31
7.3. Contrôle de la fonction ColorLed	33
7.4. Contrôle de la partie module	37
7.5. Gestion des erreurs	39
8. Utilisation du Yocto-Color en PHP	41
8.1. Préparation	41
8.2. Contrôle de la fonction ColorLed	41
8.3. Contrôle de la partie module	44
8.4. API par callback HTTP et filtres NAT	46
8.5. Gestion des erreurs	49
9. Utilisation du Yocto-Color en C++	51
9.1. Contrôle de la fonction ColorLed	51
9.2. Contrôle de la partie module	54
9.3. Gestion des erreurs	56
9.4. Intégration de la librairie Yoctopuce en C++	57
10. Utilisation du Yocto-Color en Objective-C	59
10.1. Contrôle de la fonction ColorLed	59
10.2. Contrôle de la partie module	61
10.3. Gestion des erreurs	63
11. Utilisation du Yocto-Color en VisualBasic .NET	65
11.1. Installation	65
11.2. Utilisation l'API yoctopuce dans un projet Visual Basic	65
11.3. Contrôle de la fonction ColorLed	66
11.4. Contrôle de la partie module	68
11.5. Gestion des erreurs	70
12. Utilisation du Yocto-Color en C#	73
12.1. Installation	73
12.2. Utilisation l'API yoctopuce dans un projet Visual C#	73
12.3. Contrôle de la fonction ColorLed	74
12.4. Contrôle de la partie module	76
12.5. Gestion des erreurs	79
13. Utilisation du Yocto-Color en Delphi	81
13.1. Préparation	81
13.2. Contrôle de la fonction ColorLed	81
13.3. Contrôle de la partie module	84
13.4. Gestion des erreurs	86
14. Utilisation du Yocto-Color en Python	89
14.1. Fichiers sources	89
14.2. Librairie dynamique	89
14.3. Contrôle de la fonction ColorLed	90
14.4. Contrôle de la partie module	92
14.5. Gestion des erreurs	94
15. Utilisation du Yocto-Color en Java	95
15.1. Préparation	95
15.2. Contrôle de la fonction ColorLed	95

15.3. Contrôle de la partie module	97
15.4. Gestion des erreurs	100
16. Utilisation du Yocto-Color avec Android	101
16.1. Accès Natif et Virtual Hub.	101
16.2. Préparation	101
16.3. Compatibilité	101
16.4. Activer le port USB sous Android	102
16.5. Contrôle de la fonction ColorLed	104
16.6. Contrôle de la partie module	106
16.7. Gestion des erreurs	111
17. Programmation avancée	113
17.1. Programmation par événements	113
18. Mise à jour du firmware	115
18.1. Le VirtualHub ou le YoctoHub	115
18.2. La librairie ligne de commandes	115
18.3. L'application Android Yocto-Firmware	115
18.4. La librairie de programmation	116
18.5. Le mode "mise à jour"	118
19. Utilisation avec des langages non supportés	119
19.1. Ligne de commande	119
19.2. Virtual Hub et HTTP GET	119
19.3. Utilisation des librairies dynamiques	121
19.4. Port de la librairie haut niveau	124
20. Référence de l'API de haut niveau	125
20.1. Fonctions générales	126
20.2. Interface de contrôle du module	154
20.3. Interface de la fonction ColorLed	217
21. Problèmes courants	265
21.1. Linux et USB	265
21.2. Plateformes ARM: HF et EL	266
21.3. Module alimenté mais invisible pour l'OS	266
21.4. Another process named xxx is already using yAPI	266
21.5. Déconnexions, comportement erratique	266
21.6. Par où commencer ?	266
22. Caractéristiques	267
Blueprint	269
Index	271

1. Introduction

Le module Yocto-Color est un module de 60x20mm qui permet de commander indépendamment deux leds RGB. Ces leds de moyenne puissance peuvent être pilotées soit en utilisant le système de couleur RGB, soit HSL. Le module est aussi capable d'opérer une transition entre deux couleurs de manière autonome. Le module est composé de trois parties sécables: le module de commande proprement dit et chacune des deux leds. Ce qui permet de déporter les leds pour les placer dans des endroits relativement exigus. Ce module conviendra pour faire de la signalisation musclée, ou du petit éclairage d'ambiance.



Le module Yocto-Color

Yoctopuce vous remercie d'avoir fait l'acquisition de ce Yocto-Color et espère sincèrement qu'il vous donnera entière satisfaction. Les ingénieurs Yoctopuce se sont donnés beaucoup de mal pour que votre Yocto-Color 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 n'hésitez pas à contacter le support Yoctopuce¹.

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 des fonctions du module.

¹ support@yoctopuce.com

1.1. Prérequis

Pour pouvoir profiter pleinement de votre module Yocto-Color, 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, Mac OS X, Linux et Android. Les modules Yoctopuce ne nécessitent pas l'installation de driver (ou pilote) spécifiques, car ils utilisent le driver HID² 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. Yoctopuce teste régulièrement le bon fonctionnement des modules sur Windows 7 et Windows 10.

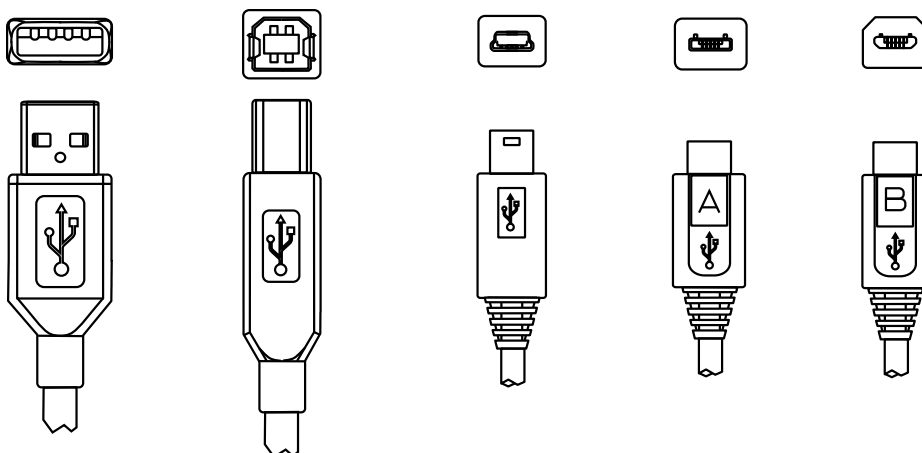
Les versions de Mac OS X actuellement supportées sont Mac OS X 10.9 (Maverick), 10.10 (Yosemite), 10.11 (El Capitan) et 10.12 (Sierra) Yoctopuce teste régulièrement le bon fonctionnement des modules sur Mac OS X 10.11.

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 3.19.

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 4.x sur un Nexus 7 et un Samsung Galaxy S3 avec la librairie Java pour Android.

Un câble USB de type A-micro B

Il existe trois tailles de connecteurs USB, 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'arboient pas une pomme. Les modules de Yoctopuce sont tous équipés d'une connectique au format micro-USB.

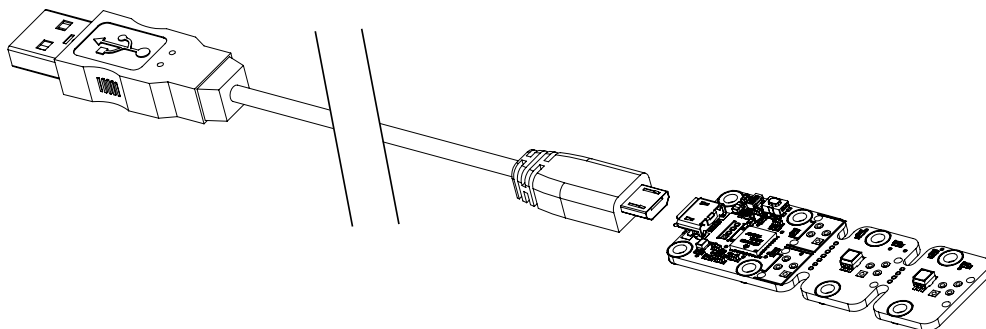


Les connecteurs USB 2 les plus courants: A, B, Mini B, Micro A, Micro B.³

² Le driver HID est celui qui gère les périphériques tels que la souris, le clavier, etc.

³ Le connecteur Mini A a existé quelque temps, mais a été retiré du standard USB http://www.usb.org/developers/Deprecation_Announcement_052507.pdf

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



Vous devez raccorder votre module Yocto-Color à l'aide d'un câble USB de type A - micro B

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

1.2. Accessoires optionnels

Les accessoires ci-dessous ne sont pas nécessaires à l'utilisation du module Yocto-Color, 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-Color à un support, vous pouvez placer des petites vis de 2.5mm avec une tête de 4.5mm au maximum dans les trous prévus ad-hoc. Il est conseillé de les visser dans des entretoises filetées, que vous pourrez fixer sur le support. Vous trouverez plus de détail à ce sujet dans le chapitre concernant le montage et la connectique.

Micro-hub USB

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

YoctoHub-Ethernet, YoctoHub-Wireless and YoctoHub-GSM

Vous pouvez ajouter une connectivité réseau à votre Yocto-Color 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 Picoflex et câble nappe souple

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

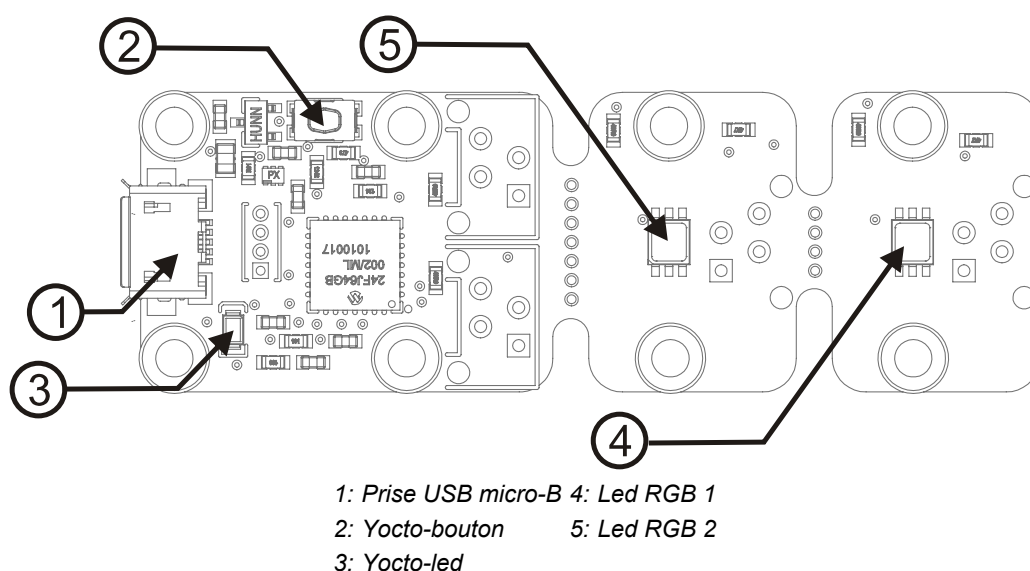
⁴ Embases Molex ref 90325-3004 ou 90325-0004, disponibles chez la plupart des fournisseurs de composants électroniques (www.mouser.com, www.digikey.com, www.farnell.com, www.distrelec.ch...). S'utilise avec les connecteurs ref 90327-3304 ou 90327-0304

Cable nappe mono-brin

Si vous désirez séparer les leds du module Yocto-Color à l'aide d'un cable nappe soudé directement aux circuit imprimé, utilisez de préférence du cable nappe étamé mono-brin: c'est le plus facile à souder. Dans tous les cas, il vous faudra 4 fils espacés de 1.27mm.

Ce même câble peut aussi être utilisé pour souder un fil directement entre le module Yocto-Color et un micro-hub USB pour éviter l'encombrement d'un cable USB.

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 au format micro-USB. Les câbles correspondants ne sont pas forcément les plus faciles à trouver, mais ces connecteurs ont l'avantage d'occuper un minimum de place.

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.

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-Color ce numéro commence par YRGBLED1. Le module peut être piloté par logiciel en utilisant ce numéro de série. Ce numéro de série ne peut pas être changé.

Le nom logique

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

2.2. Les éléments spécifiques

Les deux leds RGB

Ces deux leds ont la particularité de pouvoir émettre trois couleurs différentes: rouge, vert et bleu. Il s'agit en fait de trois leds distinctes logées dans le même boîtier. En modulant précisément la puissance de chacune ces trois leds il est possible de créer à peu près n'importe quelle couleur, blanc compris.

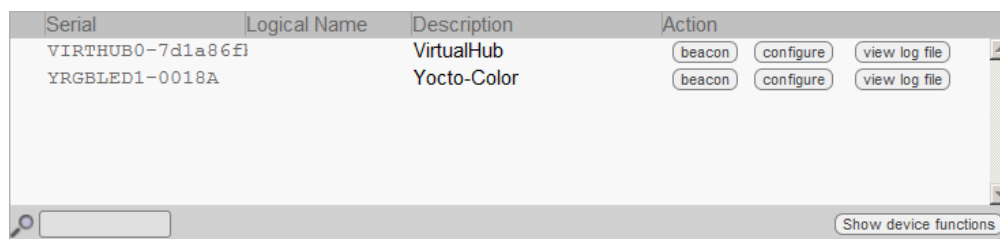
¹ court-court-court long-long-long court-court-court

² support@yoctopuce.com

3. Premiers pas

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

Rendez-vous sur le site de Yoctopuce et téléchargez le programme *Virtual Hub*¹, Il est disponible pour Windows, Linux et Mac OS X. En temps normal le programme Virtual Hub sert de couche d'abstraction pour les langages qui ne peuvent pas accéder aux couches matérielles de votre ordinateur. Mais il offre aussi une interface sommaire pour configurer vos modules et tester les fonctions de base, on accède à cette interface à l'aide d'un simple browser web ². 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
VIRTHUB0-7d1a86f1		VirtualHub	beacon configure view log file
YRGBLED1-0018A		Yocto-Color	beacon configure view log file

Search: Show device functions

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

3.1. Localisation

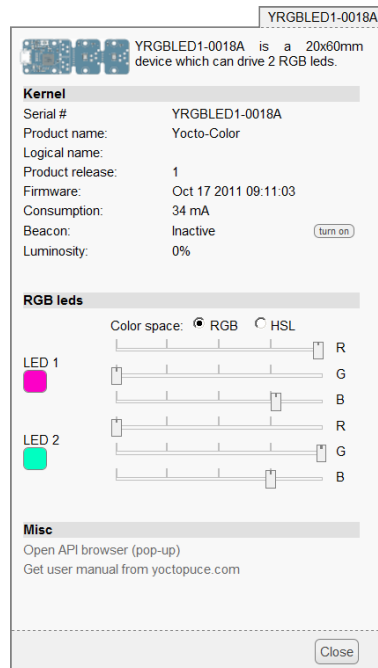
Il est alors possible de localiser physiquement chacun des modules affichés en cliquant sur le bouton **beacon**, cela a pour effet de mettre la Yocto-Led du module correspondant en mode "balise", elle se met alors à clignoter ce qui permet de la localiser facilement. Cela a aussi pour effet d'afficher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

3.2. 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-Color.

¹ www.yoctopuce.com/FR/virtualhub.php

² L'interface est testée avec Chrome, FireFox, Safari, Edge et IE 11.

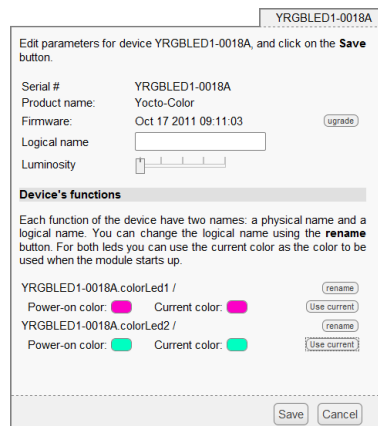


Propriétés du module Yocto-Color.

Cette fenêtre vous permet entre autres de jouer avec les deux leds RGB du module, en déplaçant les curseurs vous pouvez les couleurs affichée par les leds. Vous pouvez les piloter au choix en mode RGB ou HSL. Vous remarquerez que la consommation du module varie en fonction des valeurs affichées.

3.3. 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-Color.

Firmware

Le firmware du module peut être facilement mis à jour à l'aide de l'interface. Pour ce faire, vous devez au préalable disposer du firmware adéquat sur votre disque local. 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*³ en ligne de commande⁴.

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.

La seule fonction fournie par le module Yocto-Color est la fonction ColorLed, dont il existe deux instances: colorLed1 et colorLed2.

Couleur au démarrage

Cliquez sur le bouton **use current** pour mémoriser la couleur courante de chaque led comme couleur au démarrage. Ce réglage s'appliquera à chaque fois que le module sera remis sous tension, que ce soit par un simple chargeur USB ou par un cable USB relié à un ordinateur.

Une application de cette fonction consiste à configurer la couleur de démarrage des leds à l'aide d'un ordinateur, et ensuite d'alimenter le module à l'aide d'un simple chargeur USB. Vous pouvez ainsi facilement fabriquer un éclairage dont la couleur est configurable.

³ www.yoctopuce.com/FR/virtualhub.php

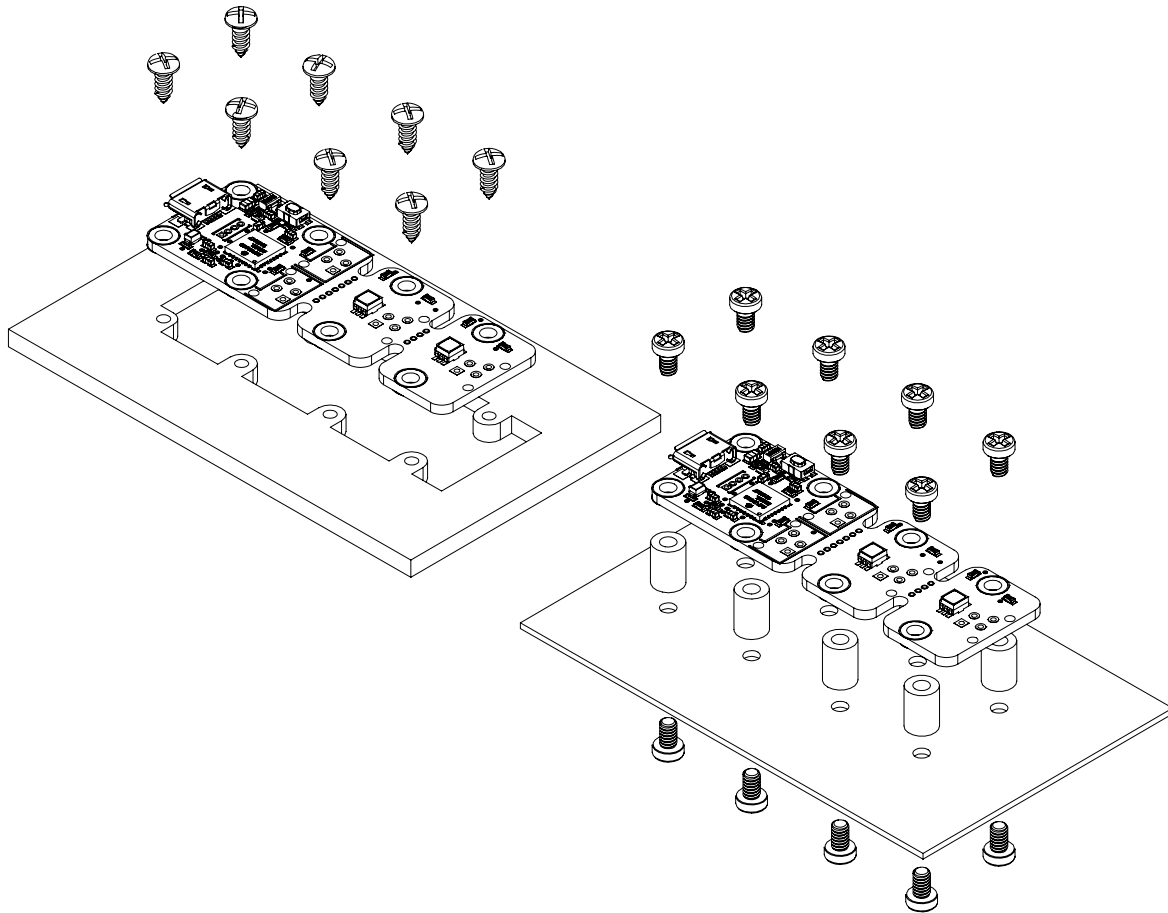
⁴ Consultez la documentation du virtual hub pour plus de détails

4. Montage et connectique

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

4.1. 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.



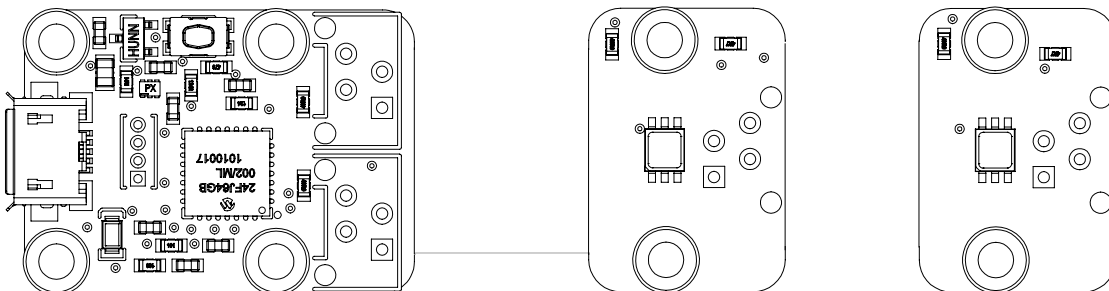
Exemples de montage sur un support.

Le module Yocto-Color dispose de trous de montage 2.5mm. Vous pouvez utiliser ces trous pour y passer des vis. Le diamètre de la tête de ces vis ne devra pas dépasser 4.5mm, sous peine d'endommager les circuits du module. Veillez à que la surface inférieure du module ne soit pas en contact avec le support. La méthode recommandée consiste à utiliser des entretoises, mais il en existe d'autres. Rien ne vous empêche de le fixer au pistolet à colle; ça ne sera pas très joli mais ça tiendra.

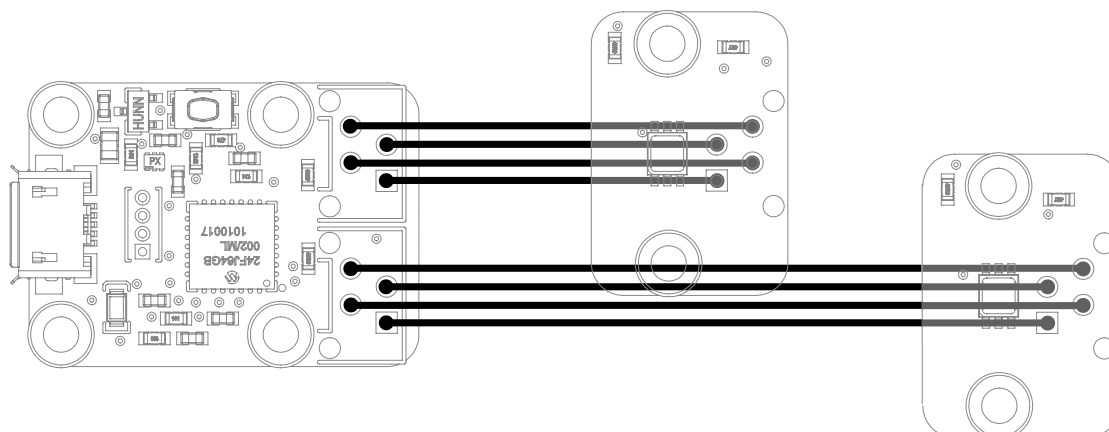
Si vous comptez visser votre module directement contre une paroi conductrice, un châssis métallique par exemple, intercalez une couche isolante entre les deux. Sinon vous allez à coup sûr provoquer un court-circuit: il y a des pads à nu sous votre module. Du simple ruban adhésif d'emballage devrait faire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

4.2. Déporter les Leds

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

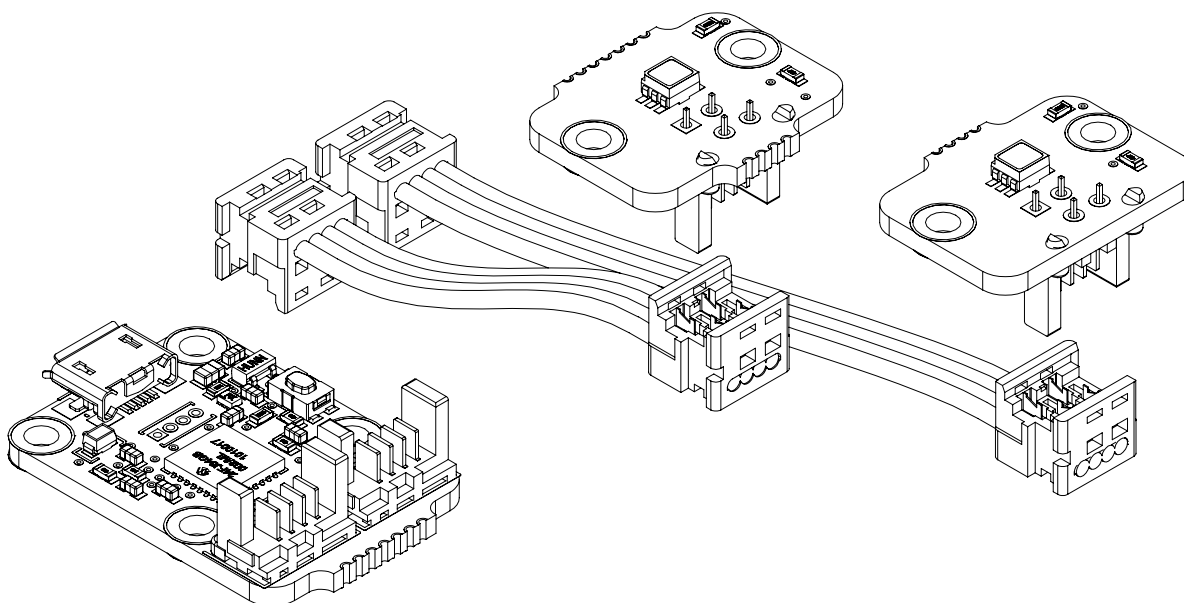


Le module Yocto-Color est conçu pour pouvoir être séparé en trois parties.



Câblage des sous module une fois ceux-ci séparés.

Une fois les modules séparés vous allez devoir les recâbler. Plusieurs solutions s'offrent à vous. Vous pouvez raccorder les sous-modules en soudant des fils électriques tout simples, c'est la solution la plus accessible mais c'est loin d'être la plus pratique. Vous pouvez utiliser du câble nappe au pas 1.27 mm. Utilisez de préférence du câble avec des conducteurs mono-brin plutôt que du multi-brin: les câbles mono-brin sont un peu moins souple, mais nettement plus facile à souder. Bien que pratique la soudure directe des câbles rend le résultat difficile à démonter. C'est pourquoi le module Yocto-Color a été conçu pour utiliser des connecteurs Picoflex de Molex¹. Ce système est largement disponible, relativement bon marché, et permet de fabriquer des solutions professionnelles sur mesure compactes et surtout démontables.



Câblage des sous-modules à l'aide de connecteurs Picoflex

Vous n'êtes pas obligé de raccorder les deux leds, le module fonctionnera parfaitement si une seule des deux leds est raccordée. Les deux sous-modules sont interchangeables: vous pouvez raccorder chaque sous-module à n'importe lequel des connecteurs du contrôleur. Il n'y a pas de moyen logiciel direct de savoir si les leds sont raccordées au module de contrôle; mais il est possible de se faire une idée en testant les variations de consommation du module: si vous donnez l'ordre d'allumer une led et que la consommation augmente, cela signifie que le sous-module correspondant est présent.

Attention, les modules Yoctopuce sécables ont souvent des systèmes de connectique très semblables. Cependant les sous-modules ne sont pas du tout compatibles entre modèles différents.

¹ Embases Molex ref 90325-3004 ou 90325-0004, disponibles chez la plupart des fournisseurs de composants électroniques (www.mouser.com, www.digikey.com, www.farnell.com, www.distrelec.ch...). S'utilise avec les connecteurs ref 90327-3304 ou 90327-0304.

Si vous raccordez un sous module de votre Yocto-Color à un autre type de module, par exemple un Yocto-Temperature, cela ne marchera pas, et vous risquez fort d'endommager votre matériel.

4.3. Contraintes d'alimentation par USB

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

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

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

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

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

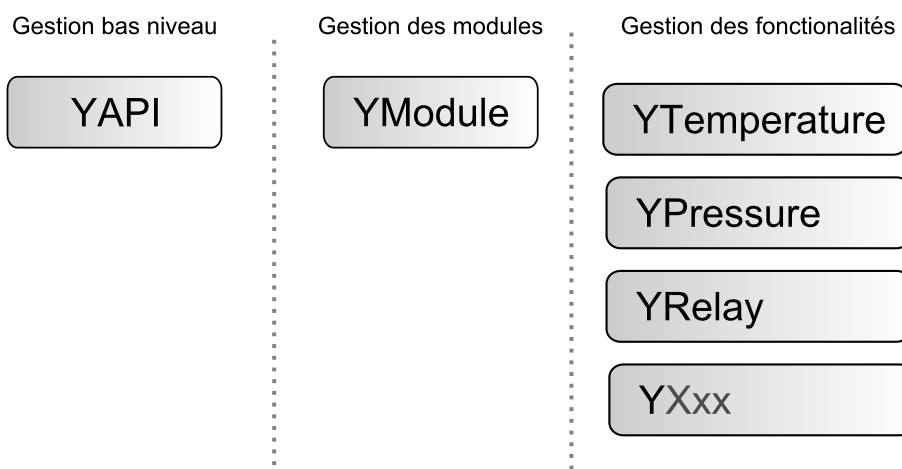
5. Programmation, concepts généraux

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-Color 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.

5.1. 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-Color 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.

5.2. Le module Yocto-Color

Le module Yocto-Color offre deux instances de la fonction *ColorLed*, correspondant aux deux leds RGB du 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

colorLed1 : ColorLed

colorLed2 : ColorLed

attribut	type	modifiable ?
logicalName	Texte	modifiable
advertisedValue	Texte	modifiable
rgbColor	Entier (24 bits)	modifiable
hslColor	Entier (24 bits)	modifiable
rgbMove	Agrégat	modifiable
hslMove	Agrégat	modifiable
rgbColorAtPowerOn	Entier (24 bits)	modifiable
blinkSeqSize	Nombre entier	lecture seule
blinkSeqMaxSize	Nombre entier	lecture seule
blinkSeqSignature	Nombre entier	lecture seule
command	Texte	modifiable

5.3. Interface de contrôle du module

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

productName

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

serialNumber

Chaîne de caractères contenant le numéro de série, unique et préprogrammé en usine. Pour un module Yocto-Color, ce numéro de série commence toujours par YRGBLED1. 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 20 en usine.

productRelease

Numéro de révision du module hardware, préprogrammé en usine.

firmwareRelease

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

persistentSettings

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

luminosity

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

beacon

Etat de la balise de localisation du module.

upTime

Temps écoulé depuis la dernière mise sous tension du module.

usbCurrent

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

rebootCountdown

Compte à rebours pour déclencher un redémarrage spontané du module.

userVar

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

5.4. Interface de la fonction ColorLed

La librairie de programmation Yoctopuce permet de piloter une LED couleur aussi bien en coordonnées RGB qu'en coordonnées HSL, les conversions RGB vers HSL étant faites automatiquement par le module. Ceci permet aisément d'allumer la LED avec une certaine teinte et d'en faire progressivement varier la saturation ou la luminosité. Si nécessaire, vous trouverez plus d'information sur la différence entre RGB et HSL dans la section suivante.

logicalName

Chaîne de caractères contenant le nom logique de la LED RGB, initialement vide. Cet attribut peut être changé au bon vouloir de l'utilisateur. Un fois initialisé à une valeur non vide, il peut servir de point de départ pour accéder directement à la LED RGB. Si deux LEDs RGB 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 la LED RGB, et qui sera publiée automatiquement jusqu'au hub parent. Pour une LED RGB, la valeur publiée est la couleur RGB de la LED (6 digits hexadécimaux).

rgbColor

Couleur courante de la LED, sous la forme d'un entier encodé en RGB. Pour en extraire les composantes rouges verte et bleus il faut pour le rouge: faire un shift à droite de 16 bit, pour le vert faire un shift à droite de 8 bit et ne garder que les 8 bits de droite, pour le bleu ne garder que les 8 bits de droite.

hslColor

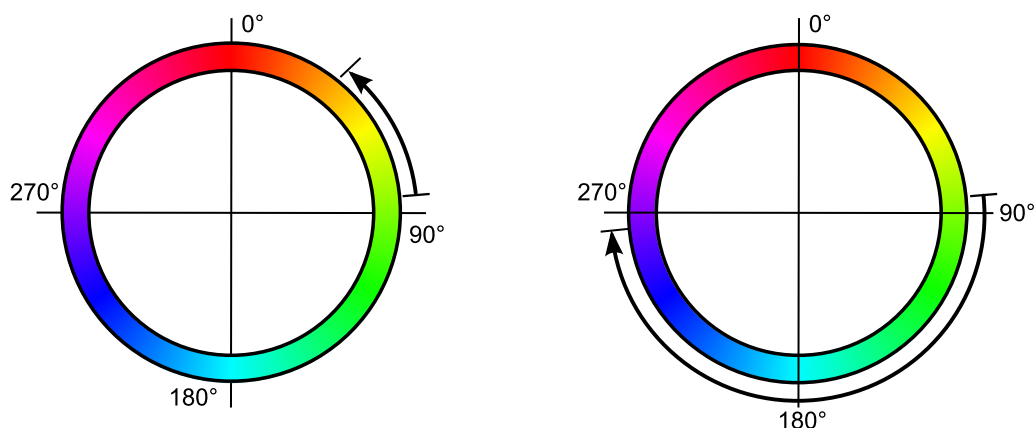
Couleur courante de la LED (comme **rgbColor**), mais exprimée dans le système de couleur HSL.

rgbMove

Permet de faire passer la LED de la couleur courante à une autre de manière continue et indépendante. La valeur est un agrégat comprenant la couleur cible encodée en RGB dans un entier, et une durée en millisecondes.

hslMove

Fonctionne de manière identique à **rgbMove**, mais en utilisant le système de couleurs HSL. En HSL, la teinte étant une valeur circulaire (0..360°) il y a toujours deux manières d'opérer la transition: en augmentant ou en diminuant le teinte. Le module optera pour la transition passant par le chemin le plus court. dans le cas d'une différence d'exactly 180°, il optera pour la transition qui augmente la valeur de la teinte.



En HSL, le module choisit la transition de teinte la plus courte. Dans le cas d'une différence de 180° la transition se fera dans le sens horaire.

rgbColorAtPowerOn

Couleur de la LED à rétablir lors la mise sous tension du module.

blinkSeqSize

Taille actuelle de la séquence de clignotement

blinkSeqMaxSize

Taille maximale de la séquence de clignotement

blinkSeqSignature

Signature de la séquence de clignotement

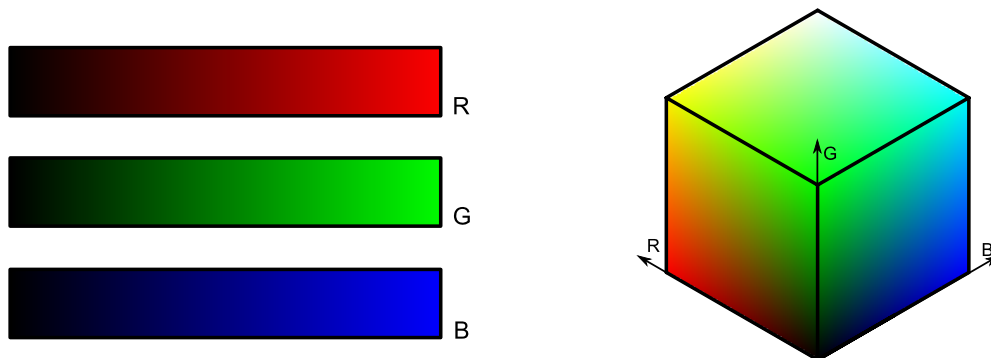
command

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

5.5. Les espaces de couleurs: RGB et HSL

Espace RGB

Il existe plusieurs manières de définir une couleur. La plus connue consiste à utiliser ses composantes Rouge, Verte, et Bleue (RGB en anglais). Chaque couleur peut en effet être définie comme un mélange des trois couleurs primaires. Malheureusement ce système est assez difficile à manipuler avec précision dès qu'il s'agit de définir une couleur non saturée. Obtenir du rose bonbon par exemple, demande pas mal de tâtonnements avant d'arriver au résultat escompté.



L'espace de couleur RGB, il est souvent représenté sous la forme d'un cube.

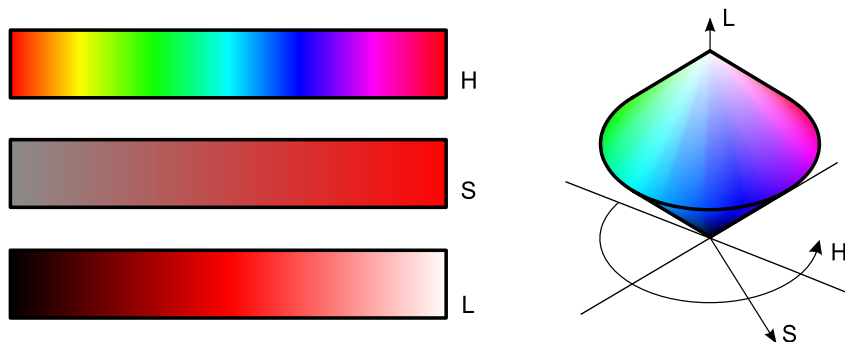
En informatique on a l'habitude de définir une couleur RGB comme un triplet 0..255 exprimé en hexadécimal¹. Ainsi le rouge pur est défini par 0xFF0000, le vert pur 0x00FF00, le bleu pur 0x0000FF, le noir 000000, le blanc 0xFFFFFF etc. Cette notation est, entre autres, utilisés en HTML pour définir les couleurs, on utilise alors le # comme préfixe au lieu du 0x.

Espace HSL

Il existe une autre manière, un peu plus simple, de définir une couleur. Elle consiste à décomposer une couleur en Teinte, Saturation et Luminosité (HSL en anglais). La teinte est définie de 0 à 360° et passe successivement par le rouge (0°) le jaune (60°), le vert (120°) le cyan (180°) le bleu (240°), le magenta (300°) pour finalement revenir au rouge (360°). La saturation, définie de 0 à 100%, décrit l'intensité de la couleur, plus on se rapproche du 0% plus la couleur paraît délavée, plus on s'approche du 100% plus la couleur apparaît éclatante. La luminosité, définie de 0 à 100%, décrit la

¹ L'hexadécimal consiste à compter en base 16 au lieu de base 10: on utilise les signes 0..9, A..F. On utilise parfois le préfixe 0x pour signaler l'utilisation de l'hexadécimal: ainsi 10 en base 10 s'écrit 0xA en hexadécimal, 15 s'écrit 0xF, 16 s'écrit 0x10, 17 s'écrit 0x11 et ainsi de suite

clarté de la couleur, elle devient plus sombre quand on se rapproche du 0% et devient plus claire quand on se rapproche du 100%. Lorsqu'on utilise ce système il est beaucoup plus facile de prévoir comment va évoluer une couleur quand on manipule une de ces coordonnées. Ce qui est loin d'être le cas avec le système RGB.



L'espace de couleur HSL, il souvent représenté sous la forme d'un double cone, dont la base est noire (L=0%) et la pointe supérieure est blanche (L=100%). La teinte est mappée sur le circonférence et la saturation sur le rayon.

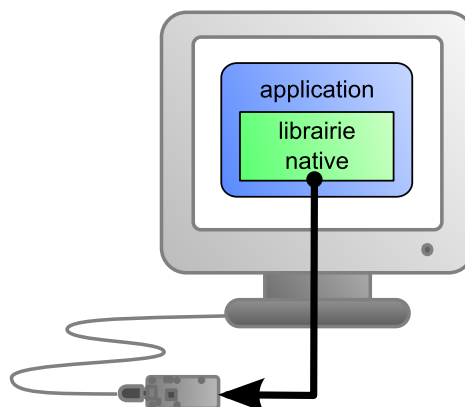
En informatique on a tendance à normaliser les coordonnées HSL sur une gamme 0..255 et on représente une couleur HSL sous la forme d'un triplet hexadécimal. Ainsi 00FF80 représente le rouge pur, F0FF80 représente le bleu pur. Il est à noter que dans ce système, plusieurs notations différentes peuvent représenter la même couleur. Ainsi toutes les notations se terminant par 00 représentent le noir, toutes les notations se terminant par FF représentent le blanc. Une notation dont la saturation est zéro représente du gris, quelque soit la valeur de la teinte.

5.6. Quelle interface: Native, DLL ou Service?

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

Contrôle natif

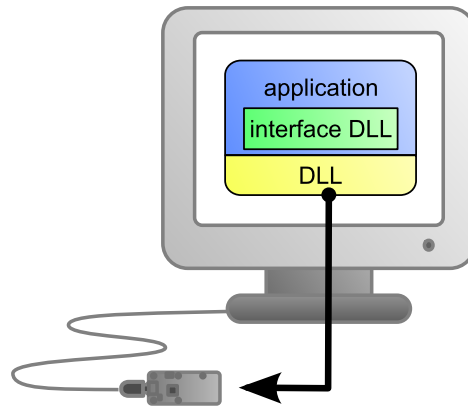
Dans ce cas de figure le programme pilotant votre projet est directement compilé avec une librairie qui offre le contrôle des modules. C'est objectivement la solution la plus simple et la plus élégante pour l'utilisateur final. Il lui suffira de brancher le câble USB et de lancer votre programme pour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



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

Contrôle natif par DLL

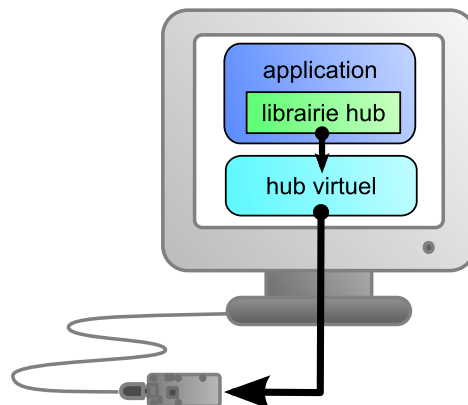
Ici l'essentiel du code permettant de contrôler les modules se trouve dans une DLL, et le programme est compilé avec une petite librairie permettant de contrôler cette DLL. C'est la manière la plus rapide pour coder le support des modules dans un langage particulier. En effet la partie "utile" du code de contrôle se trouve dans la DLL qui est la même pour tous les langages, offrir le support pour un nouveau langage se limite à coder la petite librairie qui contrôle la DLL. Du point de 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

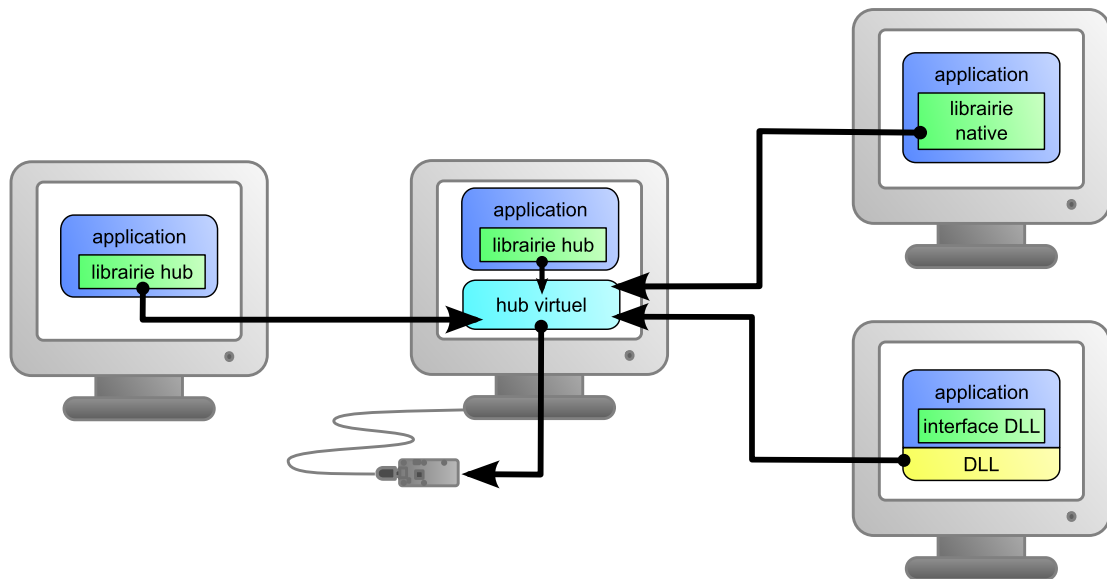
Contrôle par un service

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 librairie qui offrira toutes les fonctions nécessaires au contrôle des modules en passant par l'intermédiaire de ce VirtualHub. L'utilisateur final se verra obligé de lancer le VirtualHub avant de lancer le programme de contrôle du projet proprement dit, à moins qu'il ne décide d'installer le VirtualHub sous la forme d'un service/démon, auquel cas le VirtualHub se lancera automatiquement au démarrage de la machine..



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

En revanche la méthode de contrôle par un service offre un avantage non négligeable: l'application n'est pas 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	Hub virtuel
C++	✓	✓	✓
Objective-C	✓	-	✓
Delphi	-	✓	✓
Python	-	✓	✓
VisualBasic .Net	-	✓	✓
C# .Net	-	✓	✓
EcmaScript / JavaScript	-	-	✓
PHP	-	-	✓
Java	-	✓	✓
Java pour Android	✓	-	✓
Ligne de commande	✓	-	✓

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'elles 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()`.

5.7. Programmation, par où commencer?

Arrivé à ce point du manuel, vous devriez connaître l'essentiel de la théorie à propos de votre Yocto-Color. 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 Yoctopuce². 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.

² <http://www.yoctopuce.com/FR/libraries.php>

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

6. Utilisation du Yocto-Color en ligne de commande

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

6.1. 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-Color, ouvrir un shell et commencer à travailler en tapant par exemple:

```
C:\>YColorLed any set_rgbColor #0000FF
```

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*.

6.2. Utilisation: description générale

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

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

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

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.

¹ Si vous souhaitez recompiler l'API en ligne de commande, vous aurez aussi besoin de l'API C++

² <http://www.yoctopuce.com/FR/libraries.php>

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

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

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

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

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

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

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

6.3. Contrôle de la fonction ColorLed

Pour contrôler la fonction `ColorLed` de votre Yocto-Color, vous avez besoin de l'exécutable `YColorLed`.

Vous pouvez par exemple lancer:

```
C:\>YColorLed any set_rgbColor #0000FF
```

Cet exemple utilise la cible "`any`" pour signifier que l'on désire travailler sur la première fonction `ColorLed` 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-Color avec le numéros de série `YRGBLED1-123456` que vous auriez appelé "`MonModule`" et dont vous auriez nommé la fonction `colorLed1` "`MaFonction`", les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté).

```
C:\>YColorLed YRGBLED1-123456.colorLed1 describe
C:\>YColorLed YRGBLED1-123456.MaFonction describe
C:\>YColorLed MonModule.colorLed1 describe
C:\>YColorLed MonModule.MaFonction describe
C:\>YColorLed MaFonction describe
```

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

```
C:\>YColorLed all describe
```

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

```
C:\>YColorLed /help
```


6.4. 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 YRGBLED1-12346 set_logicalName MonPremierModule
C:\>YModule YRGBLED1-12346 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 YRGBLED1-12346 set_logicalName MonPremierModule
C:\>YModule YRGBLED1-12346 saveToFlash
```

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

```
C:\>YModule -s YRGBLED1-12346 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.

6.5. Limitations

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

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

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

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

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

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

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

7. Utilisation du Yocto-Color en JavaScript / EcmaScript

EcmaScript est le nom officiel de la version standardisée du langage de programmation communément appelé JavaScript. Cette librairie de programmation Yoctopuce utilise les nouvelles fonctionnalités introduites dans la version EcmaScript 2017. La librairie porte ainsi le nom *Librairie pour JavaScript / EcmaScript 2017*, afin de la différencier de la précédente *Librairie pour JavaScript* qu'elle remplace.

Cette librairie permet d'accéder aux modules Yoctopuce depuis tous les environnements JavaScript modernes. Elle fonctionne aussi bien depuis un navigateur internet que dans un environnement Node.js. La librairie détecte automatiquement à l'initialisation si le contexte d'utilisation est un browser ou une machine virtuelle Node.js, et utilise les librairies systèmes les plus appropriées en conséquence.

Les communications asynchrones avec les modules sont gérées dans toute la librairie à l'aide d'objets *Promise*, en utilisant la nouvelle syntaxe EcmaScript 2017 `async / await` non bloquante pour la gestion des entrées/sorties asynchrones (voir ci-dessous). Cette syntaxe est désormais disponible sans autres dans la plupart des moteurs JavaScript: il n'est plus nécessaire de transpiler le code avec Babel ou `jspm`. Voici la version minimum requise de vos moteurs JavaScript préférés, tous disponibles au téléchargement:

- Node.js v7.6 and later
- Firefox 52
- Opera 42 (incl. Android version)
- Chrome 55 (incl. Android version)
- Safari 10.1 (incl. iOS version)
- Android WebView 55
- Google V8 Javascript engine v5.5

Si vous avez besoin de la compatibilité avec des anciennes versions, vous pouvez toujours utiliser Babel pour transpiler votre code et la librairie vers un standard antérieur de JavaScript, comme décrit un peu plus bas.

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

7.1. 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.

7.2. 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`), un fichier de configuration pour jspm (`jspm.config.js`) et le code source de l'application (`src/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);
```

7.3. Contrôle de la fonction ColorLed

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

```
import { YAPI, YColorLed } from 'yoctolib-es';

// On récupère l'objet représentant le module, à travers le VirtualHub local
await YAPI.RegisterHub('127.0.0.1');
var colorled = YColorLed.FindColorLed("YRGBLED1-123456.colorLed1");

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

Voyons maintenant en détail ce que font ces quelques lignes.

Import de YAPI et YColorLed

Ces deux imports permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. YAPI doit toujours être inclus, YColorLed est nécessaire pour gérer les modules contenant une LED RGB, comme le Yocto-Color. 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.

YColorLed.FindColorLed

La méthode `FindColorLed`, permet de retrouver une LED RGB 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-Color avec le numéros de série `YRGBLED1-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `colorLed1` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```

colorLed = YColorLed.FindColorLed("YRGBLED1-123456.colorLed1")
colorLed = YColorLed.FindColorLed("YRGBLED1-123456.MaFonction")
colorLed = YColorLed.FindColorLed("MonModule.colorLed1")
colorLed = YColorLed.FindColorLed("MonModule.MaFonction")
colorLed = YColorLed.FindColorLed("MaFonction")

```

`YColorLed.FindColorLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la LED RGB.

isOnline

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

set_rgbColor

La méthode `set_rgbColor()` de l'objet renvoyé par `YColorLed.FindColorLed` permet de régler la couleur de la led correspondante. La Couleur est donnée en coordonnées RGB codées sous forme d'un entier.

rgbMove

Il existe une méthode alternative pour changer de couleur. La méthode `rgbMove()` de l'objet renvoyé par `YColorLed.FindColorLed` permet de lancer une transition entre la couleur courante et une autre couleur RGB donnée en paramètre. Cette fonction prend aussi en paramètre le temps accordé pour effectuer la transition.

Les méthodes `set_rgbColor()` et `rgbMove()` existent aussi en version HSL. En guise d'exercice, vous pourriez modifier l'exemple ci-dessous pour utiliser le système HSL. Vous pourriez ainsi observer les différences de comportement. En HSL pour le rouge est codé `0xFFFF80`, le vert est codé `0x55FF80` et du bleu est codé `0xA9FF80`.

Un exemple concret

Ouvrez une fenêtre de commande (un terminal, un shell...) et allez dans le répertoire **example_node/Doc-GettingStarted-Yocto-Color** de la librairie Yoctopuce pour EcmaScript. Vous y trouverez dans le sous-répertoire **src** 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-Color 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-Color et où vous avez lancé le VirtualHub.

```

import { YAPI, YErrorMsg, YColorLed } from 'yoctolib-es';

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

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

  // Select the relay to use
  let target;
  if(args[0] == "any") {
    let anyLed = YColorLed.FirstColorLed();
    if (anyLed == null) {
      console.log("No module connected (check USB cable)\n");
      process.exit(1);
    }
    let module = await anyLed.get_module();
    target = await module.get_serialNumber();
  } else {

```



```

    target = args[0];
  }

  let led1 = YColorLed.FindColorLed(target+'.colorLed1');
  let led2 = YColorLed.FindColorLed(target+'.colorLed2');

  let color;
  if (args[1] == "red")
    color = 0xFF0000;
  else if (args[1] == "green")
    color = 0x00FF00;
  else if (args[1] == "blue")
    color = 0x0000FF;
  else
    color = parseInt(args[1],16);

  if(await led1.isOnline()) {
    // Change the color in two different ways
    led1.set_rgbColor(color); // immediate switch
    led2.rgbMove(color,1000); // smooth transition
  } else {
    console.log("Module not connected (check identification and USB cable)\n");
  }

  await YAPI.FreeAPI();
}

if(process.argv.length < 5) {
  console.log("usage: jspm run src/demo.js <serial_number> [ color | rgb ]");
  console.log("        jspm run src/demo.js <logical name> [ color | rgb ]");
  console.log("        jspm run src/demo.js any [ color | rgb ]          (use any discovered device)");
  console.log("Eg.");
  console.log("    jspm run src/demo.js any FF1493 ");
  console.log("    jspm run src/demo.js YRGBLED1-123456 red");
} else {
  startDemo(process.argv.slice(process.argv.length - 2));
}

```

Comme décrit au début de ce chapitre, vous devez avoir installé Node.js et jspm 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
jspm 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:

```

jspm run src/demo.js [...]

```

Le même exemple, mais dans un navigateur

Si vous voulez voir comment utiliser la librairie dans un navigateur, changez de répertoire et allez dans **example_html/Doc-GettingStarted-Yocto-Color**. Vous y trouverez aussi dans le sous-répertoire **src** un code très similaire (ci-dessous), avec quelques variantes par rapport au précédent, pour permettre une interaction à travers une page HTML plutôt que sur la console JavaScript

```

import { YAPI, YErrorMsg, YColorLed } from 'yoctolib-es';

var led1, led2;

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

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

```

```

    }
    refresh();
}

async function refresh()
{
    let serial = document.getElementById('serial').value;
    if(serial == '') {
        // by default use any connected module suitable for the demo
        let anyLed = YColorLed.FirstColorLed();
        if(anyLed) {
            let module = await anyLed.module();
            serial = await module.get_serialNumber();
            document.getElementById('serial').value = serial;
        }
    }
    led1 = YColorLed.FindColorLed(serial+'.colorLed1');
    led2 = YColorLed.FindColorLed(serial+'.colorLed2');

    if(await led1.isOnline()) {
        document.getElementById('msg').value = '';
    } else {
        document.getElementById('msg').value = 'Module not connected';
    }
    setTimeout(refresh, 500);
}

window.setColor = function(color)
{
    // Change the color in two different ways
    led1.set_rgbColor(color); // immediate switch
    led2.rgbMove(color,1000); // smooth transition
};

startDemo();

```

Vous trouverez aussi à la racine de l'exemple un fichier **demo.html** qui contient les éléments d'interface HTML de l'application de démonstration:

```

<!DOCTYPE html>
<html>
<head>
    <title>Hello World</title>
    <script src='jspm_packages/system.js'></script>
    <script src='jspm.browser.js'></script>
    <script src='jspm.config.js'></script>
    <script>
        System.import('app/helloworld.js');
    </script>
    <!-- When going in production, you can generate a self-contained js file using

jspm build --minify src/demo.js demo-sfx.js

and replace the 6 lines above by just this one:

<script src='demo-sfx.js'></script>
-->
</head>
<body>
    Module to use: <input id='serial'>
    <input id='msg' style='color:red;border:none;' readonly><br>
    <a href='javascript:setColor(0xFF0000);'>Red</a><br>
    <a href='javascript:setColor(0x00FF00);'>Green</a><br>
    <a href='javascript:setColor(0x0000FF);'>Blue</a>
</body>
</html>

```

Comme précédemment, les deux commandes suivantes vont charger et installer automatiquement les composants nécessaires pour l'exécution de cet exemple:

```

npm install
jspm install

```

Il ne vous reste plus qu'à publier ce répertoire sur un serveur Web pour pouvoir tester l'exemple à travers un navigateur. Pour que le *loader* retrouve ses fichiers, vous devrez simplement indiquer dans le paramètre **baseURL** du fichier **jspm.browser.js** le chemin qui permet d'atteindre le projet, depuis la racine du serveur web. Par exemple, si vous accédez à l'exemple à travers l'URL **http://127.0.0.1/EcmaScript/example_html/Doc-GettingStarted-Yocto-Color/demo.html** vous devrez mettre dans **jspm.browser.js**:

```
SystemJS.config({
  baseURL: "/EcmaScript/example_html/Doc-GettingStarted-Yocto-Color/",
  ...
})
```

Si vous préférez lancer l'exemple en ouvrant un fichier local plutôt qu'à travers un serveur web, ou simplement si vous préférez que votre exemple ne se charge pas sous forme de multiples modules JavaScript indépendants, vous pouvez *builder* votre projet avec la commande:

```
jspm build --minify src/demo.js demo-sfx.js
```

Ceci crée un unique fichier JavaScript nommé **demo-sfx.js** à la racine du projet d'exemple, que vous pouvez directement inclure dans le fichier HTML à la place des 6 lignes de script initiales:

```
<script src='demo-sfx.js'></script>
```

Une fois *buildé* de cette façon, le projet être ouvert directement par un navigateur depuis le disque.

7.4. 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-es';

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 < 3) {
  console.log("usage: jspm run src/demo.js <serial or logicalname> [ ON | OFF ]");
} else {
  startDemo(process.argv.slice(process.argv.length - 3));
}

```

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 { YAPI, YErrorMsg, YModule } from 'yoctolib-es';

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) {
      var 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 < 3) {
  console.log("usage: jspm run src/demo.js <serial> [newLogicalName]");
} else {
  startDemo(process.argv.slice(process.argv.length - 3));
}

```

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

```
import { YAPI, YModule, YErrorMsg } from 'yoctolib-es';

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);
}
```

7.5. Gestion des erreurs

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

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

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

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

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

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

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

8. Utilisation du Yocto-Color 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.

8.1. 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.

8.2. Contrôle de la fonction ColorLed

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

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

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

² www.yoctopuce.com/FR/libraries.php

³ www.yoctopuce.com/FR/virtualhub.php

```
// On récupère l'objet représentant le module, à travers le VirtualHub local
yRegisterHub('http://127.0.0.1:4444/', $errmsg);
$colorled = yFindColorLed("YRGBLED1-123456.colorLed1");

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.php et yocto_colorled.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_colorled.php` est nécessaire pour gérer les modules contenant une LED RGB, comme le Yocto-Color.

yRegisterHub

La fonction `yRegisterHub` permet d'indiquer sur quelle machine se trouve les modules Yoctopuce, ou plus exactement sur quelle machine tourne le programme VirtualHub. Dans notre cas l'adresse `127.0.0.1:4444` indique la machine locale, en utilisant le port `4444` (le port standard utilisé par Yoctopuce). Vous pouvez parfaitement changer cette adresse, et mettre l'adresse d'une autre machine sur laquelle tournerait un autre VirtualHub.

yFindColorLed

La fonction `yFindColorLed`, permet de retrouver une LED RGB 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-Color avec le numéros de série `YRGBLED1-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `colorLed1` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
$colorled = yFindColorLed("YRGBLED1-123456.colorLed1");
$colorled = yFindColorLed("YRGBLED1-123456.MaFonction");
$colorled = yFindColorLed("MonModule.colorLed1");
$colorled = yFindColorLed("MonModule.MaFonction");
$colorled = yFindColorLed("MaFonction");
```

`yFindColorLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la LED RGB.

isOnline

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

set_rgbColor

La méthode `set_rgbColor()` de l'objet renvoyé par `yFindColorLed` permet de régler la couleur de la led correspondante. La Couleur est donnée en coordonnées RGB codées sous forme d'un entier.

rgbMove

Il existe une méthode alternative pour changer de couleur. La méthode `rgbMove()` de l'objet renvoyé par `yFindColorLed` permet de lancer une transition entre la couleur courante et une autre couleur RGB donnée en paramètre. Cette fonction prend aussi en paramètre le temps accordé pour effectuer la transition.

Les méthodes `set_rgbColor()` et `rgbMove()` existent aussi en version HSL. En guise d'exercice, vous pourriez modifier l'exemple ci-dessous pour utiliser le système HSL. Vous pourriez ainsi observer les différences de comportement. En HSL pour le rouge est codé `0xFFFF80`, le vert est codé `0x55FF80` et du bleu est codé `0xA9FF80`.

Un exemple réel

Ouvrez votre éditeur de texte préféré⁴, recopiez le code ci dessous, sauvez-le dans un répertoire accessible par votre serveur web/PHP avec les fichiers de la librairie, et ouvrez-la page avec votre browser favori. Vous trouverez aussi ce code dans le répertoire **Exemples/Doc-GettingStarted-Yocto-Color** de la librairie Yoctopuce.

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

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

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

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

      @$serial = $_GET['serial'];
      if ($serial != '') {
        // Check if a specified module is available online
        $led1 = yFindColorLed("$serial.colorLed1");
        $led2 = yFindColorLed("$serial.colorLed2");
        if (!$led1->isOnline()) {
          die("Module not connected (check serial and USB cable)");
        }
      } else {
        // or use any connected module suitable for the demo
        $led1 = yFirstColorLed();
        if(is_null($led1)) {
          die("No module connected (check USB cable)");
        } else {
          $led2 = $led1->nextColorLed();
          $serial = $led1->module()->get_serialnumber();
        }
      }
      Print("Module to use: <input name='serial' value='$serial'><br>");

      // Drive the selected module
      if (isset($_GET['color'])) {
        // Change the color in two different ways
        $color = hexdec($_GET['color']);
        $led1->set_rgbColor($color); // immediate switch
        $led2->rgbMove($color,1000); // smooth transition
      }
      yFreeAPI();
    ?>
    <input type='radio' name='color' value='0xFF0000'>Red
    <input type='radio' name='color' value='0x00FF00'>Green
    <input type='radio' name='color' value='0x0000FF'>Blue
    <br><input type='submit'>
  </FORM>
</BODY>
</HTML>
```

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

8.3. Contrôle de la partie module

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

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

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

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

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

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

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

Modifications des réglages du module

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

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

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

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

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

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

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

Enumération des modules

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

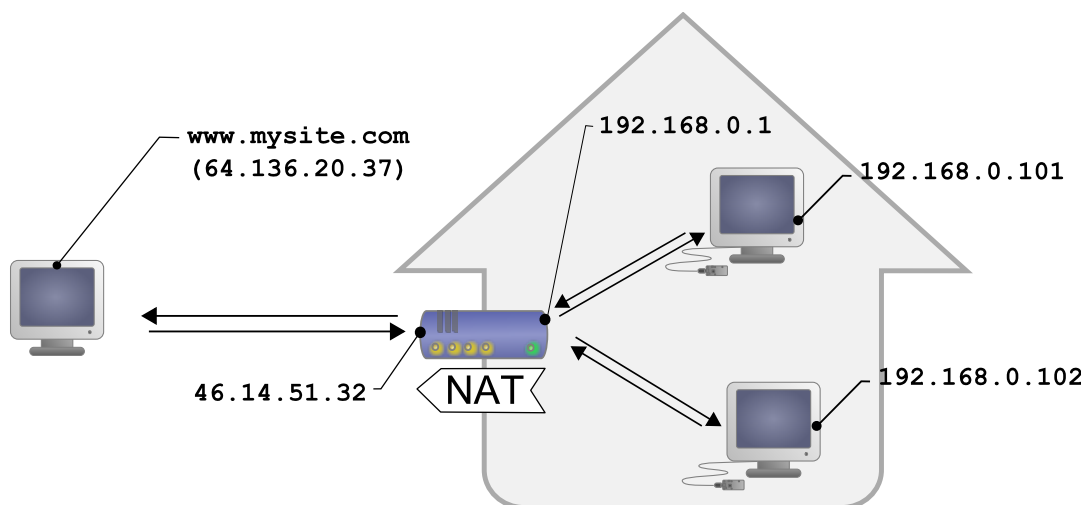
```
<HTML>
<HEAD>
<TITLE>inventory</TITLE>
</HEAD>
<BODY>
<H1>Device list</H1>
<TT>
<?php
include('yocto_api.php');
yRegisterHub("http://127.0.0.1:4444/");
$module = yFirstModule();
while (!is_null($module)) {
    printf("%s (%s)<br>", $module->get_serialNumber(),
        $module->get_productName());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

8.4. 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

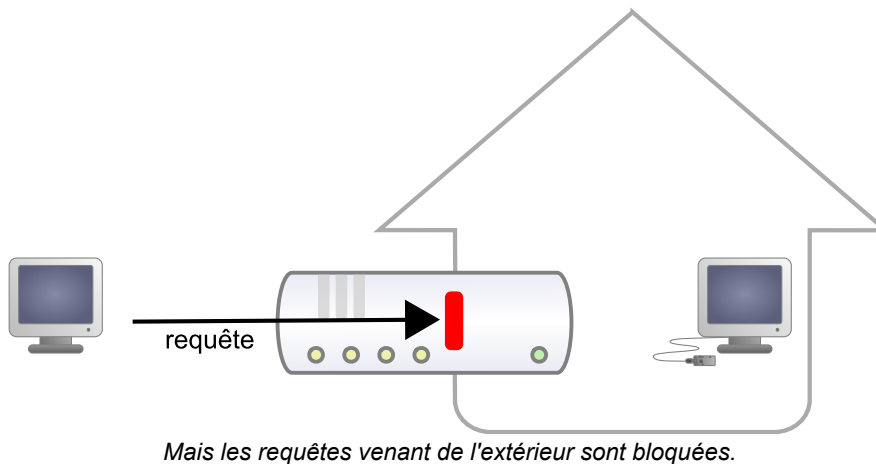
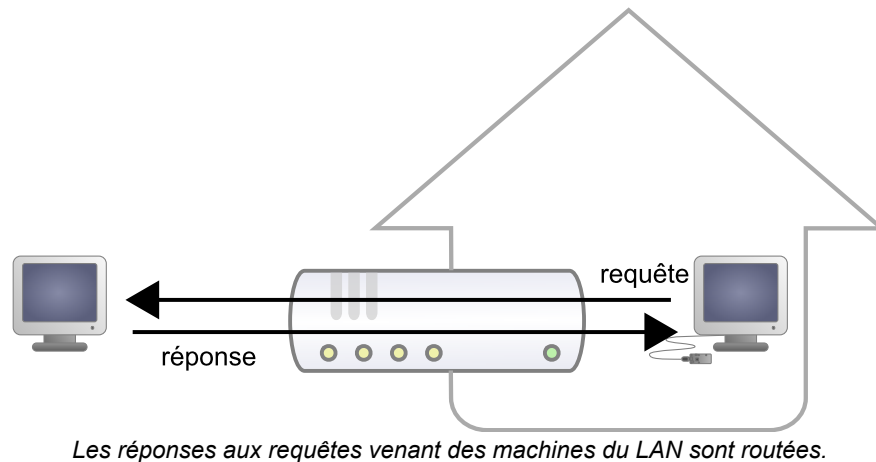
Un routeur DSL qui effectue de la traduction d'adresse réseau (NAT) fonctionne un peu comme un petit central téléphonique privé: les postes internes peuvent s'appeler l'un l'autre ainsi que faire des appels vers l'extérieur, mais vu de l'extérieur, il n'existe qu'un numéro de téléphone officiel, attribué 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

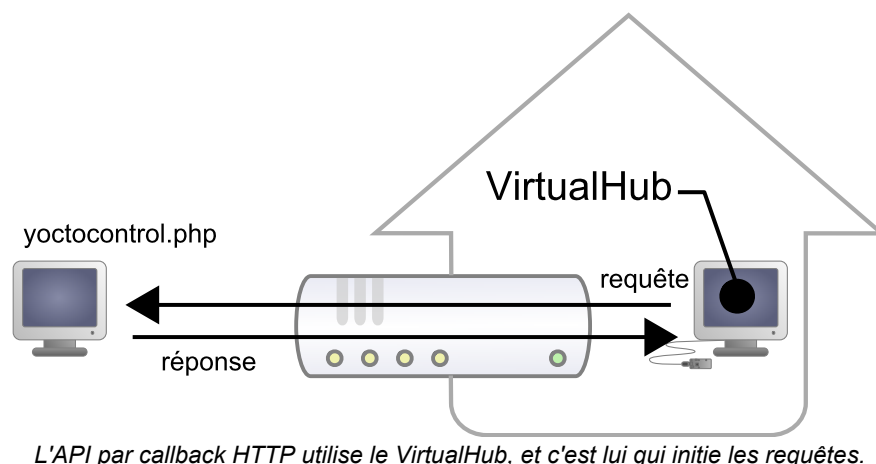
Ce qui, transposé en terme de réseau, donne : les appareils connectés sur un réseau domestique peuvent communiquer entre eux en utilisant une adresse IP locale (du genre 192.168.xxx.yyy), et contacter des serveurs sur Internet par leur adresse publique, mais vu de l'extérieur, il n'y a qu'une

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.



Voir Internet sans être vu représente un avantage de sécurité énorme. Cependant, cela signifie qu'a priori, on ne peut pas simplement monter son propre serveur Web public chez soi pour une installation domotique et offrir un accès depuis l'extérieur. Une solution à ce problème, préconisée par de nombreux vendeurs de domotique, consiste à donner une visibilité externe au serveur de domotique lui-même, en ouvrant un port et en ajoutant une règle de routage dans la configuration NAT du routeur DSL. Le problème de cette solution est qu'il expose le serveur de domotique aux attaques externes.

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



Configuration

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

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

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

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

Cliquer sur le bouton "configure" de la première ligne

VIRTHUB0-7d1a86fb09

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

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

Incoming connections

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

Outgoing callbacks

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

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

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

Edit callback

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

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

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

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

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

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

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

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

Et choisir "Yocto-API callback".

Il suffit alors de définir l'URL du script PHP et, si nécessaire, le nom d'utilisateur et le mot de passe pour accéder à cette URL. Les méthodes d'authentification supportées sont *basic* et *digest*. La

seconde est plus sûre que la première car elle permet de ne pas transférer le mot de passe sur le réseau.

Utilisation

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

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

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

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

Problèmes courants

Pour que l'API par callback HTTP fonctionne, l'option de PHP `allow_url_fopen` doit être activée. Certains hébergeurs de site web ne l'activent pas par défaut. Le problème se manifeste alors avec l'erreur suivante:

```
error: URL file-access is disabled in the server configuration
```

Pour activer cette option, il suffit de créer dans le même répertoire que le script PHP de contrôle un fichier `.htaccess` contenant la ligne suivante:

```
php_flag "allow_url_fopen" "On"
```

Selon la politique de sécurité de l'hébergeur, il n'est parfois pas possible d'autoriser cette option à la racine du site web, où même d'installer des scripts PHP recevant des données par un POST HTTP. Dans ce cas il suffit de placer le script PHP dans un sous-répertoire.

Limitations

Cette méthode de fonctionnement qui permet de passer les filtres NAT à moindre frais a malgré tout un prix. Les communications étant initiées par le *Virtual Hub* à intervalle plus ou moins régulier, le temps de réaction à un événement est nettement plus grand que si les modules Yoctopuce étaient pilotés en direct. Vous pouvez configurer le temps de réaction dans la fenêtre ad-hoc du *Virtual Hub*, mais il sera nécessairement de quelques secondes dans le meilleur des cas.

Le mode *Yocto-API par callback HTTP* n'est pour l'instant disponible qu'en PHP et Node.JS.

8.5. Gestion des erreurs

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

La manière la plus simple de contourner le problème est celle que nous avons employé pour les petits exemples précédents de ce chapitre: avant d'accéder à un module, on vérifie qu'il est en ligne avec la méthode `isOnline()` et on suppose ensuite qu'il va y rester pendant la fraction de

seconde nécessaire à exécuter les lignes de code suivantes. Ce n'est pas parfait, mais ça peut suffire dans certains cas. Il faut toutefois être conscient qu'on ne peut pas totalement exclure une erreur se produisant après le `isOnline()`, qui pourrait faire planter le programme. La seule manière de l'éviter est d'implémenter une des deux techniques de gestion des erreurs décrites ci-dessous.

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

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

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

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

9. Utilisation du Yocto-Color 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 la plus simple possible depuis le C++. La bibliothèque vous est fournie bien entendu aussi sous forme binaire, de sorte à pouvoir la linker directement si vous le préférez.

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

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

9.1. Contrôle de la fonction ColorLed

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

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

[...]
String errmsg;
YColorLed *colorled;
```

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

² www.yoctopuce.com/FR/libraries.php

```
// On récupère l'objet représentant le module (ici connecté en local sur USB)
yRegisterHub("usb", errmsg);
colorled = yFindColorLed("YRGBLED1-123456.colorLed1");

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.h et yocto_colorled.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_colorled.h` est nécessaire pour gérer les modules contenant une LED RGB, comme le Yocto-Color.

yRegisterHub

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

yFindColorLed

La fonction `yFindColorLed`, permet de retrouver une LED RGB 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-Color avec le numéros de série `YRGBLED1-123456` que vous auriez appelé `"MonModule"` et dont vous auriez nommé la fonction `colorLed1` `"MaFonction"`, les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YColorLed *colorled = yFindColorLed("YRGBLED1-123456.colorLed1");
YColorLed *colorled = yFindColorLed("YRGBLED1-123456.MaFonction");
YColorLed *colorled = yFindColorLed("MonModule.colorLed1");
YColorLed *colorled = yFindColorLed("MonModule.MaFonction");
YColorLed *colorled = yFindColorLed("MaFonction");
```

`yFindColorLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la LED RGB.

isOnline

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

set_rgbColor

La méthode `set_rgbColor()` de l'objet renvoyé par `yFindColorLed` permet de régler la couleur de la led correspondante. La Couleur est donnée en coordonnées RGB codées sous forme d'un entier.

rgbMove

Il existe une méthode alternative pour changer de couleur. La méthode `rgbMove()` de l'objet renvoyé par `yFindColorLed` permet de lancer une transition entre la couleur courante et une autre couleur RGB donnée en paramètre. Cette fonction prend aussi en paramètre le temps accordé pour effectuer la transition.

Les méthodes `set_rgbColor()` et `rgbMove()` existent aussi en version HSL. En guise d'exercice, vous pourriez modifier l'exemple ci-dessous pour utiliser le système HSL. Vous pourriez ainsi observer les différences de comportement. En HSL pour le rouge est codé `0xFFFF80`, le vert est codé `0x55FF80` et du bleu est codé `0xA9FF80`.

Un exemple réel

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

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

```
#include "yocto_api.h"
#include "yocto_colorled.h"
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

static void usage(void)
{
    cout << "usage: demo <serial_number> [ color | rgb ]" << endl;
    cout << "      demo <logical_name> [ color | rgb ]" << endl;
    cout << "      demo any [ color | rgb ]          (use any discovered device)" << endl;
    cout << "Eg." << endl;
    cout << "      demo any FF1493 " << endl;
    cout << "      demo YRGBLED1-123456 red" << endl;
    u64 now = yGetTickCount();
    while (yGetTickCount() - now < 3000) {
        // wait 3 sec to show the message
    }
    exit(1);
}

int main(int argc, const char * argv[])
{
    string      errmsg;
    string      target;
    YColorLed   *led1;
    YColorLed   *led2;
    string      color_str;
    unsigned int color;

    if(argc < 3) {
        usage();
    }
    target      = (string) argv[1];
    color_str   = (string) argv[2];
    if (color_str == "red")
        color = 0xFF0000;
    else if (color_str == "green")
        color = 0x00FF00;
    else if (color_str == "blue")
        color = 0x0000FF;
    else
        color = (unsigned int)strtoul(color_str.c_str(), NULL, 16);

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

    if (target == "any") {
        led1 = yFirstColorLed();
        if (led1 == NULL) {
            cout << "No module connected (check USB cable)" << endl;
            return 1;
        }
        led2 = led1->nextColorLed();
    } else {
        led1 = yFindColorLed(target + ".colorLed1");
        led2 = yFindColorLed(target + ".colorLed2");
    }

    if (led1->isOnline()) {
```

```

    led1->set_rgbColor(color); // immediate switch
    led2->rgbMove(color, 1000); // smooth transition
} else {
    cout << "Module not connected (check identification and USB cable)" << endl;
}
yFreeAPI();

return 0;
}

```

9.2. Contrôle de la partie module

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

```

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

#include "yocto_api.h"

using namespace std;

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

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

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

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

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

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

```

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

Modifications des réglages du module

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

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

#include "yocto_api.h"

using namespace std;

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

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

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

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

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

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

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

Enumeration des modules

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

```
#include <iostream>

#include "yocto_api.h"

using namespace std;

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

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

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

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

9.3. Gestion des erreurs

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

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

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

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

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir

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

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

9.4. Intégration de la librairie Yoctopuce en C++

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

Intégration au format source

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

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

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

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

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

Intégration en librairie statique

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

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

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

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

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

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

Intégration en librairie dynamique

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

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

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

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

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

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


10. Utilisation du Yocto-Color 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.

10.1. Contrôle de la fonction ColorLed

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

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

static void usage(void)
{
    NSLog(@"usage: demo <serial_number> [ color | rgb ]");
    NSLog(@"          demo <logical_name> [ color | rgb ]");
    NSLog(@"          demo any [ color | rgb ]           (use any discovered device)");
    NSLog(@"Eg.");
    NSLog(@"    demo any FF1493 ");
}
```

¹ www.yoctopuce.com/FR/libraries.php

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

```

NSLog(@"    demo YRGBLED1-123456 red");
exit(1);
}

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

    if(argc < 3) {
        usage();
    }

    @autoreleasepool {
        // Setup the API to use local USB devices
        if([YAPI RegisterHub:@"usb": &error] != YAPI_SUCCESS) {
            NSLog(@"RegisterHub error: %@", [error localizedDescription]);
            return 1;
        }
        NSString *target    = [NSString stringWithUTF8String:argv[1]];
        NSString *color_str = [NSString stringWithUTF8String:argv[2]];
        if ([target isEqualToString:@"any"]) {
            YColorLed *colorLed = [YColorLed FirstColorLed];
            if (colorLed == NULL) {
                NSLog(@"No Yocto-Color connected (check USB cable)");
                return 1;
            }
            target = [[colorLed module] serialNumber];
        }

        YColorLed *led1 = [YColorLed FindColorLed:[target stringByAppendingString:
            @".colorLed1"]];
        YColorLed *led2 = [YColorLed FindColorLed:[target stringByAppendingString:
            @".colorLed2"]];

        unsigned color;
        if ([color_str isEqualToString:@"red"])
            color = 0xFF0000;
        else if ([color_str isEqualToString:@"green"])
            color = 0x00FF00;
        else if ([color_str isEqualToString:@"blue"])
            color = 0x0000FF;
        else
            color = (int)strtoul([color_str UTF8String], NULL, 16);

        if ([led1 isOnline]) {
            [led1 set_rgbColor:color]; // immediate switch
            [led2 rgbMove:color:1000]; // smooth transition
        } else {
            NSLog(@"Module not connected (check identification and USB cable)\n");
        }
        [YAPI FreeAPI];
    }

    return 0;
}

```

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

yocto_api.h et yocto_colorled.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_colorled.h` est nécessaire pour gérer les modules contenant une LED RGB, comme le Yocto-Color.

[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.

[ColorLed FindColorLed]

La fonction [ColorLed FindColorLed], permet de retrouver une LED RGB 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-Color avec le numéros de série *YRGBLED1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *colorLed1* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YColorLed *colorled = [YColorLed FindColorLed:@"YRGBLED1-123456.colorLed1"];
YColorLed *colorled = [YColorLed FindColorLed:@"YRGBLED1-123456.MaFonction"];
YColorLed *colorled = [YColorLed FindColorLed:@"MonModule.colorLed1"];
YColorLed *colorled = [YColorLed FindColorLed:@"MonModule.MaFonction"];
YColorLed *colorled = [YColorLed FindColorLed:@"MaFonction"];
```

[YColorLed FindColorLed] renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la LED RGB.

isOnline

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

set_rgbColor

La méthode `set_rgbColor()` de l'objet renvoyé par `YColorLed.FindColorLed` permet de régler la couleur de la led correspondante. La Couleur est donnée en coordonnées RGB codées sous forme d'un entier.

rgbMove

Il existe une méthode alternative pour changer de couleur. La méthode `rgbMove()` de l'objet renvoyé par `YColorLed.FindColorLed` permet de lancer une transition entre la couleur courante et une autre couleur RGB donnée en paramètre. Cette fonction prend aussi en paramètre le temps accordé pour effectuer la transition.

Les méthodes `set_rgbColor()` et `rgbMove()` existent aussi en version HSL. En guise d'exercice, vous pourriez modifier l'exemple ci-dessous pour utiliser le système HSL. Vous pourriez ainsi observer les différences de comportement. En HSL pour le rouge est codé `0xFFFF80`, le vert est codé `0x55FF80` et du bleu est codé `0xA9FF80`.

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.

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

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

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

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

```

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

```

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

Modifications des réglages du module

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

```

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

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

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

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

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

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

```

```

YModule *module = [YModule FindModule:serial_or_name];

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

```

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

Enumeration des modules

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

```

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

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

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

        NSLog(@"Device list:\n");

        YModule *module = [YModule FirstModule];
        while (module != nil) {
            NSLog(@"%@ %@", module.serialNumber, module.productName);
            module = [module nextModule];
        }
        [YAPI FreeAPI];
    }
    return 0;
}

```

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

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

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

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

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

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

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

11. Utilisation du Yocto-Color en VisualBasic .NET

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

11.1. 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.

11.2. 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.

11.3. Contrôle de la fonction ColorLed

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

```
[...]
Dim errmsg As String
Dim colorled As YColorLed

REM On récupère l'objet représentant le module (ici connecté en local sur USB)
yRegisterHub("usb", errmsg)
colorled = yFindColorLed("YRGBLED1-123456.colorLed1")

REM Pour gérer le hot-plug, on vérifie que le module est là
If (colorled.isOnline()) Then
    REM Utiliser colorled.set_rgbColor(), ...
End If
```

Voyons maintenant en détail ce que font ces quelques lignes.

yRegisterHub

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

yFindColorLed

La fonction `yFindColorLed`, permet de retrouver une LED RGB 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-Color avec le numéros de série `YRGBLED1-123456` que vous auriez appelé `"MonModule"` et dont vous auriez nommé la fonction `colorLed1` `"MaFonction"`, les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
colorled = yFindColorLed("YRGBLED1-123456.colorLed1")
colorled = yFindColorLed("YRGBLED1-123456.MaFonction")
colorled = yFindColorLed("MonModule.colorLed1")
colorled = yFindColorLed("MonModule.MaFonction")
colorled = yFindColorLed("MaFonction")
```

`yFindColorLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la LED RGB.

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

isOnline

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

set_rgbColor

La méthode `set_rgbColor()` de l'objet renvoyé par `yFindColorLed` permet de régler la couleur de la led correspondante. La Couleur est donnée en coordonnées RGB codées sous forme d'un entier.

rgbMove

Il existe une méthode alternative pour changer de couleur. La méthode `rgbMove()` de l'objet renvoyé par `yFindColorLed` permet de lancer une transition entre la couleur courante et une autre couleur RGB donnée en paramètre. Cette fonction prend aussi en paramètre le temps accordé pour effectuer la transition.

Les méthodes `set_rgbColor()` et `rgbMove()` existent aussi en version HSL. En guise d'exercice, vous pourriez modifier l'exemple ci-dessous pour utiliser le système HSL. Vous pourriez ainsi observer les différences de comportement. En HSL pour le rouge est codé `0xFFFF80`, le vert est codé `0x55FF80` et du bleu est codé `0xA9FF80`.

Un exemple réel

Lancez Microsoft VisualBasic et ouvrez le projet exemple correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-Color** de la librairie Yoctopuce.

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

```
Module Module1

    Private Sub Usage()
        Dim execname = System.AppDomain.CurrentDomain.FriendlyName
        Console.WriteLine("Usage:")
        Console.WriteLine(execname + " <serial_number> [ color | rgb ]")
        Console.WriteLine(execname + " <logical_name> [ color | rgb ]")
        Console.WriteLine(execname + " any [ color | rgb ]")
        Console.WriteLine("Eg.")
        Console.WriteLine(execname + " any FF1493 ")
        Console.WriteLine(execname + " YRGBLED1-123456 red")
        System.Threading.Thread.Sleep(2500)
    End Sub

    Sub Main()
        Dim argv() As String = System.Environment.GetCommandLineArgs()
        Dim errmsg As String = ""
        Dim target As String
        Dim led1 As YColorLed
        Dim led2 As YColorLed
        Dim color_str As String
        Dim color As Integer

        If argv.Length < 3 Then Usage()

        target = argv(1)
        color_str = argv(2).ToUpper()

        If (color_str = "RED") Then
            color = &HFF0000
        ElseIf (color_str = "GREEN") Then
            color = &HFF00
        ElseIf (color_str = "BLUE") Then
            color = &HFF
        Else
            color = CInt(Val("&H" + color_str))
        End If

        REM Setup the API to use local USB devices
        If (yRegisterHub("usb", errmsg) <> YAPI_SUCCESS) Then
```

```

        Console.WriteLine("RegisterHub error: " + errormsg)
    End
End If

If target = "any" Then
    led1 = yFirstColorLed()
    If led1 Is Nothing Then
        Console.WriteLine("No module connected (check USB cable) ")
    End
    End If
    led2 = led1.nextColorLed()
Else
    led1 = yFindColorLed(target + ".colorLed1")
    led2 = yFindColorLed(target + ".colorLed2")
End If

If (led1.isOnline()) Then
    led1.set_rgbColor(color) REM immediate switch
    led2.rgbMove(color, 1000) REM smooth transition

Else
    Console.WriteLine("Module not connected (check identification and USB cable)")
End If
yFreeAPI()
End Sub

End Module

```

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.

```

Imports System.IO
Imports System.Environment

Module Module1

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

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

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

    If argv.Length < 2 Then usage()

    m = yFindModule(argv(1)) REM use serial or logical name
    If (m.isOnline()) Then
        If argv.Length > 2 Then
            If argv(2) = "ON" Then m.set_beacon(Y_BEACON_ON)
            If argv(2) = "OFF" Then m.set_beacon(Y_BEACON_OFF)
        End If
        Console.WriteLine("serial:      " + m.get_serialNumber())
        Console.WriteLine("logical name: " + m.get_logicalName())
        Console.WriteLine("luminosity:   " + Str(m.get_luminosity()))
        Console.WriteLine("beacon:      ")
        If (m.get_beacon() = Y_BEACON_ON) Then
            Console.WriteLine("ON")
        Else
            Console.WriteLine("OFF")
        End If
        Console.WriteLine("upTime:      " + Str(m.get_upTime() / 1000) + " sec")
    End If
End Sub

```

```

        Console.WriteLine("USB current: " + Str(m.get_usbCurrent()) + " mA")
        Console.WriteLine("Logs:")
        Console.WriteLine(m.get_lastLogs())
    Else
        Console.WriteLine(argv(1) + " not connected (check identification and USB cable)")
    End If
    yFreeAPI()
End Sub

End Module

```

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

Modifications des réglages du module

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

```

Module Module1

    Sub usage()

        Console.WriteLine("usage: demo <serial or logical name> <new logical name>")
    End Sub

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

        If (argv.Length <> 3) Then usage()

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

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

    End Sub

End Module

```

Attention, le nombre de cycles d'écriture de la mémoire non volatile du module est limité. Passé cette limite plus rien ne garantit que la sauvegarde des réglages se passera correctement. Cette limite,

liée à la technologie employée par le micro-processeur du module se situe aux alentours de 100000 cycles. Pour résumer vous ne pouvez employer la fonction `saveToFlash()` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

Enumeration des modules

Obtenir la liste des modules connectés se fait à l'aide de la fonction `yFirstModule()` qui renvoie le premier module trouvé, il suffit ensuite d'appeler la fonction `nextModule()` de cet objet pour trouver les modules suivants, et ce tant que la réponse n'est pas un `Nothing`. Ci-dessous un petit exemple listant les modules connectés

```
Module Module1

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

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

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

End Module
```

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

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

12. Utilisation du Yocto-Color 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.

12.1. 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.

12.2. 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**.

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.

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++

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

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

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

12.3. Contrôle de la fonction ColorLed

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

```
[...]
string errmsg = "";
YColorLed colorled;

// On récupère l'objet représentant le module (ici connecté en local sur USB)
YAPI.RegisterHub("usb", errmsg);
colorled = YColorLed.FindColorLed("YRGBLED1-123456.colorLed1");

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

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.

YColorLed.FindColorLed

La fonction `YColorLed.FindColorLed`, permet de retrouver une LED RGB 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-Color avec le numéros de série `YRGBLED1-123456` que vous auriez appelé `"MonModule"` et dont vous auriez nommé la fonction `colorLed1` `"MaFonction"`, les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
colorled = YColorLed.FindColorLed("YRGBLED1-123456.colorLed1");
colorled = YColorLed.FindColorLed("YRGBLED1-123456.MaFonction");
colorled = YColorLed.FindColorLed("MonModule.colorLed1");
colorled = YColorLed.FindColorLed("MonModule.MaFonction");
colorled = YColorLed.FindColorLed("MaFonction");
```

`YColorLed.FindColorLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la LED RGB.

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

isOnline

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

set_rgbColor

La méthode `set_rgbColor()` de l'objet renvoyé par `YColorLed.FindColorLed` permet de régler la couleur de la led correspondante. La Couleur est donnée en coordonnées RGB codées sous forme d'un entier.

rgbMove

Il existe une méthode alternative pour changer de couleur. La méthode `rgbMove()` de l'objet renvoyé par `YColorLed.FindColorLed` permet de lancer une transition entre la couleur courante et une autre couleur RGB donnée en paramètre. Cette fonction prend aussi en paramètre le temps accordé pour effectuer la transition.

Les méthodes `set_rgbColor()` et `rgbMove()` existent aussi en version HSL. En guise d'exercice, vous pourriez modifier l'exemple ci-dessous pour utiliser le système HSL. Vous pourriez ainsi observer les différences de comportement. En HSL pour le rouge est codé `0xFFFF80`, le vert est codé `0x55FF80` et du bleu est codé `0xA9FF80`.

Un exemple réel

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

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

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

namespace ConsoleApplication1
{
    class Program
    {
        static void usage()
        {
            string execname = System.AppDomain.CurrentDomain.FriendlyName;
            Console.WriteLine(execname + " <serial_number> [ color | rgb ]");
            Console.WriteLine(execname + " <logical_name> [ color | rgb ]");
            Console.WriteLine(execname + " any [ color | rgb ]");
            Console.WriteLine("Eg.");
            Console.WriteLine(execname + " any FF1493 ");
            Console.WriteLine(execname + " YRGBLED1-123456 red");
            System.Threading.Thread.Sleep(2500);
            Environment.Exit(0);
        }

        static void Main(string[] args)
        {
            string errmsg = "";
            string target;
            YColorLed led1;
            YColorLed led2;
            string color_str;
            int color;

            if (args.Length < 2) usage();

            target = args[0].ToUpper();
            color_str = args[1].ToUpper();

            if (color_str == "RED") color = 0xFF0000;
            else if (color_str == "GREEN") color = 0x00FF00;
            else if (color_str == "BLUE") color = 0x0000FF;
            else color = Convert.ToInt32("0x" + color_str, 16);
        }
    }
}
```

```

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

if (target == "ANY") {
    led1 = YColorLed.FirstColorLed();
    if (led1 == null) {
        Console.WriteLine("No module connected (check USB cable) ");
        Environment.Exit(0);
    }

    led2 = led1.nextColorLed();
} else {
    led1 = YColorLed.FindColorLed(target + ".colorLed1");
    led2 = YColorLed.FindColorLed(target + ".colorLed2");
}

if (led1.isOnline()) {
    led1.set_rgbColor(color); // immediate switch
    led2.rgbMove(color, 1000); // smooth transition
} else {
    Console.WriteLine("Module not connected (check identification and USB cable)");
}
YAPI.FreeAPI();
}
}
}

```

12.4. Contrôle de la partie module

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

```

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

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

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

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

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

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

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

```

```

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

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

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

```

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

Modifications des réglages du module

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

```

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

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

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

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

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

```

```

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

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

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

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

```

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

Enumeration des modules

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

```

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

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

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

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

```

12.5. Gestion des erreurs

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

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

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

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

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

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

13. Utilisation du Yocto-Color en Delphi

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

Les librairies pour Delphi sont fournies non pas sous forme de composants VCL, mais directement sous forme de fichiers source. Ces fichiers sont compatibles avec la plupart des versions de Delphi².

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

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

13.1. Préparation

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

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

13.2. Contrôle de la fonction ColorLed

Lancez votre environnement Delphi, copiez la DLL *yapi.dll* dans un répertoire et créez une nouvelle application console dans ce même répertoire, et copiez-coller le code ci dessous.

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

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

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

³ www.yoctopuce.com/FR/libraries.php

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

```

SysUtils,
yocto_api,
yocto_colorled;

Procedure Usage();
var
  exe : string;

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

procedure setcolor(led1,led2:TYColorLed; color:integer);
begin
  if (led1.isOnline()) then
  begin
    led1.set_rgbColor(color); // immediate switch
    led2.rgbMove(color,1000); // smooth transition
  end
  else Writeln('Module not connected (check identification and USB cable)');
end;

var
  c      : char;
  led1,led2 : TYColorLed;
  m      : TYmodule;
  errmsg  : string;
  serial  : string;

begin

  if (paramcount<1) then usage();

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

  // first one of the two RGB leds
  if paramstr(1)='any' then
  begin
    led1 := yFirstColorLed();
    if led1=nil then
    begin
      writeln('No module connected (check USB cable)');
      halt;
    end
  end
  else
    led1:= YFindColorLed(paramstr(1)+'.colorled1');

  // make sure it is online
  if not(led1.isOnline()) then
  begin
    writeln('No module connected (check USB cable)');
    halt;
  end;

  // lets find the device serial to find the second led
  m := led1.Get_module();
  serial := m.get_serialNumber();
  led1 := yFindColorLed(serial+'.colorLed1');
  led2 := yFindColorLed(serial+'.colorLed2');

  Writeln('r: set to red');
  Writeln('g: set to green');
  Writeln('b: set to blue');
  Writeln('x: exit');
  repeat
    read(c);
    case c of
      'r' : setcolor(led1,led2,$FF0000);

```



```

    'g' : setcolor(led1,led2,$00FF00);
    'b' : setcolor(led1,led2,$0000FF);
  end;
until c='x';
yFreeAPI();

end.

```

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

yocto_api et yocto_colorled

Ces deux unités permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api` doit toujours être utilisé, `yocto_colorled` est nécessaire pour gérer les modules contenant une LED RGB, comme le Yocto-Color.

yRegisterHub

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

yFindColorLed

La fonction `yFindColorLed`, permet de retrouver une LED RGB 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-Color avec le numéros de série `YRGBLED1-123456` que vous auriez appelé *"MonModule"* et dont vous auriez nommé la fonction `colorLed1` *"MaFonction"*, les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```

colorled := yFindColorLed("YRGBLED1-123456.colorLed1");
colorled := yFindColorLed("YRGBLED1-123456.MaFonction");
colorled := yFindColorLed("MonModule.colorLed1");
colorled := yFindColorLed("MonModule.MaFonction");
colorled := yFindColorLed("MaFonction");

```

`yFindColorLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la LED RGB.

isOnline

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

set_rgbColor

La méthode `set_rgbColor()` de l'objet renvoyé par `yFindColorLed` permet de régler la couleur de la led correspondante. La Couleur est donnée en coordonnées RGB codées sous forme d'un entier.

rgbMove

Il existe une méthode alternative pour changer de couleur. La méthode `rgbMove()` de l'objet renvoyé par `yFindColorLed` permet de lancer une transition entre la couleur courante et une autre couleur RGB donnée en paramètre. Cette fonction prend aussi en paramètre le temps accordé pour effectuer la transition.

Les méthodes `set_rgbColor()` et `rgbMove()` existent aussi en version HSL. En guise d'exercice, vous pourriez modifier l'exemple ci-dessous pour utiliser le système HSL. Vous pourriez ainsi observer les différences de comportement. En HSL pour le rouge est codé `0xFFFF80`, le vert est codé `0x55FF80` et du bleu est codé `0xA9FF80`.

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.

```

program modulecontrol;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  yocto_api;

const
  serial = 'YRGBLED1-123456'; // use serial number or logical name

procedure refresh(module:Tymodule) ;
begin
  if (module.isOnline()) then
  begin
    Writeln('');
    Writeln('Serial      : ' + module.get_serialNumber());
    Writeln('Logical name : ' + module.get_logicalName());
    Writeln('Luminosity   : ' + intToStr(module.get_luminosity()));
    Write('Beacon     :');
    if (module.get_beacon()=Y_BEACON_ON) then Writeln('on')
    else Writeln('off');
    Writeln('uptime      : ' + intToStr(module.get_upTime() div 1000)+'s');
    Writeln('USB current  : ' + intToStr(module.get_usbCurrent())+'mA');
    Writeln('Logs        : ');
    Writeln(module.get_lastlogs());
    Writeln('');
    Writeln('r : refresh / b:beacon ON / space : beacon off');
  end
  else Writeln('Module not connected (check identification and USB cable)');
end;

procedure beacon(module:Tymodule;state:integer);
begin
  module.set_beacon(state);
  refresh(module);
end;

var
  module : TYModule;
  c      : char;
  errmsg : string;

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

  module := yFindModule(serial);
  refresh(module);

  repeat
    read(c);
    case c of
      'r': refresh(module);
      'b': beacon(module,Y_BEACON_ON);
      ' ': beacon(module,Y_BEACON_OFF);
    end;
  until c = 'x';
  yFreeAPI();
end.

```

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

Modifications des réglages du module

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

```
program savesettings;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  yocto_api;

const
  serial = 'YRGBLED1-123456'; // 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.

```

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. La seule manière de l'éviter est d'implémenter une des deux techniques de gestion des erreurs décrites ci-dessous.

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

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

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

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

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

14. Utilisation du Yocto-Color 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.

14.1. 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.

14.2. Librairie dynamique

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

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

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

¹ <http://www.python.org/download/>

² www.yoctopuce.com/FR/libraries.php

14.3. Contrôle de la fonction ColorLed

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

```
[...]

errmsg=YRefParam()
#On récupère l'objet représentant le module (ici connecté en local sur USB)
YAPI.RegisterHub("usb",errmsg)
colorled = YColorLed.FindColorLed("YRGBLED1-123456.colorLed1")

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

[...]
```

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.

YColorLed.FindColorLed

La fonction `YColorLed.FindColorLed`, permet de retrouver une LED RGB 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-Color avec le numéros de série *YRGBLED1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *colorLed1* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
colorled = YColorLed.FindColorLed("YRGBLED1-123456.colorLed1")
colorled = YColorLed.FindColorLed("YRGBLED1-123456.MaFonction")
colorled = YColorLed.FindColorLed("MonModule.colorLed1")
colorled = YColorLed.FindColorLed("MonModule.MaFonction")
colorled = YColorLed.FindColorLed("MaFonction")
```

`YColorLed.FindColorLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la LED RGB.

isOnline

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

set_rgbColor

La méthode `set_rgbColor()` de l'objet renvoyé par `YColorLed.FindColorLed` permet de régler la couleur de la led correspondante. La Couleur est donnée en coordonnées RGB codées sous forme d'un entier.

rgbMove

Il existe une méthode alternative pour changer de couleur. La méthode `rgbMove()` de l'objet renvoyé par `YColorLed.FindColorLed` permet de lancer une transition entre la couleur courante et une autre couleur RGB donnée en paramètre. Cette fonction prend aussi en paramètre le temps accordé pour effectuer la transition.

Les méthodes `set_rgbColor()` et `rgbMove()` existent aussi en version HSL. En guise d'exercice, vous pourriez modifier l'exemple ci-dessous pour utiliser le système HSL. Vous pourriez ainsi observer les différences de comportement. En HSL pour le rouge est codé `0xFFFF80`, le vert est codé `0x55FF80` et du bleu est codé `0xA9FF80`.

Un exemple réel

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

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

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

from yocto_api import *
from yocto_colorled import *

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

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

def setcolor(led1, led2, color):
    if led1.isOnline():
        led1.set_rgbColor(color) # immediate switch
        led2.rgbMove(color, 1000) # smooth transition
    else:
        print('Module not connected (check identification and USB cable)')

errmsg = YRefParam()

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

target = sys.argv[1]

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

if target == 'any':
    # retrieve any RGB led
    led = YColorLed.FirstColorLed()
    if led is None:
        die('No module connected')
else:
    led = YColorLed.FindColorLed(target + '.colorLed1')

# we need to retrieve the second led from the device
if led.isOnline():
    m = led.get_module()
    led1 = YColorLed.FindColorLed(m.get_serialNumber() + '.colorLed1')
    led2 = YColorLed.FindColorLed(m.get_serialNumber() + '.colorLed2')
else:
    die('device not connected')

print('r: set to red')
print('g: set to green')
print('b: set to blue')
print('x: exit')

try:
```

```

    input = raw_input # python 2.x fix
except:
    pass

c = input("command:")

while c != 'x':
    if c == 'r':
        setcolor(led1, led2, 0xFF0000)
    elif c == 'g':
        setcolor(led1, led2, 0x00FF00)
    elif c == 'b':
        setcolor(led1, led2, 0x0000FF)
    c = input("command:")
YAPI.FreeAPI()

```

14.4. Contrôle de la partie module

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

```

#!/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()
```

14.5. Gestion des erreurs

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

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

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

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

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

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

15. Utilisation du Yocto-Color 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.

15.1. 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.

15.2. Contrôle de la fonction ColorLed

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

```
[...]  
  
// On récupère l'objet représentant le module (ici connecté en local sur USB)  
YAPI.RegisterHub("127.0.0.1");  
colorled = YColorLed.FindColorLed("YRGBLED1-123456.colorLed1");  
  
// Pour gérer le hot-plug, on vérifie que le module est là  
if (colorled.isOnline())  
{  
    // Utiliser colorled.set_rgbColor()
```

¹ www.yoctopuce.com/FR/libraries.php

² www.yoctopuce.com/FR/virtualhub.php

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

Voyons maintenant en détail ce que font ces quelques lignes.

YAPI.RegisterHub

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

YColorLed.FindColorLed

La fonction `YColorLed.FindColorLed`, permet de retrouver une LED RGB 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-Color avec le numéro de série `YRGBLED1-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `colorLed1` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
colorled = YColorLed.FindColorLed("YRGBLED1-123456.colorLed1")
colorled = YColorLed.FindColorLed("YRGBLED1-123456.MaFonction")
colorled = YColorLed.FindColorLed("MonModule.colorLed1")
colorled = YColorLed.FindColorLed("MonModule.MaFonction")
colorled = YColorLed.FindColorLed("MaFonction")
```

`YColorLed.FindColorLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la LED RGB.

isOnline

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

set_rgbColor

La méthode `set_rgbColor()` de l'objet renvoyé par `YColorLed.FindColorLed` permet de régler la couleur de la led correspondante. La Couleur est donnée en coordonnées RGB codées sous forme d'un entier.

rgbMove

Il existe une méthode alternative pour changer de couleur. La méthode `rgbMove()` de l'objet renvoyé par `YColorLed.FindColorLed` permet de lancer une transition entre la couleur courante et une autre couleur RGB donnée en paramètre. Cette fonction prend aussi en paramètre le temps accordé pour effectuer la transition.

Les méthodes `set_rgbColor()` et `rgbMove()` existent aussi en version HSL. En guise d'exercice, vous pourriez modifier l'exemple ci-dessous pour utiliser le système HSL. Vous pourriez ainsi observer les différences de comportement. En HSL pour le rouge est codé `0xFFFF80`, le vert est codé `0x55FF80` et du bleu est codé `0xA9FF80`.

Un exemple réel

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

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

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

public class Demo {
```

```

public static void main(String[] args) {
    try {
        // setup the API to use local VirtualHub
        YAPI.RegisterHub("127.0.0.1");
    } catch (YAPI_Exception ex) {
        System.out.println("Cannot contact VirtualHub on 127.0.0.1 (" +
ex.getLocalizedMessage() + ")");
        System.out.println("Ensure that the VirtualHub application is running");
        System.exit(1);
    }
    YColorLed led1, led2;
    if (args.length > 0) {
        led1 = YColorLed.FindColorLed(args[0] + ".colorLed1");
        led2 = YColorLed.FindColorLed(args[0] + ".colorLed2");
    } else {
        led1 = YColorLed.FirstColorLed();
        if (led1 == null) {
            System.out.println("No module connected (check USB cable)");
            System.exit(1);
        }
        led2 = led1.nextColorLed();
    }

    int all_colors[] = {0xff0000, 0x00ff00, 0x0000ff};
    for(int color : all_colors) {
        try {
            System.out.println(String.format("Change color to 0x%06x", color));
            led1.set_rgbColor(color); //led2.rgbMove(color, 1000); // smooth transition
            led2.rgbMove(color, 1000); // smooth transition
            YAPI.Sleep(1000);
        } catch (YAPI_Exception ex) {
            System.out.println("Module not connected (check identification and USB
cable)");
            break;
        }
    }

    YAPI.FreeAPI();
}

```

15.3. Contrôle de la partie module

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

```

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 au chapitre API.

Modifications des réglages du module

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

```

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();
    }
}

```

15.4. Gestion des erreurs

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

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

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.

16. Utilisation du Yocto-Color 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.

16.1. 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.

16.2. 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.

16.3. 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.

16.4. 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>
```

16.5. Contrôle de la fonction ColorLed

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

```
[...]

// On récupère l'objet représentant le module (ici connecté en local sur USB)
YAPI.EnableUSBHost(this);
YAPI.RegisterHub("usb");
colorled = YColorLed.FindColorLed("YRGBLED1-123456.colorLed1");

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

[...]
```

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.

YColorLed.FindColorLed

La fonction `YColorLed.FindColorLed` permet de retrouver une LED RGB 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-Color avec le numéro de série `YRGBLED1-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `colorLed1` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
colorled = YColorLed.FindColorLed("YRGBLED1-123456.colorLed1")
colorled = YColorLed.FindColorLed("YRGBLED1-123456.MaFonction")
colorled = YColorLed.FindColorLed("MonModule.colorLed1")
colorled = YColorLed.FindColorLed("MonModule.MaFonction")
colorled = YColorLed.FindColorLed("MaFonction")
```

`YColorLed.FindColorLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la LED RGB.

isOnline

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

set_rgbColor

La méthode `set_rgbColor()` de l'objet renvoyé par `YColorLed.FindColorLed` permet de régler la couleur de la led correspondante. La Couleur est donnée en coordonnées RGB codées sous forme d'un entier.

rgbMove

Il existe une méthode alternative pour changer de couleur. La méthode `rgbMove()` de l'objet renvoyé par `YColorLed.FindColorLed` permet de lancer une transition entre la couleur courante et une autre couleur RGB donnée en paramètre. Cette fonction prend aussi en paramètre le temps accordé pour effectuer la transition.

Les méthodes `set_rgbColor()` et `rgbMove()` existent aussi en version HSL. En guise d'exercice, vous pourriez modifier l'exemple ci-dessous pour utiliser le système HSL. Vous pourriez ainsi observer les différences de comportement. En HSL pour le rouge est codé `0xFFFF80`, le vert est codé `0x55FF80` et du bleu est codé `0xA9FF80`.

Un exemple réel

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

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

```
package com.yoctopuce.doc_exemples;

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

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

public class GettingStarted_Yocto_Color extends Activity implements OnItemClickListener
{
    private YColorLed color = null;
    private ArrayAdapter<String> aa;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.gettingstarted_yocto_color);
        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();
        aa.clear();
        try {
            YAPI.EnableUSBHost(this);
            YAPI.RegisterHub("usb");
            YAPI.UpdateDeviceList();
            YColorLed c = YColorLed.FirstColorLed();
            while (c != null) {
                String hwid = c.get_hardwareId();
                aa.add(hwid);
                c = c.nextColorLed();
            }
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
        aa.notifyDataSetChanged();
    }
}
```

```

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

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

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

/** Called when the user touches the button State A */
public void updateColor(View view)
{
    if (color == null)
        return;

    SeekBar red_bar = (SeekBar) findViewById(R.id.seekBarRed);
    int red = red_bar.getProgress() * 255 / red_bar.getMax();
    SeekBar green_bar = (SeekBar) findViewById(R.id.seekBarGreen);
    int green = green_bar.getProgress() * 255 / green_bar.getMax();
    SeekBar blue_bar = (SeekBar) findViewById(R.id.seekBarBlue);
    int blue = blue_bar.getProgress() * 255 / blue_bar.getMax();
    int newcolor = (red << 16) + (green << 8) + blue;
    switch (view.getId()) {
    case R.id.moveColorButton:
        try {
            color.rgbMove(newcolor, 1000);
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
        break;
    case R.id.setColorButton:
        try {
            color.set_rgbColor(newcolor);
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
        break;
    }
}
}

```

16.6. Contrôle de la partie module

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

```

package com.yoctopuce.doc_examples;

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

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

```



```

import com.yoctopuce.YoctoAPI.YModule;

public class ModuleControl extends Activity implements OnItemSelectedListener
{

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

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

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

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

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

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

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

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

```

```

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

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

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

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

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

```

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

Modifications des réglages du module

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

```

package com.yoctopuce.doc_examples;

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

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

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

    @Override

```

```

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

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

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

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

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

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

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

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

    EditText edit = (EditText) findViewById(R.id.newname);
    String newname = edit.getText().toString();
    try {
        if (!YAPI.CheckLogicalName(newname)) {
            Toast.makeText(getApplicationContext(), "Invalid name (" + newname + ")",
            Toast.LENGTH_LONG).show();

```

```

        return;
    }
    module.set_logicalName(newname);
    module.saveToFlash(); // do not forget this
    edit.setText("");
} catch (YAPI_Exception ex) {
    ex.printStackTrace();
}
DisplayModuleInfo();
}
}

```

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

Enumeration des modules

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

```

package com.yoctopuce.doc_examples;

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

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

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

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

        try {
            YAPI.UpdateDeviceList();
            YModule module = YModule.FirstModule();
            while (module != null) {
                String line = module.get_serialNumber() + " (" + module.get_productName() +
                ")";

                TextView tx = new TextView(this);
                tx.setText(line);
                tx.setTextSize(TypedValue.COMPLEX_UNIT_SP, 20);
                layout.addView(tx);
                module = module.nextModule();
            }
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    protected void onStart()

```

```

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

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

```

16.7. 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.

17. Programmation avancée

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

17.1. 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.

18. Mise à jour du firmware

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

18.1. Le VirtualHub ou le YoctoHub

Il est possible de mettre à jour un module directement depuis l'interface web du VirtualHub ou du YoctoHub. Il suffit d'accéder à la fenêtre de configuration du module que à mettre à jour et de cliquer sur le bouton "upgrade". Le VirtualHub démarre un assistant qui vous guidera durant la procédure de mise à jour.

Si pour une raison ou une autre, la mise à jour venait à échouer et que le module de fonctionnait plus, débranchez puis rebranchez le module en maintenant sur le Yocto-bouton appuyé. Le module va démarrer en mode "mise à jour" et sera listé en dessous des modules connectés.

18.2. La librairie ligne de commandes

Tous les outils en lignes de commandes ont la possibilité de mettre à jour les modules Yoctopuce grâce à la commande `downloadAndUpdate`. Le mécanisme de sélection des modules fonctionne comme pour une commande traditionnelle. La [cible] est le nom du module qui va être mis à jour. Vous pouvez aussi utiliser les alias "any" ou "all", ou encore une liste de noms, séparés par des virgules, sans espace.

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

L'exemple suivant met à jour tous les modules Yoctopuce connectés en USB.

```
C:\>YModule all downloadAndUpdate
ok: Yocto-PowerRelay RELAYHI1-266C8(rev=15430) is up to date.
ok: 0 / 0 hubs in 0.000000s.
ok: 0 / 0 shields in 0.000000s.
ok: 1 / 1 devices in 0.130000s 0.130000s per device.
ok: All devices are now up to date.
C:\>
```

18.3. L'application Android Yocto-Firmware

Il est possible de mettre à jour le firmware de vos modules depuis votre téléphone ou tablette Android avec l'application [Yocto-Firmware](#). Cette application liste tous les modules Yoctopuce

branchés en USB et vérifie si un firmware plus récent est disponible sur www.yoctopuce.com. Si un firmware plus récent est disponible, il est possible de mettre à jour le module. L'application se charge de télécharger et d'installer le nouveau firmware en préservant les paramètres du module.

Attention, pendant la mise à jour du firmware, le module redémarre plusieurs fois. Android interprète le reboot d'un périphérique USB comme une déconnexion et reconnexion du périphérique USB, et demande à nouveau l'autorisation d'utiliser le port USB. L'utilisateur est obligé de cliquer sur **OK** pour que la procédure de mise à jour se termine correctement.

18.4. La librairie de programmation

Si vous avez besoin d'intégrer la mise à jour de firmware dans votre application, les librairies proposent une API pour mettre à jour vos modules.¹

Sauvegarder et restaurer les paramètres

La méthode `get_allSettings()` retourne un buffer binaire qui permet de sauvegarder les paramètres persistants d'un module. Cette fonction est très utile pour sauvegarder la configuration réseau d'un YoctoHub par exemple.

```
YWireless wireless = YWireless.FindWireless("reference");
YModule m = wireless.get_module();
byte[] default_config = m.get_allSettings();
saveFile("default.bin", default_config);
...
```

Ces paramètres peuvent être appliqués sur d'autres modules à l'aide de la méthode `set_allSettings()`.

```
byte[] default_config = loadFile("default.bin");
YModule m = YModule.FirstModule();
while (m != null) {
    if (m.get_productName() == "YoctoHub-Wireless") {
        m.set_allSettings(default_config);
    }
    m = m.next();
}
```

Chercher le bon firmware

La première étape pour mettre à jour un module Yoctopuce est de trouver quel firmware il faut utiliser, c'est le travail de la méthode `checkFirmware(path, onlynew)` de l'objet `YModule`. Cette méthode vérifie que le firmware passé en argument (`path`) est compatible avec le module. Si le paramètre `onlynew` est vrai, cette méthode vérifie si le firmware est plus récent que la version qui est actuellement utilisée par le module. Quand le fichier n'est pas compatible (ou si le fichier est plus vieux que la version installée), cette méthode retourne une chaîne vide. Si au contraire le fichier est valide, la méthode retourne le chemin d'accès d'un fichier.

Le code suivant vérifie si le fichier `c:\tmp\METEOMK1.17328.byn` est compatible avec le module stocké dans la variable `m`.

```
YModule m = YModule.FirstModule();
...
...
string path = "c:\\tmp\\METEOMK1.17328.byn";
string newfirm = m.checkFirmware(path, false);
if (newfirm != "") {
    Console.WriteLine("firmware " + newfirm + " is compatible");
}
...
```

¹ Les librairies JavaScript, Node.js et PHP ne permettent pas encore de mettre à jour les modules, mais ces fonctions seront disponibles dans un prochain build.

Il est possible de passer un répertoire en argument (au lieu d'un fichier). Dans ce cas la méthode va parcourir récursivement tous les fichiers du répertoire et retourner le firmware compatible le plus récent. Le code suivant vérifie s'il existe un firmware plus récent dans le répertoire `c:\tmp\`.

```
YModule m = YModule.FirstModule();
...
...
string path = "c:\\tmp";
string newfirm = m.checkFirmware(path, true);
if (newfirm != "") {
    Console.WriteLine("firmware " + newfirm + " is compatible and newer");
}
...
```

Il est aussi possible de passer la chaîne `"www.yoctopuce.com"` en argument pour vérifier s'il existe un firmware plus récent publié sur le site web de Yoctopuce. Dans ce cas, la méthode retournera l'URL du firmware. Vous pourrez soit utiliser cette URL pour télécharger le firmware sur votre disque, soit utiliser cette URL lors de la mise à jour du firmware (voir ci-dessous). Bien évidemment, cette possibilité ne fonctionne que si votre machine est reliée à Internet.

```
YModule m = YModule.FirstModule();
...
...
string url = m.checkFirmware("www.yoctopuce.com", true);
if (url != "") {
    Console.WriteLine("new firmware is available at " + url);
}
...
```

Mettre à jour le firmware

La mise à jour du firmware peut prendre plusieurs minutes, c'est pourquoi le processus de mise à jour est exécuté par la librairie en arrière plan et est contrôlé par le code utilisateur à l'aide de la classe `YFirmwareUpdate`.

Pour mettre à jour un module Yoctopuce, il faut obtenir une instance de la classe `YFirmwareUpdate` à l'aide de la méthode `updateFirmware` d'un objet `YModule`. Le seul paramètre de cette méthode est le *path* du firmware à installer. Cette méthode ne démarre pas immédiatement la mise à jour, mais retourne un objet `YFirmwareUpdate` configuré pour mettre à jour le module.

```
string newfirm = m.checkFirmware("www.yoctopuce.com", true);
....
YFirmwareUpdate fw_update = m.updateFirmware(newfirm);
```

La méthode `startUpdate()` démarre la mise à jour en arrière plan. Ce processus en arrière plan se charge automatiquement de:

1. sauvegarder des paramètres du module,
2. redémarrer le module en mode "mise à jour"
3. mettre à jour le firmware
4. démarrer le module avec la nouvelle version du firmware
5. restaurer les paramètres

Les méthodes `get_progress()` et `get_progressMessage()` permettent de suivre la progression de la mise à jour. `get_progress()` retourne la progression sous forme de pourcentage (100 = mise à jour terminée). `get_progressMessage()` retourne une chaîne de caractères décrivant l'opération en cours (effacement, écriture, reboot,...). Si la méthode `get_progress()` retourne une valeur négative, c'est que le processus de mise à jour a échoué. Dans ce cas la méthode `get_progressMessage()` retourne le message d'erreur.

Le code suivant démarre la mise à jour et affiche la progression sur la sortie standard.

```

YFirmwareUpdate fw_update = m.updateFirmware(newfirm);
....
int status = fw_update.startUpdate();
while (status < 100 && status >= 0) {
    int newstatus = fw_update.get_progress();
    if (newstatus != status) {
        Console.WriteLine(status + "% "
            + fw_update.get_progressMessage());
    }
    YAPI.Sleep(500, ref errmsg);
    status = newstatus;
}

if (status < 0) {
    Console.WriteLine("Firmware Update failed: "
        + fw_update.get_progressMessage());
} else {
    Console.WriteLine("Firmware Updated Successfully!");
}

```

Particularité d'Android

Il est possible de mettre à jour un firmware d'un module en utilisant la librairie Android. Mais pour les modules branchés en USB, Android va demander à l'utilisateur d'autoriser l'application à accéder au port USB.

Pendant la mise à jour du firmware, le module redémarre plusieurs fois. Android interprète le reboot d'un périphérique USB comme une déconnexion et reconnexion du port USB, et interdit tout accès USB tant que l'utilisateur n'a pas fermé le pop-up. L'utilisateur est obligé de cliquer sur *OK* pour que la procédure de mise à jour puisse continuer correctement. **Il n'est pas possible de mettre à jour un module branché en USB à un appareil Android sans que l'utilisateur ne soit obligé d'interagir avec l'appareil.**

18.5. Le mode "mise à jour"

Si vous désirez effacer tous les paramètres du module ou que votre module ne démarre plus correctement, il est possible d'installer un firmware depuis le mode "mise à jour".

Pour forcer le module à fonctionner dans le mode "mise à jour", débranchez-le, attendez quelques secondes, et rebranchez-le en maintenant le *Yocto-Bouton* appuyé. Cela a pour effet de faire démarrer le module en mode "mise à jour". Ce mode de fonctionnement est protégé contre les corruptions et est toujours accessible.

Dans ce mode, le module n'est plus détecté par les objets YModules. Pour obtenir la liste des modules connectés en mode "mise à jour", il faut utiliser la fonction `YAPI.GetAllBootLoaders()`. Cette fonction retourne un tableau de chaînes de caractères avec le numéro de série des modules en le mode "mise à jour".

```
List<string> allBootLoader = YAPI.GetAllBootLoaders();
```

La procédure de mise à jour est identique au cas standard (voir section précédente), mais il faut instancier manuellement l'objet `YFirmwareUpdate` au lieu d'appeler `module.updateFirmware()`. Le constructeur prend en argument trois paramètres: le numéro de série du module, le path du firmware à installer, et un tableau de bytes avec les paramètres à restaurer à la fin de la mise à jour (ou `null` pour restaurer les paramètres d'origine).

```

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

```

19. 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.

19.1. Ligne de commande

Le moyen le plus simple pour contrôler des modules Yoctopuce depuis un langage non supporté consiste à utiliser l'API en ligne de commande à travers des appels système. L'API en ligne de commande se présente en effet sous la forme d'un ensemble de petits exécutables qu'il est facile d'appeler et dont la sortie est facile à analyser. La plupart des langages de programmation permettant d'effectuer des appels système, cela permet de résoudre le problème en quelques lignes.

Cependant, si l'API en ligne de commande est la solution la plus facile, ce n'est pas la plus rapide ni la plus efficace. A chaque appel, l'exécutable devra initialiser sa propre API et faire l'inventaire des modules USB connectés. Il faut compter environ une seconde par appel.

19.2. Virtual Hub et HTTP GET

Le *Virtual Hub* est disponible pour presque toutes les plateformes actuelles, il sert généralement de passerelle pour permettre l'accès aux modules Yoctopuce depuis des langages qui interdisent l'accès direct aux couches matérielles d'un ordinateur (Javascript, PHP, Java...).

Il se trouve que le *Virtual Hub* est en fait un petit serveur Web qui est capable de router des requêtes HTTP vers les modules Yoctopuce. Ce qui signifie que si vous pouvez faire une requête HTTP depuis votre langage de programmation, vous pouvez contrôler des modules Yoctopuce, même si ce langage n'est pas officiellement supporté.

Interface REST

A bas niveau, les modules sont pilotés à l'aide d'une API REST. Ainsi pour contrôler un module, il suffit de faire les requêtes HTTP appropriées sur le *Virtual Hub*. Par défaut le port HTTP du *Virtual Hub* est 4444.

Un des gros avantages de cette technique est que les tests préliminaires sont très faciles à mettre en œuvre, il suffit d'un *Virtual Hub* et d'un simple browser Web. Ainsi, si vous copiez l'URL suivante dans votre browser favori, alors que le *Virtual Hub* est en train de tourner, vous obtiendrez la liste des modules présents.

```
http://127.0.0.1:4444/api/services/whitePages.txt
```

Remarquez que le résultat est présenté sous forme texte, mais en demandant *whitePages.xml* vous auriez obtenu le résultat en XML. De même, *whitePages.json* aurait permis d'obtenir le résultat en JSON. L'extension *html* vous permet même d'afficher une interface sommaire vous permettant de changer les valeurs en direct. Toute l'API REST est disponible dans ces différents formats.

Contrôle d'un module par l'interface REST

Chaque module Yoctopuce a sa propre interface REST disponible sous différentes formes. Imaginons un Yocto-Color avec le numéro de de série *YRGBLED1-12345* et le nom logique *monModule*. L'URL suivante permettra de connaître l'état du module.

```
http://127.0.0.1:4444/bySerial/YRGBLED1-12345/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/YRGBLED1-12345/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/YRGBLED1-12345/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 *colorLed1*, il suffit de construire l'URL suivante.

```
http://127.0.0.1:4444/bySerial/YRGBLED1-12345/api/colorLed1.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/YRGBLED1-12345/api/colorLed1/logicalName
```

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

```
http://127.0.0.1:4444/bySerial/YRGBLED1-12345/api/colorLed1?logicalName=maFonction
```

Vous trouverez la liste des attributs disponibles pour votre Yocto-Color 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/YRGBLED1-12345/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/YRGBLED1-12345/dataLogger.json?id=colorLed1&utc=1389801080
```

19.3. Utilisation des bibliothèques dynamiques

L'API Yoctopuce bas niveau est disponible sous différents formats de bibliothèque dynamiques écrites en C, dont les sources sont disponibles avec l'API C++. Utiliser une de ces bibliothèques bas niveau vous permettra de vous passer du *Virtual Hub*.

Filename	Plateforme
libyapi.dylib	Max OS X
libyapi-amd64.so	Linux Intel (64 bits)
libyapi-armel.so	Linux ARM EL
libyapi-armhf.so	Linux ARM HL
libyapi-i386.so	Linux Intel (32 bits)
yapi64.dll	Windows (64 bits)
yapi.dll	Windows (32 bits)

Ces bibliothèques dynamiques contiennent toutes les fonctionnalités nécessaires pour reconstruire entièrement toute l'API haut niveau dans n'importe quel langage capable d'intégrer ces bibliothèques. Ce chapitre se limite cependant à décrire une utilisation de base des modules.

Contrôle d'un module

Les trois fonctions essentielles de l'API bas niveau sont les suivantes:

```
int yapiInitAPI(int connection_type, char *errmsg);
int yapiUpdateDeviceList(int forceupdate, char *errmsg);
int yapiHTTPRequest(char *device, char *request, char* buffer, int buffsize, int *fullsize,
char *errmsg);
```

La fonction *yapiInitAPI* permet d'initialiser l'API et doit être appelée une fois en début du programme. Pour une connexion de type USB, le paramètre *connection_type* doit prendre la valeur 1. *errmsg* est un pointeur sur un buffer de 255 caractères destiné à récupérer un éventuel message d'erreur. Ce pointeur peut être aussi mis à *NULL*. La fonction retourne un entier négatif en cas d'erreur, ou zéro dans le cas contraire.

La fonction *yapiUpdateDeviceList* gère l'inventaire des modules Yoctopuce connectés, elle doit être appelée au moins une fois. Pour pouvoir gérer le hot plug, et détecter d'éventuels nouveaux modules connectés, cette fonction devra être appelée à intervalles réguliers. Le paramètre *forceupdate* devra être à la valeur 1 pour forcer un scan matériel. Le paramètre *errmsg* devra pointer sur un buffer de 255 caractères pour récupérer un éventuel message d'erreur. Ce pointeur peut aussi être à *null*. Cette fonction retourne un entier négatif en cas d'erreur, ou zéro dans le cas contraire.

Enfin, la fonction *yapiHTTPRequest* permet d'envoyer des requêtes HTTP à l'API REST du module. Le paramètre *device* devra contenir le numéro de série ou le nom logique du module que vous cherchez à atteindre. Le paramètre *request* doit contenir la requête HTTP complète (y compris les sauts de ligne terminaux). *buffer* doit pointer sur un buffer de caractères suffisamment grand pour contenir la réponse. *buffsize* doit contenir la taille du buffer. *fullsize* est un pointeur sur un entier qui sera affecté à la taille effective de la réponse. Le paramètre *errmsg* devra pointer sur un buffer de

255 caractères pour récupérer un éventuel message d'erreur. Ce pointeur peut aussi être à *null*. Cette fonction retourne un entier négatif en cas d'erreur, ou zéro dans le cas contraire.

Le format des requêtes est le même que celui décrit dans la section *Virtual Hub et HTTP GET*. Toutes les chaînes de caractères utilisées par l'API sont des chaînes constituées de caractères 8 bits: l'Unicode et l'UTF8 ne sont pas supportés.

Le résultat retourné dans la variable *buffer* respecte le protocole HTTP, il inclut donc un header HTTP. Ce header se termine par deux lignes vides, c'est-à-dire une séquence de quatre caractères ASCII 13, 10, 13, 10.

Voici un programme d'exemple écrit en pascal qui utilise la DLL *yapi.dll* pour lire puis changer la luminosité d'un module.

```
// Dll functions import
function yapiInitAPI(mode:integer;
    errmsg : pansichar):integer;cdecl;
    external 'yapi.dll' name 'yapiInitAPI';
function yapiUpdateDeviceList(force:integer;errmsg : pansichar):integer;cdecl;
    external 'yapi.dll' name 'yapiUpdateDeviceList';
function yapiHTTPRequest(device:pansichar;url:pansichar; buffer:pansichar;
    buffsize:integer;var fullsize:integer;
    errmsg : pansichar):integer;cdecl;
    external 'yapi.dll' name 'yapiHTTPRequest';

var
    errmsgBuffer : array [0..256] of ansichar;
    dataBuffer : array [0..1024] of ansichar;
    errmsg,data : pansichar;
    fullsize,p : integer;

const
    serial = 'YRGBLED1-12345';
    getValue = 'GET /api/module/luminosity HTTP/1.1'#13#10#13#10;
    setValue = 'GET /api/module?luminosity=100 HTTP/1.1'#13#10#13#10;

begin
    errmsg := @errmsgBuffer;
    data := @dataBuffer;
    // API initialization
    if(yapiInitAPI(1,errmsg)<0) then
    begin
        writeln(errmsg);
        halt;
    end;

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

    // requests the module luminosity
    if (yapiHTTPRequest(serial,getValue,data,sizeof(dataBuffer),fullsize,errmsg)<0) then
    begin
        writeln(errmsg);
        halt;
    end;

    // searches for the HTTP header end
    p := pos(#13#10#13#10,data);

    // displays the response minus the HTTP header
    writeln(copy(data,p+4,length(data)-p-3));

    // change the luminosity
    if (yapiHTTPRequest(serial,setValue,data,sizeof(dataBuffer),fullsize,errmsg)<0) then
    begin
        writeln(errmsg);
        halt;
    end;

end.
```


Inventaire des modules

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

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

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

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

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

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

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

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

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

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

function yapiGetAllDevices( buffer:pointer;
  maxsize:integer;
  var neededsize:integer;
  errmsg : pansichar):integer; cdecl;
  external 'yapi.dll' name 'yapiGetAllDevices';
```

```

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

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

begin
    errmsg := @errmsgBuffer;

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

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

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

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

```

VB6 et yapi.dll

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

19.4. Port de la librairie haut niveau

Toutes les sources de l'API Yoctopuce étant fournies dans leur intégralité, vous pouvez parfaitement entreprendre le port complet de l'API dans le langage de votre choix. Sachez cependant qu'une grande partie du code source de l'API est généré automatiquement.

Ainsi, il n'est pas nécessaire de porter la totalité de l'API, il suffit de porter le fichier `yocto_api` et un de ceux correspondant à une fonctionnalité, par exemple `yocto_relay`. Moyennant un peu de travail supplémentaire, Yoctopuce sera alors en mesure de générer tous les autres fichiers. C'est pourquoi il est fortement recommandé de contacter le support Yoctopuce avant d'entreprendre le port de la librairie Yoctopuce dans un autre langage. Un travail collaboratif sera profitable aux deux parties.

20. Référence de l'API de haut niveau

Ce chapitre résume les fonctions de l'API de haut niveau pour commander votre Yocto-Color. 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.

20.1. Fonctions générales

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

yCheckLogicalName(name)

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

yDisableExceptions()

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

yEnableExceptions()

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

yEnableUSBHost(osContext)

Cette fonction est utilisée uniquement sous Android.

yFreeAPI()

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

yGetAPIVersion()

Retourne la version de la librairie Yoctopuce utilisée.

yGetTickCount()

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

yHandleEvents(errmsg)

Maintient la communication de la librairie avec les modules Yoctopuce.

yInitAPI(mode, errmsg)

Initialise la librairie de programmation de Yoctopuce explicitement.

yPreregisterHub(url, errmsg)

Alternative plus tolérante à `RegisterHub()`.

yRegisterDeviceArrivalCallback(arrivalCallback)

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

yRegisterDeviceRemovalCallback(removalCallback)

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

yRegisterHub(url, errmsg)

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

yRegisterHubDiscoveryCallback(hubDiscoveryCallback)

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

yRegisterLogFunction(logfun)

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

ySelectArchitecture(arch)

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

ySetDelegate(object)

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

ySetTimeout(callback, ms_timeout, args)

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

ySetUSBPacketAckMs(pktAckDelay)

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

ySleep(ms_duration, errmsg)

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

yTestHub(url, mstimeout, errmsg)

Test si un hub est joignable.

yTriggerHubDiscovery(errmsg)

Relance une détection des hubs réseau.

yUnregisterHub(url)

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

yUpdateDeviceList(errmsg)

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

yUpdateDeviceList_async(callback, context)

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

YAPI.CheckLogicalName() yCheckLogicalName()

YAPI

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

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

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

Paramètres :

name une chaîne de caractères contenant le nom vérifier.

Retourne :

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

YAPI.DisableExceptions() yDisableExceptions()

YAPI

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

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

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

YAPI.EnableExceptions() yEnableExceptions()

YAPI

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

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

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

YAPI.EnableUSBHost() yEnableUSBHost()

YAPI

Cette fonction est utilisée uniquement sous Android.

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

Avant d'appeler `yRegisterHub("usb")` il faut activer le port USB host du système. 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 aux modules à travers le réseau.

Paramètres :

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

YAPI.FreeAPI() yFreeAPI()

YAPI

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

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

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

YAPI.GetAPIVersion() yGetAPIVersion()

YAPI

Retourne la version de la librairie Yoctopuce utilisée.

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

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

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

Retourne :

une chaîne de caractères décrivant la version de la librairie.

YAPI.GetTickCount() yGetTickCount()

YAPI

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

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

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

Retourne :

un long entier contenant la valeur du compteur de millisecondes.

YAPI.HandleEvents() yHandleEvents()

YAPI

Maintient la communication de la librairie avec les modules Yoctopuce.

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

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

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

YAPI.InitAPI() yInitAPI()

YAPI

Initialise la librairie de programmation de Yoctopuce explicitement.

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

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

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

Paramètres :

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

Retourne :

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

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

YAPI.PreregisterHub() yPreregisterHub()

YAPI

Alternative plus tolérante à RegisterHub().

```

js  function yPreregisterHub( url, errmsg)
nodejs function PreregisterHub( url, errmsg)
cpp  YRETCODE yPreregisterHub( const string& url, string& errmsg)
m    +(YRETCODE) PreregisterHub :(NSString *) url :(NSError**) errmsg
pas  function yPreregisterHub( url: string, var errmsg: string): integer
vb   function yPreregisterHub( ByVal url As String,
                               ByRef errmsg As String) As Integer

cs   int PreregisterHub( string url, ref string errmsg)
java int PreregisterHub( String url)
uwp  async Task<int> PreregisterHub( string url)
py   def PreregisterHub( url, errmsg=None)
php  function yPreregisterHub( $url, &$errmsg)
es   function PreregisterHub( url, errmsg)

```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

YAPI.RegisterDeviceArrivalCallback() yRegisterDeviceArrivalCallback()

YAPI

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

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

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

Paramètres :

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

YAPI.RegisterDeviceRemovalCallback() yRegisterDeviceRemovalCallback()

YAPI

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

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

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

Paramètres :

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

YAPI.RegisterHub() yRegisterHub()

YAPI

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

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

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

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

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

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

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

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

http://nom:mot_de_passe@adresse:port

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

YAPI.RegisterHubDiscoveryCallback() yRegisterHubDiscoveryCallback()

YAPI

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

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

la fonction de callback reçoit deux chaînes de caractères en paramètre La première chaîne contient le numéro de série du hub réseau et la deuxième chaîne contient l'URL du hub. L'URL peut être passée directement en argument à la fonction yRegisterHub. Le callback sera appelé pendant l'exécution de la fonction yUpdateDeviceList, que vous devrez appeler régulièrement.

Paramètres :

hubDiscoveryCallback une procédure qui prend deux chaîne de caractères en paramètre, ou null

YAPI.RegisterLogFunction() yRegisterLogFunction()

YAPI

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

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

Utile pour débbuger le fonctionnement de l'API.

Paramètres :

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

YAPI.SelectArchitecture() ySelectArchitecture()

YAPI

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

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

Par défaut, la librairie Python détecte automatiquement la version de la librairie dynamique à utiliser pour accéder au port USB. Sous Linux ARM il n'est pas possible de détecter de manière fiable si il s'agit d'une installation Soft float (armel) ou Hard float (armhf). Dans ce cas, il est donc recommandé d'appeler `SelectArchitecture()` avant tout autre appel à la librairie pour forcer l'utilisation d'une architecture spécifiée.

Paramètres :

arch une chaîne de caractère spécifiant l'architecture à utiliser. Les valeurs possibles sont "armhf", "armel", "i386", "x86_64", "32bit", "64bit"

Retourne :

rien.

En cas d'erreur, déclenche une exception.

YAPI.SetDelegate() ySetDelegate()

YAPI

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

```
m +(void) SetDelegate :(id) object
```

Les méthodes yDeviceArrival et yDeviceRemoval seront appelées pendant l'exécution de la fonction yUpdateDeviceList, que vous devrez appeler régulièrement.

Paramètres :

object un objet qui doit se conformer au protocole YAPIDelegate, ou nil

YAPI.SetTimeout() ySetTimeout()

YAPI

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

```
js function ySetTimeout( callback, ms_timeout, args)
nodejs function SetTimeout( callback, ms_timeout, arguments)
es function SetTimeout( callback, ms_timeout, args)
```

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

YAPI.SetUSBPacketAckMs() ySetUSBPacketAckMs()

YAPI

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

```
java void SetUSBPacketAckMs( int pktAckDelay)
```

Cette fonction permet à la librairie de fonctionner même sur les téléphones Android qui perdent des paquets USB. Par défaut, la quittance est désactivée, car elle double le nombre de paquets échangés et donc ralentit sensiblement le fonctionnement de L'API. La quittance des paquets USB ne doit donc être activée que sur des tablette ou des téléphones qui posent problème. Un délais de 50 millisecondes est en général suffisant. En cas de doute contacter le support Yoctopuce. Pour désactiver la quittance des paquets USB, appeler cette fonction avec la valeur 0. Note : Cette fonctionnalité est disponible uniquement sous Android.

Paramètres :

pktAckDelay nombre de ms avant que le module ne renvoie

YAPI.Sleep() ySleep()

YAPI

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

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

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

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

YAPI.TestHub() yTestHub()

YAPI

Test si un hub est joignable.

```

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

```

Cette méthode n'enregistre pas le hub, elle ne fait que de vérifier que le hub est joignable. Le paramètre url suit les mêmes conventions que la méthode RegisterHub. Cette méthode est utile pour vérifier les paramètres d'authentification d'un hub. Il est possible de forcer la méthode à rendre la main après mstimeout millisecondes.

Paramètres :

- url** une chaîne de caractères contenant "usb","callback", ou l'URL racine du VirtualHub à utiliser.
- mstimeout** le nombre de millisecondes disponible pour tester la connexion.
- errmsg** une chaîne de caractères passée par référence, dans laquelle sera stocké un éventuel message d'erreur.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

En cas d'erreur retourne un code d'erreur négatif.

YAPI.TriggerHubDiscovery() yTriggerHubDiscovery()

YAPI

Relance une détection des hubs réseau.

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

Si une fonction de callback est enregistrée avec yRegisterDeviceRemovalCallback elle sera appelée à chaque hub réseau qui répondra à la détection SSDP.

Paramètres :

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

Retourne :

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

YAPI.UnregisterHub() yUnregisterHub()

YAPI

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

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

Paramètres :

url une chaîne de caractères contenant "usb" ou

YAPI.UpdateDeviceList() yUpdateDeviceList()

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

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

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

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

YAPI.UpdateDeviceList_async() yUpdateDeviceList_async()

YAPI

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

```
js function yUpdateDeviceList_async( callback, context)
nodejs function UpdateDeviceList_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-qu'el à la fonction de callback

Retourne :

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

20.2. Interface de contrôle du module

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

yFindModule(func)

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

yFindModuleInContext(yctx, func)

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

yFirstModule()

Commence l'énumération des modules accessibles par la librairie.

Méthodes des objets YModule

module→checkFirmware(path, onlynew)

Teste si le fichier byn est valide pour le module.

module→clearCache()

Invalide le cache.

module→describe()

Retourne un court texte décrivant le module.

module→download(pathname)

Télécharge le fichier choisi du module et retourne son contenu.

module→functionBaseType(functionIndex)

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

module→functionCount()

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

module→functionId(functionIndex)

Retourne l'identifiant matériel de la *nième* fonction du module.

module→functionName(functionIndex)

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

module→functionType(functionIndex)

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

module→functionValue(functionIndex)

Retourne la valeur publiée par la *nième* fonction du module.

module→get_allSettings()

Retourne tous les paramètres de configuration du module.

module→get_beacon()

Retourne l'état de la balise de localisation.

module→get_errorMessage()

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

module→get_errorType()

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

module→get_firmwareRelease()

Retourne la version du logiciel embarqué du module.

module→get_functionIds(funType)

Retourne les identifiants matériels des fonctions correspondant au type passé en argument.

module→get_hardwareId()

Retourne l'identifiant unique du module.

module→get_icon2d()

Retourne l'icône du module.

module→get_lastLogs()

Retourne une chaîne de caractère contenant les derniers logs du module.

module→get_logicalName()

Retourne le nom logique du module.

module→get_luminosity()

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

module→get_parentHub()

Retourne le numéro de série du YoctoHub sur lequel est connecté le module.

module→get_persistentSettings()

Retourne l'état courant des réglages persistents du module.

module→get_productId()

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

module→get_productName()

Retourne le nom commercial du module, préprogrammé en usine.

module→get_productRelease()

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

module→get_rebootCountdown()

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

module→get_serialNumber()

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

module→get_subDevices()

Retourne la liste des modules branchés au module courant.

module→get_upTime()

Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module

module→get_url()

Retourne l'URL utilisée pour accéder au module.

module→get_usbCurrent()

Retourne le courant consommé par le module sur le bus USB, en milliampères.

module→get_userdata()

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

module→get_userVar()

Retourne la valeur entière précédemment stockée dans cet attribut.

module→hasFunction(funcId)

Teste la présence d'une fonction pour le module courant.

module→isOnline()

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

module→isOnline_async(callback, context)

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

module→load(msValidity)

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

module→load_async(msValidity, callback, context)

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

module→log(text)

Ajoute un message arbitraire dans les logs du module.

module→nextModule()

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

module→reboot(secBeforeReboot)

Agende un simple redémarrage du module dans un nombre donné de secondes.

module→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→triggerFirmwareUpdate(secBeforeReboot)

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

module→updateFirmware(path)

Prepares une mise à jour de firmware du module.

module→**updateFirmwareEx**(**path**, **force**)

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() yFindModule()

YModule

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

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

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

Paramètres :

func une chaîne de caractères contenant soit le numéro de série, soit le nom logique du module désiré

Retourne :

un objet de classe `YModule` qui permet ensuite de contrôler le module ou d'obtenir de plus amples informations sur le module.

YModule.FindModuleInContext() yFindModuleInContext()

YModule

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

java	YModule FindModuleInContext(YAPIContext yctx , String func)
uwp	YModule FindModuleInContext(YAPIContext yctx , string func)
es	function FindModuleInContext(yctx , func)

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence le module sans ambiguïté

Retourne :

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

YModule.FirstModule() yFirstModule()

YModule

Commence l'énumération des modules accessibles par la librairie.

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

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

Retourne :

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

module→checkFirmware()**YModule**

Teste si le fichier byn est valide pour le module.

js	function checkFirmware (path , onlynew)
nodejs	function checkFirmware (path , onlynew)
cpp	string checkFirmware (string path , bool onlynew)
m	-(NSString*) checkFirmware : (NSString*) path : (bool) onlynew
pas	function checkFirmware (path : string, onlynew : boolean): string
vb	function checkFirmware () As String
cs	string checkFirmware (string path , bool onlynew)
java	String checkFirmware (String path , boolean onlynew)
uwp	async Task<string> checkFirmware (string path , bool onlynew)
py	def checkFirmware (path , onlynew)
php	function checkFirmware (\$ path , \$ onlynew)
es	function checkFirmware (path , onlynew)
cmd	YModule target checkFirmware path onlynew

Cette méthode est utile pour vérifier si il est nécessaire de mettre à jour le module avec un nouveau firmware. Il est possible de passer un répertoire qui contiens plusieurs fichier .byn. Dans ce cas cette methode retourne le path du fichier .byn compatible le plus récent. Si le parametre onlynew est vrais, les firmwares équivalents ou plus anciens que le firmware actuellement installé sont ignorés.

Paramètres :

- path** le path d'un fichier .byn ou d'un répertoire contenant plusieurs fichier .byn
- onlynew** retourne uniquement les fichiers strictement plus récents

Retourne :

le path du fichier .byn à utiliser, ou une chaîne vide si aucun firmware plus récent n'est disponible En cas d'erreur, déclenche une exception ou retourne une chaine de caractère qui comment par "error:".

module→**clearCache()****YModule**

Invalide le cache.

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

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

module→describe()**YModule**

Retourne un court texte décrivant le module.

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

Ce texte peut contenir soit le nom logique du module, soit son numéro de série.

Retourne :

une chaîne de caractères décrivant le module

module→download()**YModule**

Télécharge le fichier choisi du module et retourne son contenu.

js	function download (pathname)
nodejs	function download (pathname)
cpp	string download (string pathname)
m	-(NSMutableData*) download : (NSString*) pathname
pas	function download (pathname : string): TByteArray
vb	function download () As Byte
cs	byte[] download (string pathname)
java	byte[] download (String pathname)
uwp	async Task<byte[]> download (string pathname)
py	def download (pathname)
php	function download (\$pathname)
es	function download (pathname)
cmd	YModule target download pathname

Paramètres :

pathname nom complet du fichier

Retourne :

le contenu du fichier chargé

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

module→**functionBaseType()****YModule**

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

js	function functionBaseType (functionIndex)
nodejs	function functionBaseType (functionIndex)
cpp	string functionBaseType (int functionIndex)
pas	function functionBaseType (functionIndex : integer): string
vb	function functionBaseType (ByVal functionIndex As Integer) As String
cs	string functionBaseType (int functionIndex)
java	String functionBaseType (int functionIndex)
py	def functionBaseType (functionIndex)
php	function functionBaseType (\$functionIndex)
es	function functionBaseType (functionIndex)

Par exemple, le type de base de toutes les fonctions de mesure est "Sensor".

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant au type de base de la fonction

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

module→functionCount()**YModule**

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

js	function functionCount ()
nodejs	function functionCount ()
cpp	int functionCount ()
m	-(int) functionCount
pas	function functionCount (): integer
vb	function functionCount () As Integer
cs	int functionCount ()
java	int functionCount ()
py	def functionCount ()
php	function functionCount ()
es	function functionCount ()

Retourne :

le nombre de fonctions sur le module

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

module→**functionId()****YModule**

Retourne l'identifiant matériel de la *n*ème fonction du module.

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

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant à l'identifiant matériel unique de la fonction désirée

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

module→**functionName()****YModule**

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

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

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant au nom logique de la fonction désirée

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

module→functionType()**YModule**

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

js	function functionType (functionIndex)
nodejs	function functionType (functionIndex)
cpp	string functionType (int functionIndex)
pas	function functionType (functionIndex : integer): string
vb	function functionType (ByVal functionIndex As Integer) As String
cs	string functionType (int functionIndex)
java	String functionType (int functionIndex)
py	def functionType (functionIndex)
php	function functionType (\$functionIndex)
es	function functionType (functionIndex)

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant au type de la fonction

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

module→**functionValue()****YModule**

Retourne la valeur publiée par la *nième* fonction du module.

js	function functionValue (functionIndex)
nodejs	function functionValue (functionIndex)
cpp	string functionValue (int functionIndex)
m	-(NSString*) functionValue : (int) functionIndex
pas	function functionValue (functionIndex : integer): string
vb	function functionValue (ByVal functionIndex As Integer) As String
cs	string functionValue (int functionIndex)
java	String functionValue (int functionIndex)
py	def functionValue (functionIndex)
php	function functionValue (\$functionIndex)
es	function functionValue (functionIndex)

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant à la valeur publiée par la fonction désirée

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

module→get_allSettings() module→allSettings()

YModule

Retourne tous les paramètres de configuration du module.

js	function get_allSettings ()
nodejs	function get_allSettings ()
cpp	string get_allSettings ()
m	-(NSMutableData*) allSettings
pas	function get_allSettings (): TByteArray
vb	function get_allSettings () As Byte
cs	byte[] get_allSettings ()
java	byte[] get_allSettings ()
uwp	async Task<byte[]> get_allSettings ()
py	def get_allSettings ()
php	function get_allSettings ()
es	function get_allSettings ()
cmd	YModule target get_allSettings

Utile pour sauvgarder les noms logiques, les calibrations et fichiers uploadés d'un module.

Retourne :

un objet binaire avec tous les paramètres

En cas d'erreur, déclenche une exception ou retourne un objet binaire de taille 0.

module→**get_beacon()****YModule****module**→**beacon()**

Retourne l'état de la balise de localisation.

js	function get_beacon ()
nodejs	function get_beacon ()
cpp	Y_BEACON_enum get_beacon ()
m	-(Y_BEACON_enum) beacon
pas	function get_beacon (): Integer
vb	function get_beacon () As Integer
cs	int get_beacon ()
java	int get_beacon ()
uwp	async Task<int> get_beacon ()
py	def get_beacon ()
php	function get_beacon ()
es	function get_beacon ()
cmd	YModule target get_beacon

Retourne :

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

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

module→**get_errorMessage()**
module→**errorMessage()**

YModule

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

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

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

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du module

module→**get_errorType()****YModule****module**→**errorType()**

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

js	function get_errorType ()
nodejs	function get_errorType ()
cpp	YRETCODE get_errorType ()
pas	function get_errorType (): YRETCODE
vb	function get_errorType () As YRETCODE
cs	YRETCODE get_errorType ()
java	int get_errorType ()
py	def get_errorType ()
php	function get_errorType ()
es	function get_errorType ()

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

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du module

module→**get_firmwareRelease()**
module→**firmwareRelease()**

YModule

Retourne la version du logiciel embarqué du module.

js	function get_firmwareRelease ()
nodejs	function get_firmwareRelease ()
cpp	string get_firmwareRelease ()
m	-(NSString*) firmwareRelease
pas	function get_firmwareRelease (): string
vb	function get_firmwareRelease () As String
cs	string get_firmwareRelease ()
java	String get_firmwareRelease ()
uwp	async Task<string> get_firmwareRelease ()
py	def get_firmwareRelease ()
php	function get_firmwareRelease ()
es	function get_firmwareRelease ()
cmd	YModule target get_firmwareRelease

Retourne :

une chaîne de caractères représentant la version du logiciel embarqué du module

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

module→**get_functionIds()****YModule****module**→**functionIds()**

Retourne les identifiants matériels des fonctions correspondant au type passé en argument.

js	function get_functionIds (funType)
nodejs	function get_functionIds (funType)
cpp	vector<string> get_functionIds (string funType)
m	-(NSMutableArray*) functionIds : (NSString*) funType
pas	function get_functionIds (funType : string): TStringArray
vb	function get_functionIds () As List
cs	List<string> get_functionIds (string funType)
java	ArrayList<String> get_functionIds (String funType)
uwp	async Task<List<string>> get_functionIds (string funType)
py	def get_functionIds (funType)
php	function get_functionIds (\$funType)
es	function get_functionIds (funType)
cmd	YModule target get_functionIds funType

Paramètres :

funType Le type de fonction (Relay, LightSensor, Voltage,...)

Retourne :

un tableau de chaînes de caractère.

module→**get_hardwareId()**
module→**hardwareId()**

YModule

Retourne l'identifiant unique du module.

js	function get_hardwareId ()
nodejs	function get_hardwareId ()
cpp	string get_hardwareId ()
m	-(NSString*) hardwareId
vb	function get_hardwareId () As String
cs	string get_hardwareId ()
java	String get_hardwareId ()
py	def get_hardwareId ()
php	function get_hardwareId ()
es	function get_hardwareId ()

L'identifiant unique est composé du numéro de série du module suivi de la chaîne ".module".

Retourne :

une chaîne de caractères identifiant la fonction

module→**get_icon2d()****YModule****module**→**icon2d()**

Retourne l'icône du module.

js	function get_icon2d ()
nodejs	function get_icon2d ()
cpp	string get_icon2d ()
m	-(NSMutableData*) icon2d
pas	function get_icon2d (): TByteArray
vb	function get_icon2d () As Byte
cs	byte[] get_icon2d ()
java	byte[] get_icon2d ()
uwp	async Task<byte[]> get_icon2d ()
py	def get_icon2d ()
php	function get_icon2d ()
es	function get_icon2d ()
cmd	YModule target get_icon2d

L'icone est au format PNG et a une taille maximale de 1536 octets.

Retourne :

un buffer binaire contenant l'icone, au format png. En cas d'erreur, déclenche une exception ou retourne YAPI_INVALID_STRING.

module→get_lastLogs() module→lastLogs()

YModule

Retourne une chaîne de caractère contenant les derniers logs du module.

js	function get_lastLogs ()
nodejs	function get_lastLogs ()
cpp	string get_lastLogs ()
m	-(NSString*) lastLogs
pas	function get_lastLogs (): string
vb	function get_lastLogs () As String
cs	string get_lastLogs ()
java	String get_lastLogs ()
uwp	async Task<string> get_lastLogs ()
py	def get_lastLogs ()
php	function get_lastLogs ()
es	function get_lastLogs ()
cmd	YModule target get_lastLogs

Cette méthode retourne les derniers logs qui sont encore stocké dans le module.

Retourne :

une chaîne de caractère contenant les derniers logs du module. En cas d'erreur, déclenche une exception ou retourne YAPI_INVALID_STRING.

module→**get_logicalName()****YModule****module**→**logicalName()**

Retourne le nom logique du module.

js	function get_logicalName ()
nodejs	function get_logicalName ()
cpp	string get_logicalName ()
m	-(NSString*) logicalName
pas	function get_logicalName (): string
vb	function get_logicalName () As String
cs	string get_logicalName ()
java	String get_logicalName ()
uwp	async Task<string> get_logicalName ()
py	def get_logicalName ()
php	function get_logicalName ()
es	function get_logicalName ()
cmd	YModule target get_logicalName

Retourne :

une chaîne de caractères représentant le nom logique du module

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

module→**get_luminosity()**
module→**luminosity()**

YModule

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

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

Retourne :

un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

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

module→**get_parentHub()****YModule****module**→**parentHub()**

Retourne le numéro de série du YoctoHub sur lequel est connecté le module.

js	function get_parentHub ()
nodejs	function get_parentHub ()
cpp	string get_parentHub ()
m	-(NSString*) parentHub
pas	function get_parentHub (): string
vb	function get_parentHub () As String
cs	string get_parentHub ()
java	String get_parentHub ()
py	def get_parentHub ()
php	function get_parentHub ()
cmd	YModule target get_parentHub

Si le module est connecté par USB, ou si le module est le YoctoHub racine, une chaîne vide est retournée.

Retourne :

une chaîne de caractères contenant le numéro de série du YoctoHub, ou une chaîne vide.

module→**get_persistentSettings()**
module→**persistentSettings()**

YModule

Retourne l'état courant des réglages persistents du module.

js	function get_persistentSettings ()
nodejs	function get_persistentSettings ()
cpp	Y_PERSISTENTSETTINGS_enum get_persistentSettings ()
m	-(Y_PERSISTENTSETTINGS_enum) persistentSettings
pas	function get_persistentSettings (): Integer
vb	function get_persistentSettings () As Integer
cs	int get_persistentSettings ()
java	int get_persistentSettings ()
uwp	async Task<int> get_persistentSettings ()
py	def get_persistentSettings ()
php	function get_persistentSettings ()
es	function get_persistentSettings ()
cmd	YModule target get_persistentSettings

Retourne :

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

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

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

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

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

Retourne :

un entier représentant l'identifiant USB du module, préprogrammé en usine

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

module→**get_productName()****YModule****module**→**productName()**

Retourne le nom commercial du module, préprogrammé en usine.

js	function get_productName ()
nodejs	function get_productName ()
cpp	string get_productName ()
m	-(NSString*) productName
pas	function get_productName (): string
vb	function get_productName () As String
cs	string get_productName ()
java	String get_productName ()
uwp	async Task<string> get_productName ()
py	def get_productName ()
php	function get_productName ()
es	function get_productName ()
cmd	YModule target get_productName

Retourne :

une chaîne de caractères représentant le nom commercial du module, préprogrammé en usine

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

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

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

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

Retourne :

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

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

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

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

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

Retourne :

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

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

module→**get_serialNumber()****YModule****module**→**serialNumber()**

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

js	function get_serialNumber ()
nodejs	function get_serialNumber ()
cpp	string get_serialNumber ()
m	-(NSString*) serialNumber
pas	function get_serialNumber (): string
vb	function get_serialNumber () As String
cs	string get_serialNumber ()
java	String get_serialNumber ()
uwp	async Task<string> get_serialNumber ()
py	def get_serialNumber ()
php	function get_serialNumber ()
es	function get_serialNumber ()
cmd	YModule target get_serialNumber

Retourne :

une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine

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

module→get_subDevices()
module→subDevices()

YModule

Retourne la liste des modules branchés au module courant.

js	function get_subDevices ()
nodejs	function get_subDevices ()
cpp	vector<string> get_subDevices ()
m	-(NSMutableArray*) subDevices
pas	function get_subDevices (): TStringArray
vb	function get_subDevices () As List
cs	List<string> get_subDevices ()
java	ArrayList<String> get_subDevices ()
py	def get_subDevices ()
php	function get_subDevices ()
cmd	YModule target get_subDevices

Cette fonction n'est pertinente que lorsqu'elle appelée pour un YoctoHub ou pour le VirtualHub. Dans le cas contraire, un tableau vide est retourné.

Retourne :

un tableau de chaînes de caractères contenant les numéros de série des sous-modules connectés au module

module→**get_upTime()****YModule****module**→**upTime()**

Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module

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

Retourne :

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

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

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

Retourne l'URL utilisée pour accéder au module.

js	function get_url ()
nodejs	function get_url ()
cpp	string get_url ()
m	-(NSString*) url
pas	function get_url (): string
vb	function get_url () As String
cs	string get_url ()
java	String get_url ()
py	def get_url ()
php	function get_url ()
cmd	YModule target get_url

Si le module est connecté par USB la chaîne de caractère 'usb' est retournée.

Retourne :

une chaîne de caractère contenant l'URL du module.

module→**get_usbCurrent()****YModule****module**→**usbCurrent()**

Retourne le courant consommé par le module sur le bus USB, en milliampères.

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

Retourne :

un entier représentant le courant consommé par le module sur le bus USB, en milliampères

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

module→get_userdata()**YModule****module→userdata()**

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

js	function get_userdata ()
nodejs	function get_userdata ()
cpp	void * get_userdata ()
m	-(id) <code>userData</code>
pas	function get_userdata (): Tobject
vb	function get_userdata () As Object
cs	object get_userdata ()
java	Object get_userdata ()
py	def get_userdata ()
php	function get_userdata ()
es	function get_userdata ()

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

Retourne :

l'objet stocké précédemment par l'appelant.

module→**get_userVar()****YModule****module**→**userVar()**

Retourne la valeur entière précédemment stockée dans cet attribut.

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

Au démarrage du module (ou après un redémarrage), la valeur est toujours zéro.

Retourne :

un entier représentant la valeur entière précédemment stockée dans cet attribut

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

module→hasFunction()**YModule**

Teste la présence d'une fonction pour le module courant.

js	function hasFunction (funcId)
nodejs	function hasFunction (funcId)
cpp	bool hasFunction (string funcId)
m	-(bool) hasFunction : (NSString*) funcId
pas	function hasFunction (funcId : string): boolean
vb	function hasFunction () As Boolean
cs	bool hasFunction (string funcId)
java	boolean hasFunction (String funcId)
uwp	async Task<bool> hasFunction (string funcId)
py	def hasFunction (funcId)
php	function hasFunction (\$funcId)
es	function hasFunction (funcId)
cmd	YModule target hasFunction funcId

La méthode prend en paramètre l'identifiant de la fonction (relay1, voltage2,...) et retourne un booléen.

Paramètres :

funcId identifiant matériel de la fonction

Retourne :

vrai si le module inclut la fonction demandée

module→isOnline()**YModule**

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

js	function isOnline ()
nodejs	function isOnline ()
cpp	bool isOnline ()
m	-(BOOL) isOnline
pas	function isOnline (): boolean
vb	function isOnline () As Boolean
cs	bool isOnline ()
java	boolean isOnline ()
py	def isOnline ()
php	function isOnline ()
es	function isOnline ()

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

Retourne :

true si le module est joignable, false sinon

module→isOnline_async()**YModule**

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

```
js function isOnline_async( callback, context)
nodejs 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)
nodejs	function load (msValidity)
cpp	YRETCODE load (int msValidity)
m	-(YRETCODE) load : (int) msValidity
pas	function load (msValidity : integer): YRETCODE
vb	function load (ByVal msValidity As Integer) As YRETCODE
cs	YRETCODE load (ulong msValidity)
java	int load (long msValidity)
py	def load (msValidity)
php	function load (\$msValidity)
es	function load (msValidity)

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→load_async()**YModule**

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

```
js function load_async( msValidity, callback, context)
nodejs 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)
nodejs	function log (text)
cpp	int log (string text)
m	-(int) log : (NSString*) text
pas	function log (text : string): LongInt
vb	function log () As Integer
cs	int log (string text)
java	int log (String text)
uwp	async Task<int> log (string text)
py	def log (text)
php	function log (\$ text)
es	function log (text)
cmd	YModule target log text

Cette fonction est utile en particulier pour tracer l'exécution de callbacks HTTP. Si un saut de ligne est désiré après le message, il doit être inclus dans la chaîne de caractère.

Paramètres :

text le message à ajouter aux logs du module.

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→nextModule()**YModule**

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

js	function nextModule ()
nodejs	function nextModule ()
cpp	YModule * nextModule ()
m	-(YModule*) nextModule
pas	function nextModule (): TYModule
vb	function nextModule () As YModule
cs	YModule nextModule ()
java	YModule nextModule ()
uwp	YModule nextModule ()
py	def nextModule ()
php	function nextModule ()
es	function nextModule ()

Retourne :

un pointeur sur un objet `YModule` accessible en ligne, ou `null` lorsque l'énumération est terminée.

module→reboot()**YModule**

Agende un simple redémarrage du module dans un nombre donné de secondes.

js	function reboot (secBeforeReboot)
nodejs	function reboot (secBeforeReboot)
cpp	int reboot (int secBeforeReboot)
m	-(int) reboot : (int) secBeforeReboot
pas	function reboot (secBeforeReboot : LongInt): LongInt
vb	function reboot () As Integer
cs	int reboot (int secBeforeReboot)
java	int reboot (int secBeforeReboot)
uwp	async Task<int> reboot (int secBeforeReboot)
py	def reboot (secBeforeReboot)
php	function reboot (\$ secBeforeReboot)
es	function reboot (secBeforeReboot)
cmd	YModule target reboot secBeforeReboot

Paramètres :

secBeforeReboot nombre de secondes avant de redémarrer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→registerLogCallback()**YModule**

Enregistre une fonction de callback qui sera appelée à chaque fois le module émet un message de log.

cpp	<code>void registerLogCallback(YModuleLogCallback callback)</code>
m	<code>-(void) registerLogCallback : (YModuleLogCallback) callback</code>
vb	<code>function registerLogCallback(ByVal callback As YModuleLogCallback) As Integer</code>
cs	<code>int registerLogCallback(LogCallback callback)</code>
java	<code>void registerLogCallback(LogCallback callback)</code>
py	<code>def registerLogCallback(callback)</code>

Utile pour déboguer le fonctionnement d'un module Yoctopuce.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet module qui a produit un log, un chaîne de caractère qui contiens le log

module→revertFromFlash()**YModule**

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→saveToFlash()**YModule**

Sauve les réglages courants dans la mémoire non volatile du module.

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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

Rétablit tous les paramètres du module.

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

Utile pour restorer les noms logiques et les calibrations du module depuis une sauvgarde. N'oubliez pas d'appeler la méthode `saveToF\ash( )` 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()**
module→**setAllSettingsAndFiles()**

YModule

Rétablit tous les paramètres de configuration et fichiers sur un module.

js	function set_allSettingsAndFiles (settings)
nodejs	function set_allSettingsAndFiles (settings)
cpp	int set_allSettingsAndFiles (string settings)
m	-(int) setAllSettingsAndFiles : (NSData*) settings
pas	function set_allSettingsAndFiles (settings : TByteArray): LongInt
vb	procedure set_allSettingsAndFiles ()
cs	int set_allSettingsAndFiles ()
java	int set_allSettingsAndFiles (byte[] settings)
uwp	async Task<int> set_allSettingsAndFiles ()
py	def set_allSettingsAndFiles (settings)
php	function set_allSettingsAndFiles (\$ settings)
es	function set_allSettingsAndFiles (settings)
cmd	YModule target set_allSettingsAndFiles settings

Cette méthode est utile pour récupérer les noms logiques, les calibrations, les fichiers uploadés, etc. du module depuis une 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)
nodejs	function set_beacon (newval)
cpp	int set_beacon (Y_BEACON_enum newval)
m	-(int) setBeacon : (Y_BEACON_enum) newval
pas	function set_beacon (newval : Integer): integer
vb	function set_beacon (ByVal newval As Integer) As Integer
cs	int set_beacon (int newval)
java	int set_beacon (int newval)
uwp	async Task<int> set_beacon (int newval)
py	def set_beacon (newval)
php	function set_beacon (\$ newval)
es	function set_beacon (newval)
cmd	YModule target set_beacon newval

Paramètres :

newval soit Y_BEACON_OFF, soit Y_BEACON_ON

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→set_logicalName() module→setLogicalName()

YModule

Change le nom logique du module.

js	function set_logicalName (newval)
nodejs	function set_logicalName (newval)
cpp	int set_logicalName (const string& newval)
m	-(int) setLogicalName : (NSString*) newval
pas	function set_logicalName (newval : string): integer
vb	function set_logicalName (ByVal newval As String) As Integer
cs	int set_logicalName (string newval)
java	int set_logicalName (String newval)
uwp	async Task<int> set_logicalName (string newval)
py	def set_logicalName (newval)
php	function set_logicalName (\$ newval)
es	function set_logicalName (newval)
cmd	YModule target set_logicalName newval

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→**set_luminosity()****YModule****module**→**setLuminosity()**

Modifie la luminosité des leds informatives du module.

js	function set_luminosity (newval)
nodejs	function set_luminosity (newval)
cpp	int set_luminosity (int newval)
m	-(int) setLuminosity : (int) newval
pas	function set_luminosity (newval : LongInt): integer
vb	function set_luminosity (ByVal newval As Integer) As Integer
cs	int set_luminosity (int newval)
java	int set_luminosity (int newval)
uwp	async Task<int> set_luminosity (int newval)
py	def set_luminosity (newval)
php	function set_luminosity (\$newval)
es	function set_luminosity (newval)
cmd	YModule target set_luminosity newval

Le paramètre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval un entier représentant la luminosité des leds informatives du module

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→**set_userdata()****YModule****module**→**setUserData()**

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

js	function set_userdata (data)
nodejs	function set_userdata (data)
c++	void set_userdata (void* data)
m	-(void) setUserData : (id) data
pas	procedure set_userdata (data : Tobject)
vb	procedure set_userdata (ByVal data As Object)
cs	void set_userdata (object data)
java	void set_userdata (Object data)
py	def set_userdata (data)
php	function set_userdata (\$ data)
es	function set_userdata (data)

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

Paramètres :

data objet quelconque à mémoriser

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

Stocke une valeur 32 bits dans la mémoire volatile du module.

js	function set_userVar (newval)
nodejs	function set_userVar (newval)
cpp	int set_userVar (int newval)
m	-(int) setUserVar : (int) newval
pas	function set_userVar (newval : LongInt): integer
vb	function set_userVar (ByVal newval As Integer) As Integer
cs	int set_userVar (int newval)
java	int set_userVar (int newval)
uwp	async Task<int> set_userVar (int newval)
py	def set_userVar (newval)
php	function set_userVar (\$newval)
es	function set_userVar (newval)
cmd	YModule target set_userVar newval

Cet attribut est à la disposition du programmeur pour y stocker par exemple une variable d'état. Au démarrage du module (ou après un redémarrage), la valeur est toujours zéro.

Paramètres :

newval un entier

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→triggerFirmwareUpdate()**YModule**

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

js	function triggerFirmwareUpdate (secBeforeReboot)
nodejs	function triggerFirmwareUpdate (secBeforeReboot)
cpp	int triggerFirmwareUpdate (int secBeforeReboot)
m	-(int) triggerFirmwareUpdate : (int) secBeforeReboot
pas	function triggerFirmwareUpdate (secBeforeReboot : LongInt): LongInt
vb	function triggerFirmwareUpdate () As Integer
cs	int triggerFirmwareUpdate (int secBeforeReboot)
java	int triggerFirmwareUpdate (int secBeforeReboot)
uwp	async Task<int> triggerFirmwareUpdate (int secBeforeReboot)
py	def triggerFirmwareUpdate (secBeforeReboot)
php	function triggerFirmwareUpdate (\$secBeforeReboot)
es	function triggerFirmwareUpdate (secBeforeReboot)
cmd	YModule target triggerFirmwareUpdate secBeforeReboot

Paramètres :

secBeforeReboot nombre de secondes avant de redémarrer

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→updateFirmware()**YModule**

Prepare une mise à jour de firmware du module.

js	function updateFirmware (path)
nodejs	function updateFirmware (path)
cpp	YFirmwareUpdate updateFirmware (string path)
m	-(YFirmwareUpdate*) updateFirmware : (NSString*) path
pas	function updateFirmware (path : string): TYFirmwareUpdate
vb	function updateFirmware () As YFirmwareUpdate
cs	YFirmwareUpdate updateFirmware (string path)
java	YFirmwareUpdate updateFirmware (String path)
uwp	async Task<YFirmwareUpdate> updateFirmware (string path)
py	def updateFirmware (path)
php	function updateFirmware (\$path)
es	function updateFirmware (path)
cmd	YModule target updateFirmware path

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

Paramètres :

path le path du fichier .byn à utiliser

Retourne :

un objet YFirmwareUpdate ou NULL en cas d'erreur

module→updateFirmwareEx()**YModule**

Prepares une mise à jour de firmware du module.

js	function updateFirmwareEx (path , force)
nodejs	function updateFirmwareEx (path , force)
cpp	YFirmwareUpdate updateFirmwareEx (string path , bool force)
m	-(YFirmwareUpdate*) updateFirmwareEx : (NSString*) path : (bool) force
pas	function updateFirmwareEx (path : string, force : boolean): TYFirmwareUpdate
vb	function updateFirmwareEx () As YFirmwareUpdate
cs	YFirmwareUpdate updateFirmwareEx (string path , bool force)
java	YFirmwareUpdate updateFirmwareEx (String path , boolean force)
uwp	async Task<YFirmwareUpdate> updateFirmwareEx (string path , bool force)
py	def updateFirmwareEx (path , force)
php	function updateFirmwareEx (\$ path , \$ force)
es	function updateFirmwareEx (path , force)
cmd	YModule target updateFirmwareEx path force

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

Paramètres :

path le path du fichier .byn à utiliser

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

Retourne :

un objet YFirmwareUpdate ou NULL en cas d'erreur

module→**wait_async()****YModule**

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

```
js function wait_async( callback, context)
nodejs function wait_async( callback, context)
es function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

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

Retourne :

rien du tout.

20.3. Interface de la fonction ColorLed

La librairie de programmation Yoctopuce permet de piloter une LED couleur aussi bien en coordonnées RGB qu'en coordonnées HSL, les conversions RGB vers HSL étant faites automatiquement par le module. Ceci permet aisément d'allumer la LED avec une certaine teinte et d'en faire progressivement varier la saturation ou la luminosité. Si nécessaire, vous trouverez plus d'information sur la différence entre RGB et HSL dans la section suivante.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

yFindColorLed(func)

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

yFindColorLedInContext(yctx, func)

Permet de retrouver une LED RGB d'après un identifiant donné dans un Context YAPI.

yFirstColorLed()

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

yFirstColorLedInContext(yctx)

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

Méthodes des objets YColorLed

colorled→addHslMoveToBlinkSeq(HSLcolor, msDelay)

Ajoute une transition à la séquence de clignotement du module, la transition s'effectuera dans l'espace de couleur HSL.

colorled→addRgbMoveToBlinkSeq(RGBcolor, msDelay)

Ajoute une transition à la séquence de clignotement du module, la transition s'effectuera dans l'espace de couleur RGB

colorled→clearCache()

Invalide le cache.

colorled→describe()

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

colorled→get_advertisedValue()

Retourne la valeur courante de la LED RGB (pas plus de 6 caractères).

colorled→get_blinkSeqMaxSize()

Retourne la longueur maximum de la sequence de clignotement.

colorled→get_blinkSeqSignature()

	Retourne la signature de la signature de la séquence de clignotement.
colorled→get_blinkSeqSize()	Retourne la longueur actuelle de la sequence de clignotement.
colorled→get_errorMessage()	Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la LED RGB.
colorled→get_errorType()	Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la LED RGB.
colorled→get_friendlyName()	Retourne un identifiant global de la LED RGB au format <code>NOM_MODULE . NOM_FONCTION</code> .
colorled→get_functionDescriptor()	Retourne un identifiant unique de type <code>YFUN_DESCR</code> correspondant à la fonction.
colorled→get_functionId()	Retourne l'identifiant matériel de la LED RGB, sans référence au module.
colorled→get_hardwareId()	Retourne l'identifiant matériel unique de la LED RGB au format <code>SERIAL . FUNCTIONID</code> .
colorled→get_hslColor()	Retourne la couleur HSL courante de la LED.
colorled→get_logicalName()	Retourne le nom logique de la LED RGB.
colorled→get_module()	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
colorled→get_module_async(callback, context)	Retourne l'objet <code>YModule</code> correspondant au module Yoctopuce qui héberge la fonction.
colorled→get_rgbColor()	Retourne la couleur RGB courante de la LED.
colorled→get_rgbColorAtPowerOn()	Retourne la couleur configurée pour être affichage à l'allumage du module.
colorled→get_userData()	Retourne le contenu de l'attribut <code>userData</code> , précédemment stocké à l'aide de la méthode <code>set_userData</code> .
colorled→hslMove(hsl_target, ms_duration)	Effectue une transition continue dans l'espace HSL entre la couleur courante et une nouvelle couleur.
colorled→isOnline()	Vérifie si le module hébergeant la LED RGB est joignable, sans déclencher d'erreur.
colorled→isOnline_async(callback, context)	Vérifie si le module hébergeant la LED RGB est joignable, sans déclencher d'erreur.
colorled→load(msValidity)	Met en cache les valeurs courantes de la LED RGB, avec une durée de validité spécifiée.
colorled→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.
colorled→load_async(msValidity, callback, context)	Met en cache les valeurs courantes de la LED RGB, avec une durée de validité spécifiée.
colorled→muteValueCallbacks()	Désactive l'envoi de chaque changement de la valeur publiée au hub parent.
colorled→nextColorLed()	

Continue l'énumération des LEDs RGB commencée à l'aide de `yFirstColorLed()`.

`colorled→registerValueCallback(callback)`

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

`colorled→resetBlinkSeq()`

efface le contenu de la sequence de clignotement.

`colorled→rgbMove(rgb_target, ms_duration)`

Effectue une transition continue dans l'espace RGB entre la couleur courante et une nouvelle couleur.

`colorled→set_hslColor(newval)`

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

`colorled→set_logicalName(newval)`

Modifie le nom logique de la LED RGB.

`colorled→set_rgbColor(newval)`

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

`colorled→set_rgbColorAtPowerOn(newval)`

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

`colorled→set_userData(data)`

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

`colorled→startBlinkSeq()`

Démarre l'exécution de la séquence préprogrammée de clignotement.

`colorled→stopBlinkSeq()`

Arrête l'exécution de la séquence préprogrammée de clignotement.

`colorled→unmuteValueCallbacks()`

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

`colorled→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.

YColorLed.FindColorLed() yFindColorLed()

YColorLed

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

js	function yFindColorLed (func)
nodejs	function FindColorLed (func)
cpp	YColorLed* yFindColorLed (string func)
m	+(YColorLed*) FindColorLed : (NSString*) func
pas	function yFindColorLed (func : string): TYColorLed
vb	function yFindColorLed (ByVal func As String) As YColorLed
cs	YColorLed FindColorLed (string func)
java	YColorLed FindColorLed (String func)
uwp	YColorLed FindColorLed (string func)
py	def FindColorLed (func)
php	function yFindColorLed (\$func)
es	function FindColorLed (func)

L'identifiant peut être spécifié sous plusieurs formes:

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

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

Paramètres :

func une chaîne de caractères qui référence la LED RGB sans ambiguïté

Retourne :

un objet de classe `YColorLed` qui permet ensuite de contrôler la LED RGB.

YColorLed.FindColorLedInContext() yFindColorLedInContext()

YColorLed

Permet de retrouver une LED RGB d'après un identifiant donné dans un Context YAPI.

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

Paramètres :

yctx un contexte YAPI

func une chaîne de caractères qui référence la LED RGB sans ambiguïté

Retourne :

un objet de classe `YColorLed` qui permet ensuite de contrôler la LED RGB.

YColorLed.FirstColorLed() yFirstColorLed()

YColorLed

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

js	function yFirstColorLed ()
nodejs	function FirstColorLed ()
cpp	YColorLed* yFirstColorLed ()
m	+(YColorLed*) FirstColorLed
pas	function yFirstColorLed (): TYColorLed
vb	function yFirstColorLed () As YColorLed
cs	YColorLed FirstColorLed ()
java	YColorLed FirstColorLed ()
uwp	YColorLed FirstColorLed ()
py	def FirstColorLed ()
php	function yFirstColorLed ()
es	function FirstColorLed ()

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

Retourne :

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

YColorLed.FirstColorLedInContext() yFirstColorLedInContext()

YColorLed

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

java	YColorLed FirstColorLedInContext(YAPIContext yctx)
uwp	YColorLed FirstColorLedInContext(YAPIContext yctx)
es	function FirstColorLedInContext(yctx)

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

Paramètres :

yctx un contexte YAPI.

Retourne :

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

colorled→addHslMoveToBlinkSeq()**YColorLed**

Ajoute une transition à la séquence de clignotement du module, la transition s'effectuera dans l'espace de couleur HSL.

```

js function addHslMoveToBlinkSeq( HSLcolor, msDelay)
nodejs function addHslMoveToBlinkSeq( HSLcolor, msDelay)
cpp int addHslMoveToBlinkSeq( int HSLcolor, int msDelay)
m -(int) addHslMoveToBlinkSeq : (int) HSLcolor
: (int) msDelay
pas function addHslMoveToBlinkSeq( HSLcolor: LongInt,
msDelay: LongInt): LongInt
vb function addHslMoveToBlinkSeq( ) As Integer
cs int addHslMoveToBlinkSeq( int HSLcolor, int msDelay)
java int addHslMoveToBlinkSeq( int HSLcolor, int msDelay)
uwp async Task<int> addHslMoveToBlinkSeq( int HSLcolor, int msDelay)
py def addHslMoveToBlinkSeq( HSLcolor, msDelay)
php function addHslMoveToBlinkSeq( $HSLcolor, $msDelay)
es function addHslMoveToBlinkSeq( HSLcolor, msDelay)
cmd YColorLed target addHslMoveToBlinkSeq HSLcolor msDelay

```

Paramètres :

HSLcolor couleur HSL désirée à la fin de la transition
msDelay 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.

colored→addRgbMoveToBlinkSeq()

YColorLed

Ajoute une transition à la séquence de clignotement du module, la transition s'effectuera dans l'espace de couleur RGB

js	function addRgbMoveToBlinkSeq(RGBcolor, msDelay,
nodejs	function addRgbMoveToBlinkSeq(RGBcolor, msDelay)
cpp	int addRgbMoveToBlinkSeq(int RGBcolor, int msDelay)
m	-(int) addRgbMoveToBlinkSeq : (int) RGBcolor : (int) msDelay
pas	function addRgbMoveToBlinkSeq(RGBcolor: LongInt, msDelay: LongInt): LongInt
vb	function addRgbMoveToBlinkSeq() As Integer
cs	int addRgbMoveToBlinkSeq(int RGBcolor, int msDelay)
java	int addRgbMoveToBlinkSeq(int RGBcolor, int msDelay)
uwp	async Task<int> addRgbMoveToBlinkSeq(int RGBcolor, int msDelay)
py	def addRgbMoveToBlinkSeq(RGBcolor, msDelay)
php	function addRgbMoveToBlinkSeq(\$RGBcolor, \$msDelay)
es	function addRgbMoveToBlinkSeq(RGBcolor, msDelay)
cmd	YColorLed target addRgbMoveToBlinkSeq RGBcolor msDelay

Paramètres :

RGBcolor couleur RGB désirée à la fin de la transition
msDelay 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.

colorled→clearCache()**YColorLed**

Invalide le cache.

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

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

colorled→describe()**YColorLed**

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

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

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

Retourne :

une chaîne de caractères décrivant la LED RGB (ex:
`Relay(MyCustomName.relay1)=RELAYL01-123456.relay1`)

colored→**get_advertisedValue()****YColorLed****colored**→**advertisedValue()**

Retourne la valeur courante de la LED RGB (pas plus de 6 caractères).

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

Retourne :

une chaîne de caractères représentant la valeur courante de la LED RGB (pas plus de 6 caractères).

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

colorled→get_blinkSeqMaxSize()

colorled→blinkSeqMaxSize()

YColorLed

Retourne la longueur maximum de la sequence de clignotement.

js	function get_blinkSeqMaxSize ()
nodejs	function get_blinkSeqMaxSize ()
cpp	int get_blinkSeqMaxSize ()
m	-(int) blinkSeqMaxSize
pas	function get_blinkSeqMaxSize (): LongInt
vb	function get_blinkSeqMaxSize () As Integer
cs	int get_blinkSeqMaxSize ()
java	int get_blinkSeqMaxSize ()
uwp	async Task<int> get_blinkSeqMaxSize ()
py	def get_blinkSeqMaxSize ()
php	function get_blinkSeqMaxSize ()
es	function get_blinkSeqMaxSize ()
cmd	YColorLed target get_blinkSeqMaxSize

Retourne :

un entier représentant la longueur maximum de la sequence de clignotement

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

colorled→get_blinkSeqSignature()**YColorLed****colorled→blinkSeqSignature()**

Retourne la signature de la signature de la séquence de clignotement.

js	function get_blinkSeqSignature ()
nodejs	function get_blinkSeqSignature ()
cpp	int get_blinkSeqSignature ()
m	-(int) blinkSeqSignature
pas	function get_blinkSeqSignature (): LongInt
vb	function get_blinkSeqSignature () As Integer
cs	int get_blinkSeqSignature ()
java	int get_blinkSeqSignature ()
uwp	async Task<int> get_blinkSeqSignature ()
py	def get_blinkSeqSignature ()
php	function get_blinkSeqSignature ()
es	function get_blinkSeqSignature ()
cmd	YColorLed target get_blinkSeqSignature

Les séquences de clignotement ne pouvant pas être relues du module, ce mécanisme peut être utilisé pour détecter si une séquence spécifique est déjà programmée.

Retourne :

un entier représentant la signature de la signature de la séquence de clignotement

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

colorled→get_blinkSeqSize() colorled→blinkSeqSize()

YColorLed

Retourne la longueur actuelle de la sequence de clignotement.

js	function get_blinkSeqSize ()
nodejs	function get_blinkSeqSize ()
cpp	int get_blinkSeqSize ()
m	-(int) blinkSeqSize
pas	function get_blinkSeqSize (): LongInt
vb	function get_blinkSeqSize () As Integer
cs	int get_blinkSeqSize ()
java	int get_blinkSeqSize ()
uwp	async Task<int> get_blinkSeqSize ()
py	def get_blinkSeqSize ()
php	function get_blinkSeqSize ()
es	function get_blinkSeqSize ()
cmd	YColorLed target get_blinkSeqSize

Retourne :

un entier représentant la longueur actuelle de la sequence de clignotement

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

colorled→get_errorMessage()**YColorLed****colorled→errorMessage()**

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

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

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

Retourne :

une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la LED RGB.

colorled→get_errorType() colorled→errorType()

YColorLed

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

js	function get_errorType ()
nodejs	function get_errorType ()
cpp	YRETCODE get_errorType ()
pas	function get_errorType (): YRETCODE
vb	function get_errorType () As YRETCODE
cs	YRETCODE get_errorType ()
java	int get_errorType ()
py	def get_errorType ()
php	function get_errorType ()
es	function get_errorType ()

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

Retourne :

un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation de la LED RGB.

colorled→**get_friendlyName()****YColorLed****colorled**→**friendlyName()**

Retourne un identifiant global de la LED RGB au format NOM_MODULE . NOM_FONCTION.

js	function get_friendlyName ()
nodejs	function get_friendlyName ()
cpp	string get_friendlyName ()
m	-(NSString*) friendlyName
cs	string get_friendlyName ()
java	String get_friendlyName ()
py	def get_friendlyName ()
php	function get_friendlyName ()
es	function get_friendlyName ()

Le chaîne retournée utilise soit les noms logiques du module et de la LED RGB si ils sont définis, soit respectivement le numéro de série du module et l'identifiant matériel de la LED RGB (par exemple: `MyCustomName.relay1`)

Retourne :

une chaîne de caractères identifiant la LED RGB en utilisant les noms logiques (ex: `MyCustomName.relay1`)

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

colorled→get_functionDescriptor() colorled→functionDescriptor()

YColorLed

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

js	function get_functionDescriptor ()
nodejs	function get_functionDescriptor ()
cpp	YFUN_DESCR get_functionDescriptor ()
m	-(YFUN_DESCR) functionDescriptor
pas	function get_functionDescriptor (): YFUN_DESCR
vb	function get_functionDescriptor () As YFUN_DESCR
cs	YFUN_DESCR get_functionDescriptor ()
java	String get_functionDescriptor ()
py	def get_functionDescriptor ()
php	function get_functionDescriptor ()
es	function get_functionDescriptor ()

Cet identifiant peut être utilisé pour tester si deux instance de YFunction référencent physiquement la même fonction sur le même module.

Retourne :

un identifiant de type YFUN_DESCR.

Si la fonction n'a jamais été contactée, la valeur retournée sera Y_FUNCTIONDESCRIPTOR_INVALID

colorled→**get_functionId()****YColorLed****colorled**→**functionId()**

Retourne l'identifiant matériel de la LED RGB, sans référence au module.

js	function get_functionId ()
nodejs	function get_functionId ()
cpp	string get_functionId ()
m	-(NSString*) functionId
vb	function get_functionId () As String
cs	string get_functionId ()
java	String get_functionId ()
py	def get_functionId ()
php	function get_functionId ()
es	function get_functionId ()

Par exemple `relay1`.

Retourne :

une chaîne de caractères identifiant la LED RGB (ex: `relay1`)

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

colorled→**get_hardwareId()**
colorled→**hardwareId()**

YColorLed

Retourne l'identifiant matériel unique de la LED RGB au format SERIAL . FUNCTIONID.

js	function get_hardwareId ()
nodejs	function get_hardwareId ()
cpp	string get_hardwareId ()
m	-(NSString*) hardwareId
vb	function get_hardwareId () As String
cs	string get_hardwareId ()
java	String get_hardwareId ()
py	def get_hardwareId ()
php	function get_hardwareId ()
es	function get_hardwareId ()

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la LED RGB (par exemple RELAYL01-123456 . relay1).

Retourne :

une chaîne de caractères identifiant la LED RGB (ex: RELAYL01-123456 . relay1)

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

colorled→get_hslColor()
colorled→hslColor()**YColorLed**

Retourne la couleur HSL courante de la LED.

js	function get_hslColor ()
nodejs	function get_hslColor ()
cpp	int get_hslColor ()
m	-(int) hslColor
pas	function get_hslColor (): LongInt
vb	function get_hslColor () As Integer
cs	int get_hslColor ()
java	int get_hslColor ()
uwp	async Task<int> get_hslColor ()
py	def get_hslColor ()
php	function get_hslColor ()
es	function get_hslColor ()
cmd	YColorLed target get_hslColor

Retourne :

un entier représentant la couleur HSL courante de la LED

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

colorled→get_logicalName() colorled→logicalName()

YColorLed

Retourne le nom logique de la LED RGB.

js	function get_logicalName ()
nodejs	function get_logicalName ()
cpp	string get_logicalName ()
m	-(NSString*) logicalName
pas	function get_logicalName (): string
vb	function get_logicalName () As String
cs	string get_logicalName ()
java	String get_logicalName ()
uwp	async Task<string> get_logicalName ()
py	def get_logicalName ()
php	function get_logicalName ()
es	function get_logicalName ()
cmd	YColorLed target get_logicalName

Retourne :

une chaîne de caractères représentant le nom logique de la LED RGB.

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

colorled→get_module()
colorled→module()**YColorLed**

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

js	function get_module ()
nodejs	function get_module ()
cpp	YModule * get_module ()
m	-(YModule*) module
pas	function get_module (): TYModule
vb	function get_module () As YModule
cs	YModule get_module ()
java	YModule get_module ()
py	def get_module ()
php	function get_module ()
es	function get_module ()

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

Retourne :

une instance de YModule

colorled→**get_module_async()**
colorled→**module_async()****YColorLed**

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

js	function get_module_async (callback , context)
nodejs	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.

colorled→get_rgbColor()
colorled→rgbColor()**YColorLed**

Retourne la couleur RGB courante de la LED.

js	function get_rgbColor ()
nodejs	function get_rgbColor ()
cpp	int get_rgbColor ()
m	-(int) rgbColor
pas	function get_rgbColor (): LongInt
vb	function get_rgbColor () As Integer
cs	int get_rgbColor ()
java	int get_rgbColor ()
uwp	async Task<int> get_rgbColor ()
py	def get_rgbColor ()
php	function get_rgbColor ()
es	function get_rgbColor ()
cmd	YColorLed target get_rgbColor

Retourne :

un entier représentant la couleur RGB courante de la LED

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

colorled→get_rgbColorAtPowerOn() colorled→rgbColorAtPowerOn()

YColorLed

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

js	function get_rgbColorAtPowerOn ()
nodejs	function get_rgbColorAtPowerOn ()
cpp	int get_rgbColorAtPowerOn ()
m	-(int) rgbColorAtPowerOn
pas	function get_rgbColorAtPowerOn (): LongInt
vb	function get_rgbColorAtPowerOn () As Integer
cs	int get_rgbColorAtPowerOn ()
java	int get_rgbColorAtPowerOn ()
uwp	async Task<int> get_rgbColorAtPowerOn ()
py	def get_rgbColorAtPowerOn ()
php	function get_rgbColorAtPowerOn ()
es	function get_rgbColorAtPowerOn ()
cmd	YColorLed target get_rgbColorAtPowerOn

Retourne :

un entier représentant la couleur configurée pour être affichage à l'allumage du module

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

colorled→**get_userdata()****YColorLed****colorled**→**userData()**

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

js	function get_userdata ()
nodejs	function get_userdata ()
cpp	void * get_userdata ()
m	-(id) <code>userData</code>
pas	function get_userdata (): Tobject
vb	function get_userdata () As Object
cs	object get_userdata ()
java	Object get_userdata ()
py	def get_userdata ()
php	function get_userdata ()
es	function get_userdata ()

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

Retourne :

l'objet stocké précédemment par l'appelant.

colorled→hslMove()**YColorLed**

Effectue une transition continue dans l'espace HSL entre la couleur courante et une nouvelle couleur.

js	function hslMove (hsl_target , ms_duration)
nodejs	function hslMove (hsl_target , ms_duration)
cpp	int hslMove (int hsl_target , int ms_duration)
m	-(int) hslMove : (int) hsl_target : (int) ms_duration
pas	function hslMove (hsl_target : LongInt, ms_duration : LongInt): integer
vb	function hslMove (ByVal hsl_target As Integer, ByVal ms_duration As Integer) As Integer
cs	int hslMove (int hsl_target , int ms_duration)
java	int hslMove (int hsl_target , int ms_duration)
py	def hslMove (hsl_target , ms_duration)
php	function hslMove (\$ hsl_target , \$ ms_duration)
es	function hslMove (hsl_target , ms_duration)
cmd	YColorLed target hslMove hsl_target ms_duration

Paramètres :

hsl_target couleur HSL désirée à la fin de la transition
ms_duration durée de la transition, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colored→isOnline()**YColorLed**

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

js	function isOnline ()
nodejs	function isOnline ()
cpp	bool isOnline ()
m	-(BOOL) isOnline
pas	function isOnline (): boolean
vb	function isOnline () As Boolean
cs	bool isOnline ()
java	boolean isOnline ()
py	def isOnline ()
php	function isOnline ()
es	function isOnline ()

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

Retourne :

true si la LED RGB est joignable, false sinon

colorled→isOnline_async()**YColorLed**

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

```
js  function isOnline_async( callback, context)
nodejs function isOnline_async( callback, context)
```

Si les valeurs des attributs en cache de la LED RGB 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.

colored→load()**YColorLed**

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

js	function load (msValidity)
nodejs	function load (msValidity)
cpp	YRETCODE load (int msValidity)
m	-(YRETCODE) load : (int) msValidity
pas	function load (msValidity : integer): YRETCODE
vb	function load (ByVal msValidity As Integer) As YRETCODE
cs	YRETCODE load (ulong msValidity)
java	int load (long msValidity)
py	def load (msValidity)
php	function load (\$msValidity)
es	function load (msValidity)

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→loadAttribute()**YColorLed**

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)
nodejs	function loadAttribute (attrName)
cpp	string loadAttribute (string attrName)
m	-(NSString*) loadAttribute : (NSString*) attrName
pas	function loadAttribute (attrName : string): string
vb	function loadAttribute () As String
cs	string loadAttribute (string attrName)
java	String loadAttribute (String attrName)
uwp	async Task<string> loadAttribute (string attrName)
py	def loadAttribute (attrName)
php	function loadAttribute (\$attrName)
es	function loadAttribute (attrName)

Paramètres :

attrName le nom de l'attribut désiré

Retourne :

une chaîne de caractères représentant la valeur actuelle de l'attribut.

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

colorled→load_async()**YColorLed**

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

```
js  function load_async( msValidity, callback, context)
nodejs 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-qu'el à la fonction de callback

Retourne :

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

colorled→muteValueCallbacks()**YColorLed**

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

js	function muteValueCallbacks ()
nodejs	function muteValueCallbacks ()
cpp	int muteValueCallbacks ()
m	-(int) muteValueCallbacks
pas	function muteValueCallbacks (): LongInt
vb	function muteValueCallbacks () As Integer
cs	int muteValueCallbacks ()
java	int muteValueCallbacks ()
uwp	async Task<int> muteValueCallbacks ()
py	def muteValueCallbacks ()
php	function muteValueCallbacks ()
es	function muteValueCallbacks ()
cmd	YColorLed 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.

colorled→nextColorLed()**YColorLed**

Continue l'énumération des LEDs RGB commencée à l'aide de `yFirstColorLed()`.

js	function nextColorLed ()
nodejs	function nextColorLed ()
cpp	YColorLed * nextColorLed ()
m	-(YColorLed*) nextColorLed
pas	function nextColorLed (): TYColorLed
vb	function nextColorLed () As YColorLed
cs	YColorLed nextColorLed ()
java	YColorLed nextColorLed ()
uwp	YColorLed nextColorLed ()
py	def nextColorLed ()
php	function nextColorLed ()
es	function nextColorLed ()

Retourne :

un pointeur sur un objet `YColorLed` accessible en ligne, ou `null` lorsque l'énumération est terminée.

colorled→registerValueCallback()**YColorLed**

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

js	function registerValueCallback (callback)
nodejs	function registerValueCallback (callback)
cpp	int registerValueCallback (YColorLedValueCallback callback)
m	-(int) registerValueCallback : (YColorLedValueCallback) callback
pas	function registerValueCallback (callback : TYColorLedValueCallback): LongInt
vb	function registerValueCallback () As Integer
cs	int registerValueCallback (ValueCallback callback)
java	int registerValueCallback (UpdateCallback callback)
uwp	async Task<int> registerValueCallback (ValueCallback callback)
py	def registerValueCallback (callback)
php	function registerValueCallback (\$callback)
es	function registerValueCallback (callback)

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'objet fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

colorled→resetBlinkSeq()**YColorLed**

efface le contenu de la sequence de clignotement.

js	function resetBlinkSeq ()
nodejs	function resetBlinkSeq ()
cpp	int resetBlinkSeq ()
m	-(int) resetBlinkSeq
pas	function resetBlinkSeq (): LongInt
vb	function resetBlinkSeq () As Integer
cs	int resetBlinkSeq ()
java	int resetBlinkSeq ()
uwp	async Task<int> resetBlinkSeq ()
py	def resetBlinkSeq ()
php	function resetBlinkSeq ()
es	function resetBlinkSeq ()
cmd	YColorLed target resetBlinkSeq

Retourne :

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

colorled→rgbMove()**YColorLed**

Effectue une transition continue dans l'espace RGB entre la couleur courante et une nouvelle couleur.

js	function rgbMove (rgb_target , ms_duration)
nodejs	function rgbMove (rgb_target , ms_duration)
cpp	int rgbMove (int rgb_target , int ms_duration)
m	-(int) rgbMove : (int) rgb_target : (int) ms_duration
pas	function rgbMove (rgb_target : LongInt, ms_duration : LongInt): integer
vb	function rgbMove (ByVal rgb_target As Integer, ByVal ms_duration As Integer) As Integer
cs	int rgbMove (int rgb_target , int ms_duration)
java	int rgbMove (int rgb_target , int ms_duration)
py	def rgbMove (rgb_target , ms_duration)
php	function rgbMove (\$ rgb_target , \$ ms_duration)
es	function rgbMove (rgb_target , ms_duration)
cmd	YColorLed target rgbMove rgb_target ms_duration

Paramètres :

rgb_target couleur RGB désirée à la fin de la transition
ms_duration durée de la transition, en millisecondes

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→set_hslColor() **colorled→setHslColor()**

YColorLed

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

js	function set_hslColor (newval)
nodejs	function set_hslColor (newval)
cpp	int set_hslColor (int newval)
m	-(int) setHslColor : (int) newval
pas	function set_hslColor (newval : LongInt): integer
vb	function set_hslColor (ByVal newval As Integer) As Integer
cs	int set_hslColor (int newval)
java	int set_hslColor (int newval)
uwp	async Task<int> set_hslColor (int newval)
py	def set_hslColor (newval)
php	function set_hslColor (\$newval)
es	function set_hslColor (newval)
cmd	YColorLed target set_hslColor newval

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→set_logicalName() colorled→setLogicalName()

YColorLed

Modifie le nom logique de la LED RGB.

js	function set_logicalName (newval)
nodejs	function set_logicalName (newval)
cpp	int set_logicalName (const string& newval)
m	-(int) setLogicalName : (NSString*) newval
pas	function set_logicalName (newval : string): integer
vb	function set_logicalName (ByVal newval As String) As Integer
cs	int set_logicalName (string newval)
java	int set_logicalName (String newval)
uwp	async Task<int> set_logicalName (string newval)
py	def set_logicalName (newval)
php	function set_logicalName (\$ newval)
es	function set_logicalName (newval)
cmd	YColorLed 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 la LED RGB.

Retourne :

YAPI_SUCCESS si l'appel se déroule sans erreur.

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

colorled→set_rgbColor() **colorled→setRgbColor()**

YColorLed

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

js	function set_rgbColor (newval)
nodejs	function set_rgbColor (newval)
cpp	int set_rgbColor (int newval)
m	-(int) setRgbColor : (int) newval
pas	function set_rgbColor (newval : LongInt): integer
vb	function set_rgbColor (ByVal newval As Integer) As Integer
cs	int set_rgbColor (int newval)
java	int set_rgbColor (int newval)
uwp	async Task<int> set_rgbColor (int newval)
py	def set_rgbColor (newval)
php	function set_rgbColor (\$newval)
es	function set_rgbColor (newval)
cmd	YColorLed target set_rgbColor newval

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

Paramètres :

newval un entier représentant la couleur courante de la LED, en utilisant une couleur RGB (Rouge Vert Bleu)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colorled→set_rgbColorAtPowerOn() colorled→setRgbColorAtPowerOn()

YColorLed

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

js	function set_rgbColorAtPowerOn(newval)
nodejs	function set_rgbColorAtPowerOn(newval)
cpp	int set_rgbColorAtPowerOn(int newval)
m	-(int) setRgbColorAtPowerOn : (int) newval
pas	function set_rgbColorAtPowerOn(newval: LongInt): integer
vb	function set_rgbColorAtPowerOn(ByVal newval As Integer) As Integer
cs	int set_rgbColorAtPowerOn(int newval)
java	int set_rgbColorAtPowerOn(int newval)
uwp	async Task<int> set_rgbColorAtPowerOn(int newval)
py	def set_rgbColorAtPowerOn(newval)
php	function set_rgbColorAtPowerOn(\$newval)
es	function set_rgbColorAtPowerOn(newval)
cmd	YColorLed target set_rgbColorAtPowerOn newval

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

colored→**set_userdata()****YColorLed****colored**→**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)
nodejs	function set_userdata (data)
cpp	void set_userdata (void* data)
m	-(void) setUserData : (id) data
pas	procedure set_userdata (data : Tobject)
vb	procedure set_userdata (ByVal data As Object)
cs	void set_userdata (object data)
java	void set_userdata (Object data)
py	def set_userdata (data)
php	function set_userdata (\$data)
es	function set_userdata (data)

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

Paramètres :

data objet quelconque à mémoriser

colorled→startBlinkSeq()**YColorLed**

Démarre l'exécution de la séquence préprogrammée de clignotement.

js	function startBlinkSeq ()
nodejs	function startBlinkSeq ()
cpp	int startBlinkSeq ()
m	-(int) startBlinkSeq
pas	function startBlinkSeq (): LongInt
vb	function startBlinkSeq () As Integer
cs	int startBlinkSeq ()
java	int startBlinkSeq ()
uwp	async Task<int> startBlinkSeq ()
py	def startBlinkSeq ()
php	function startBlinkSeq ()
es	function startBlinkSeq ()
cmd	YColorLed target startBlinkSeq

La séquence va tourner en boucle jusqu'à ce qu'elle soit stoppée par stopBlinkSeq ou un changement explicite.

Retourne :

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

colorled→stopBlinkSeq()**YColorLed**

Arrête l'exécution de la séquence préprogrammée de clignotement.

js	function stopBlinkSeq ()
nodejs	function stopBlinkSeq ()
cpp	int stopBlinkSeq ()
m	-(int) stopBlinkSeq
pas	function stopBlinkSeq (): LongInt
vb	function stopBlinkSeq () As Integer
cs	int stopBlinkSeq ()
java	int stopBlinkSeq ()
uwp	async Task<int> stopBlinkSeq ()
py	def stopBlinkSeq ()
php	function stopBlinkSeq ()
es	function stopBlinkSeq ()
cmd	YColorLed target stopBlinkSeq

Retourne :

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

colorled→unmuteValueCallbacks()**YColorLed**

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

js	function unmuteValueCallbacks ()
nodejs	function unmuteValueCallbacks ()
cpp	int unmuteValueCallbacks ()
m	-(int) unmuteValueCallbacks
pas	function unmuteValueCallbacks (): LongInt
vb	function unmuteValueCallbacks () As Integer
cs	int unmuteValueCallbacks ()
java	int unmuteValueCallbacks ()
uwp	async Task<int> unmuteValueCallbacks ()
py	def unmuteValueCallbacks ()
php	function unmuteValueCallbacks ()
es	function unmuteValueCallbacks ()
cmd	YColorLed 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.

colorled→wait_async()**YColorLed**

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)
nodejs function wait_async( callback, context)
es function wait_async( callback, context)
```

La fonction callback peut donc librement utiliser des fonctions synchrones ou asynchrones, sans risquer de bloquer la machine virtuelle Javascript.

Paramètres :

callback fonction de callback qui sera appelée dès que toutes les commandes en cours d'exécution sur le module seront terminées La fonction callback reçoit deux arguments: le contexte fourni par l'appelant et l'objet fonction concerné.

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

Retourne :

rien du tout.

21. Problèmes courants

21.1. Linux et USB

Pour fonctionner correctement sous Linux la librairie a besoin d'avoir accès en écriture à tous les périphériques USB Yoctopuce. Or, par défaut, sous Linux les droits d'accès des utilisateurs non-root à USB sont limités à la lecture. Afin d'éviter de devoir lancer les exécutables en tant que root, il faut créer une nouvelle règle *udev* pour autoriser un ou plusieurs utilisateurs à accéder en écriture aux périphériques Yoctopuce.

Pour ajouter une règle *udev* à votre installation, il faut ajouter un fichier avec un nom au format "`##-nomArbitraire.rules`" dans le répertoire `/etc/udev/rules.d`. Lors du démarrage du système, *udev* va lire tous les fichiers avec l'extension `.rules` de ce répertoire en respectant l'ordre alphabétique (par exemple, le fichier `"51-custom.rules"` sera interprété APRES le fichier `"50-udev-default.rules"`).

Le fichier `"50-udev-default"` contient les règles *udev* par défaut du système. Pour modifier le comportement par défaut du système, il faut donc créer un fichier qui commence par un nombre plus grand que 50, qui définira un comportement plus spécifique que le défaut du système. Notez que pour ajouter une règle vous aurez besoin d'avoir un accès root sur le système.

Dans le répertoire `udev_conf` de l'archive du *VirtualHub*¹ 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.

```
# 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

¹ <http://www.yoctopuce.com/EN/virtualhub.php>

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"
```

21.2. 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.

21.3. Module alimenté mais invisible pour l'OS

Si votre Yocto-Color 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.

21.4. 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 ².

21.5. Déconnexions, comportement erratique

Si votre Yocto-Color 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 ^{3 4}.

21.6. Par où commencer ?

Si c'est la première fois que vous utilisez un module Yoctopuce et ne savez pas trop par où commencer, allez donc jeter un coup d'il sur le blog de Yoctopuce. Il y a une section dédiée aux débutants ⁵.

² voir: <http://www.yoctopuce.com/FR/article/message-d-erreur-another-process-is-already-using-yapi>

³ voir: <http://www.yoctopuce.com/FR/article/cables-usb-la-taille-compte>

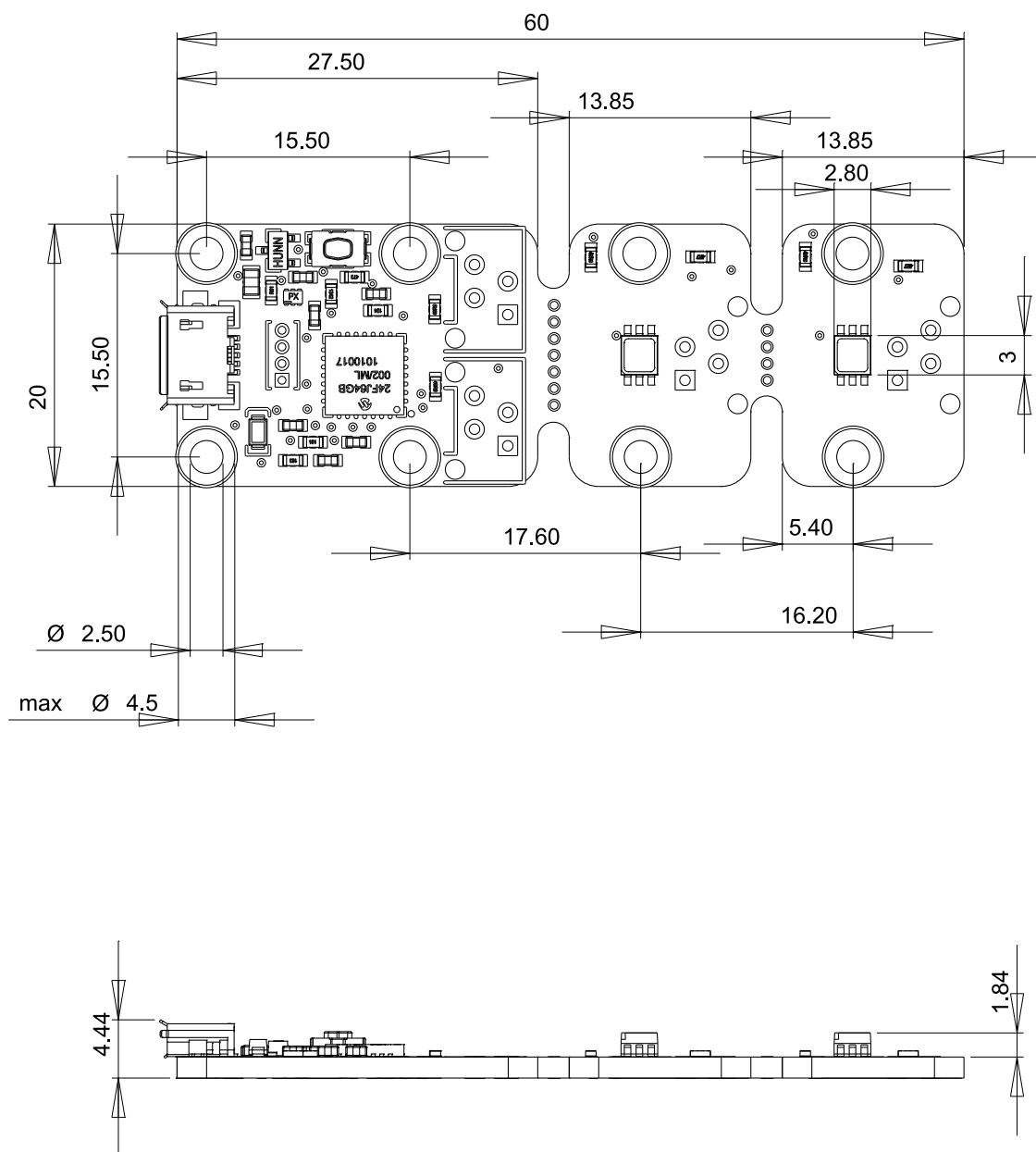
⁴ voir: <http://www.yoctopuce.com/FR/article/combien-de-capteurs-usb-peut-on-connecter>

⁵ voir: http://www.yoctopuce.com/FR/blog_by_categories/pour-les-debutants

22. Caractéristiques

Vous trouverez résumées ci dessous les principales caractéristiques techniques de votre module Yocto-Color

Largeur	20 mm
Longueur	60 mm
Poids	4 g
Connecteur USB	micro-B
Canaux	2
Système d'exploitation supportés	Windows, Linux (Intel + ARM), Mac OS X, Android
Drivers	Fonctionne sans driver
API / SDK / Librairie (USB+TCP)	C++, Objective-C, C#, VB .NET, Delphi, Python, Java/Android
API / SDK / Librairie (seul.TCP)	Javascript, Node.js, PHP, Java
RoHS	oui
USB Vendor ID	0x24E0
USB Device ID	0x0014
Cables et boîtiers	disponibles séparément



All dimensions are in mm
Toutes les dimensions sont en mm

Yocto-LED

A4

Scale
2:1
Echelle

Index

A

Accès 101
Accessoires 3
Activer 102
addHslMoveToBlinkSeq, YColorLed 223
addRgbMoveToBlinkSeq, YColorLed 224
Alimentation 14
Alimenté 266
Already 266
Android 101, 102, 115
Another 266
Application 115
Asynchrones 30
Avancée 113

B

Basic 65
Bloquantes 30
Blueprint 269

C

C# 73
C++ 51, 57
Callback 46
Caractéristiques 267
checkFirmware, YModule 160
CheckLogicalName, YAPI 127
clearCache, YColorLed 225
clearCache, YModule 161
ColorLed 19, 26, 33, 41, 51, 59, 66, 74, 81, 90, 95, 104, 217
Commande 25, 119
Commandes 115
Commencer 23, 266
Compatibilité 101
Comportement 266
Concepts 15
Configuration 8
Connectique 11
Contraintes 14
Contrôle 17, 26, 27, 33, 37, 41, 44, 51, 54, 59, 61, 66, 68, 74, 76, 81, 84, 90, 92, 95, 97, 104, 106, 154
Couleurs 20
Courants 265

D

Déconnexions 266
Delphi 81
Déporter 12
describe, YColorLed 226
describe, YModule 162

Description 25
DisableExceptions, YAPI 128
download, YModule 163
Dynamique 89
Dynamiques 121

E

EcmaScript 29, 31
Éléments 5, 6
EnableExceptions, YAPI 129
EnableUSBHost, YAPI 130
Erratique 266
Erreurs 39, 49, 56, 63, 70, 79, 86, 94, 100, 111
Espaces 20
Événements 113

F

Fichiers 89
Filtres 46
FindColorLed, YColorLed 219
FindColorLedInContext, YColorLed 220
FindModule, YModule 157
FindModuleInContext, YModule 158
Firmware 115
FirstColorLed, YColorLed 221
FirstColorLedInContext, YColorLed 222
FirstModule, YModule 159
Fixation 11
Fonctions 30, 126
FreeAPI, YAPI 131
functionBaseType, YModule 164
functionCount, YModule 165
functionId, YModule 166
functionName, YModule 167
functionType, YModule 168
functionValue, YModule 169

G

get_advertisedValue, YColorLed 227
get_allSettings, YModule 170
get_beacon, YModule 171
get_blinkSeqMaxSize, YColorLed 228
get_blinkSeqSignature, YColorLed 229
get_blinkSeqSize, YColorLed 230
get_errorMessage, YColorLed 231
get_errorMessage, YModule 172
get_errorType, YColorLed 232
get_errorType, YModule 173
get_firmwareRelease, YModule 174
get_friendlyName, YColorLed 233
get_functionDescriptor, YColorLed 234
get_functionId, YColorLed 235
get_functionIds, YModule 175

get_hardwareId, YColorLed 236
get_hardwareId, YModule 176
get_hslColor, YColorLed 237
get_icon2d, YModule 177
get_lastLogs, YModule 178
get_logicalName, YColorLed 238
get_logicalName, YModule 179
get_luminosity, YModule 180
get_module, YColorLed 239
get_module_async, YColorLed 240
get_parentHub, YModule 181
get_persistentSettings, YModule 182
get_productId, YModule 183
get_productName, YModule 184
get_productRelease, YModule 185
get_rebootCountdown, YModule 186
get_rgbColor, YColorLed 241
get_rgbColorAtPowerOn, YColorLed 242
get_serialNumber, YModule 187
get_subDevices, YModule 188
get_upTime, YModule 189
get_url, YModule 190
get_usbCurrent, YModule 191
get_userData, YColorLed 243
get_userData, YModule 192
get_userVar, YModule 193
GetAPIVersion, YAPI 132
GetTickCount, YAPI 133

H

HandleEvents, YAPI 134
hasFunction, YModule 194
hslMove, YColorLed 244
HTTP 46, 119
Hub 101

I

InitAPI, YAPI 135
Installation 25, 65, 73
Intégration 57
Interface 17, 19, 21, 154, 217
Introduction 1
Invisible 266
isOnline, YColorLed 245
isOnline, YModule 195
isOnline_async, YColorLed 246
isOnline_async, YModule 196

J

Java 95
JavaScript 29-31
Jour 115, 118

L

Langages 119
Leds 12
Librairie 31, 57, 89, 115, 116, 124

Librairies 121
Limitations 27
Linux 265
load, YColorLed 247
load, YModule 197
load_async, YColorLed 249
load_async, YModule 198
loadAttribute, YColorLed 248
Localisation 7
log, YModule 199

M

Mais 266
Mise 115, 118
Mode 118
Module 7, 17, 27, 37, 44, 54, 61, 68, 76, 84, 92, 97, 106, 154, 266
Montage 11
muteValueCallbacks, YColorLed 250

N

Named 266
Natif 101
Native 21
.NET 65
nextColorLed, YColorLed 251
nextModule, YModule 200
Niveau 124, 125

O

Objective-C 59

P

Paradigme 15
Plateformes 266
Port 102, 124
Pour 31, 266
Préparation 41, 81, 95, 101
PreregisterHub, YAPI 136
Prérequis 2
Présentation 5
Problèmes 265
Process 266
Programmation 15, 23, 113, 116
Projet 65, 73
Python 89

R

reboot, YModule 201
Référence 125
RegisterDeviceArrivalCallback, YAPI 137
RegisterDeviceRemovalCallback, YAPI 138
RegisterHub, YAPI 139
RegisterHubDiscoveryCallback, YAPI 141
registerLogCallback, YModule 202
RegisterLogFunction, YAPI 142

registerValueCallback, YColorLed 252
resetBlinkSeq, YColorLed 253
revertFromFlash, YModule 203
rgbMove, YColorLed 254

S

saveToFlash, YModule 204
SelectArchitecture, YAPI 143
Service 21
set_allSettings, YModule 205
set_allSettingsAndFiles, YModule 206
set_beacon, YModule 207
set_hslColor, YColorLed 255
set_logicalName, YColorLed 256
set_logicalName, YModule 208
set_luminosity, YModule 209
set_rgbColor, YColorLed 257
set_rgbColorAtPowerOn, YColorLed 258
set_userData, YColorLed 259
set_userData, YModule 210
set_userVar, YModule 211
SetDelegate, YAPI 144
SetTimeout, YAPI 145
SetUSBPacketAckMs, YAPI 146
Sleep, YAPI 147
Sources 89
startBlinkSeq, YColorLed 260
stopBlinkSeq, YColorLed 261
Supportés 119

T

Test 7
TestHub, YAPI 148
triggerFirmwareUpdate, YModule 212
TriggerHubDiscovery, YAPI 149

U

unmuteValueCallbacks, YColorLed 262
UnregisterHub, YAPI 150
UpdateDeviceList, YAPI 151
UpdateDeviceList_async, YAPI 152
updateFirmware, YModule 213
updateFirmwareEx, YModule 214
Utiliser 31

V

Virtual 101, 119

VirtualHub 115
Visual 65, 73
VisualBasic 65

W

wait_async, YColorLed 263
wait_async, YModule 215

Y

YAPI 266
yCheckLogicalName 127
YColorLed 219-263
yDisableExceptions 128
yEnableExceptions 129
yEnableUSBHost 130
yFindColorLed 219
yFindColorLedInContext 220
yFindModule 157
yFindModuleInContext 158
yFirstColorLed 221
yFirstColorLedInContext 222
yFirstModule 159
yFreeAPI 131
yGetAPIVersion 132
yGetTickCount 133
yHandleEvents 134
yInitAPI 135
YModule 157-215
Yocto-Color 17, 25, 29, 41, 51, 59, 65, 73, 81, 89, 95, 101
Yocto-Firmware 115
YoctoHub 115
yPreregisterHub 136
yRegisterDeviceArrivalCallback 137
yRegisterDeviceRemovalCallback 138
yRegisterHub 139
yRegisterHubDiscoveryCallback 141
yRegisterLogFunction 142
ySelectArchitecture 143
ySetDelegate 144
ySetTimeout 145
ySetUSBPacketAckMs 146
ySleep 147
yTestHub 148
yTriggerHubDiscovery 149
yUnregisterHub 150
yUpdateDeviceList 151
yUpdateDeviceList_async 152