Yocto-Demo, Mode d'emploi

Table des matières

1. Introduction **1**
- 1.1. Prérequis 1
- 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. Contraintes d'alimentation par USB 12

5. Programmation, concepts généraux **13**
- 5.1. Paradigme de programmation 13
- 5.2. Le module Yocto-Demo 15
- 5.3. Interface de contrôle du module 15
- 5.4. Interface de la fonction Led 16
- 5.5. Quelle interface: Native, DLL ou Service? 17
- 5.6. Programmation, par où commencer? 19

6. Utilisation du Yocto-Demo en ligne de commande **21**
- 6.1. Installation 21
- 6.2. Utilisation: description générale 21
- 6.3. Contrôle de la fonction Led 22
- 6.4. Contrôle de la partie module 23
- 6.5. Limitations 23

7. Utilisation du Yocto-Demo en Javascript **25**
- 7.1. Préparation 25
- 7.2. Contrôle de la fonction Led 25

7.3. Contrôle de la partie module 27
7.4. Gestion des erreurs 30

8. Utilisation du Yocto-Demo en PHP 33
8.1. Préparation 33
8.2. Contrôle de la fonction Led 33
8.3. Contrôle de la partie module 35
8.4. API par callback HTTP et filtres NAT 38
8.5. Gestion des erreurs 41

9. Utilisation du Yocto-Demo en C++ 43
9.1. Contrôle de la fonction Led 43
9.2. Contrôle de la partie module 45
9.3. Gestion des erreurs 47
9.4. Intégration de la librairie Yoctopuce en C++ 48

10. Utilisation du Yocto-Demo en Objective-C 51
10.1. Contrôle de la fonction Led 51
10.2. Contrôle de la partie module 53
10.3. Gestion des erreurs 55

11. Utilisation du Yocto-Demo en VisualBasic .NET 57
11.1. Installation 57
11.2. Utilisation l'API yoctopuce dans un projet Visual Basic 57
11.3. Contrôle de la fonction Led 58
11.4. Contrôle de la partie module 60
11.5. Gestion des erreurs 62

12. Utilisation du Yocto-Demo en C# 63
12.1. Installation 63
12.2. Utilisation l'API yoctopuce dans un projet Visual C# 63
12.3. Contrôle de la fonction Led 64
12.4. Contrôle de la partie module 66
12.5. Gestion des erreurs 68

13. Utilisation du Yocto-Demo en Delphi 71
13.1. Préparation 71
13.2. Contrôle de la fonction Led 71
13.3. Contrôle de la partie module 73
13.4. Gestion des erreurs 76

14. Utilisation du Yocto-Demo en Python 77
14.1. Fichiers sources 77
14.2. Librairie dynamique 77
14.3. Contrôle de la fonction Led 78
14.4. Contrôle de la partie module 79
14.5. Gestion des erreurs 81

15. Utilisation du Yocto-Demo en Java 83
15.1. Préparation 83
15.2. Contrôle de la fonction Led 83
15.3. Contrôle de la partie module 85
15.4. Gestion des erreurs 87

16. Utilisation du Yocto-Demo avec Android **89**
16.1. Accès Natif et Virtual Hub. 89
16.2. Préparation 89
16.3. Compatibilité 89
16.4. Activer le port USB sous Android 90
16.5. Contrôle de la fonction Led 92
16.6. Contrôle de la partie module 94
16.7. Gestion des erreurs 99

17. Programmation avancée **101**
17.1. Programmation par événements 101

18. Utilisation avec des langages non supportés **103**
18.1. Ligne de commande 103
18.2. Virtual Hub et HTTP GET 103
18.3. Utilisation des librairies dynamiques 105
18.4. Port de la librairie haut niveau 108

19. Référence de l'API de haut niveau **109**
19.1. Fonctions générales 110
19.2. Interface de contrôle du module 136
19.3. Interface de la fonction Led 186

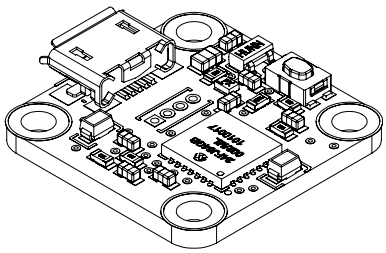
20. Problèmes courants **217**
20.1. Linux et USB 217
20.2. Plateformes ARM: HF et EL 218

21. Caractéristiques **219**
Blueprint 221

Index **223**

1. Introduction

Le module Yocto-Demo est un petit module de 20x20mm qui permet de commander une petite led verte par USB. En soit, commander un simple led par USB n'a pas grand intérêt. Le but visé par ce module est de vous familiariser avec les API de programmation de des modules Yoctopuce. En effet une fois que vous saurez piloter ce module il y a de de grandes chances que vous sachiez comment piloter les autre modules de la gamme.



Le module Yocto-Demo

Yoctopuce vous remercie d'avoir fait l'acquisition de ce Yocto-Demo et espère sincèrement qu'il vous donnera entière satisfaction. Les ingénieurs Yoctopuce se sont donnés beaucoup de mal pour que votre Yocto-Demo 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 Yoctopuce[1].

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.

1.1. Prérequis

Pour pouvoir profiter pleinement de votre module Yocto-Demo, 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.

[1] support@yoctopuce.com

Yoctopuce fourni les librairies 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 HID[2] fourni en standard dans tous les systèmes d'exploitation.

Les versions de Windows actuellement supportées sont Windows XP, Windows 2003, Windows Vista, Windows 7 et Windows 8.1. Les versions 32 bit et 64 bit sont supportées. Yoctopuce teste régulièrement le bon fonctionnement des modules sur Windows XP et Windows 7.

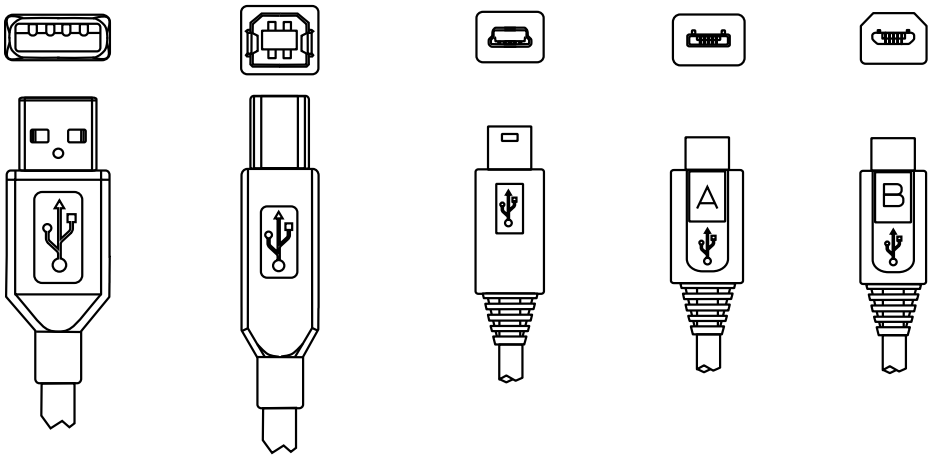
Les versions de Mac OS X actuellement supportées sont Mac OS X 10.6 (Snow Leopard), 10.7 (Lion), 10.8 (Mountain Lion) et 10.9 (Maverick). Yoctopuce teste régulièrement le bon fonctionnement des modules sur Mac OS X 10.9 et 10.7.

Les versions de Linux supportées sont les kernels 2.6 et 3.0. D'autre versions du kernel et même d'autres variantes d'Unix sont très susceptibles d'être utilisées sans problème, puisque le support de Linux est fait via l'API standard de la **libusb**, disponible aussi pour FreeBSD par exemple. Yoctopuce teste régulièrement le bon fonctionnement des modules sur un kernel Linux 2.6.

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 cable USB de type A-micro B

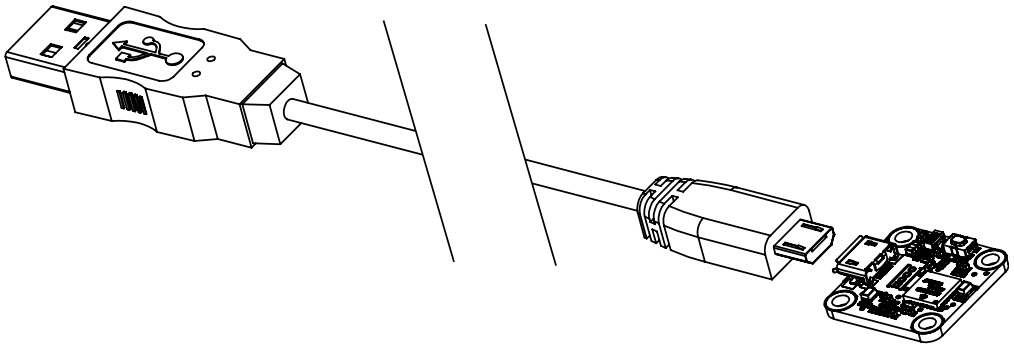
Il existe trois tailles de connecteurs USB, la taille "normale" que vous utilisez probablement pour brancher votre imprimante, la taille mini encore très courante et enfin la taille micro, souvent utilisée pour raccorder les téléphones portables, pour autant qu'ils n'arborent pas une pomme. Les modules 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]

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

[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*



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

Si vous branchez un hub USB entre l'ordinateur et le module Yocto-Demo, 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-Demo, 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-Demo à un support, vous pouvez placer des petites vis de 2.5mm avec une tête de 4.5mm au maximum dans les trous prévus ad-hoc. Il est conseillé de les visser dans des entretoises filetées, que vous pourrez fixer sur le support. Vous trouverez plus de détail à ce sujet dans le chapitre concernant le montage et la connectique.

Micro-hub USB

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

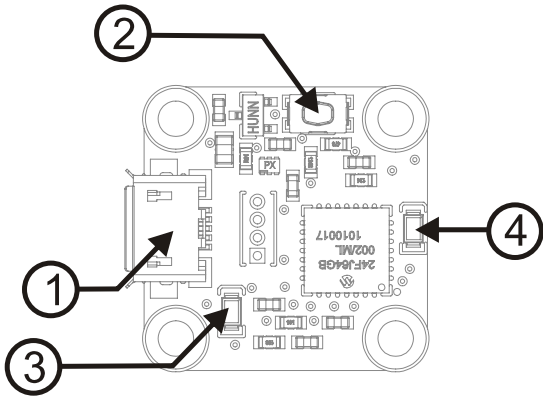
YoctoHub-Ethernet et YoctoHub-Wireless

Vous pouvez ajouter une connectivité réseau à votre Yocto-Demo grâce aux hubs YoctoHub-Ethernet et YoctoHub-Wireless. Le premier offre une connectivité Ethernet, le second offre une connectivité Wifi. Chacun de ces hubs peut piloter jusqu'à trois modules Yoctopuce et se comporte exactement comme un ordinateur normal qui ferait tourner un *VirtualHub*.

Cable nappe mono-brin

Si vous désirez souder le module Yocto-Demo directement à un micro-hub USB pour éviter l'encombrement d'un vrai cable USB, 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.

2. Présentation



1: prise USB 3: Yocto-led
2: Yocto-bouton 4: led de test

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 morse[1]. Si cela arrivait débranchez puis rebranchez le module. Si le problème venait à se reproduire vérifiez que le module contient bien la dernière version du firmware, et dans l'affirmative contactez le support Yoctopuce[2].

La sonde de courant

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

Le numéro de série

Chaque Yocto-module a un numéro de série unique attribué en usine, pour les modules Yocto-Demo ce numéro commence par YCTOPOC1. Le module peut être piloté par logiciel en utilisant ce numéro de série. Ce numéro de série ne peut pas être changé.

Le nom logique

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

2.2. Les éléments spécifiques

La led de test

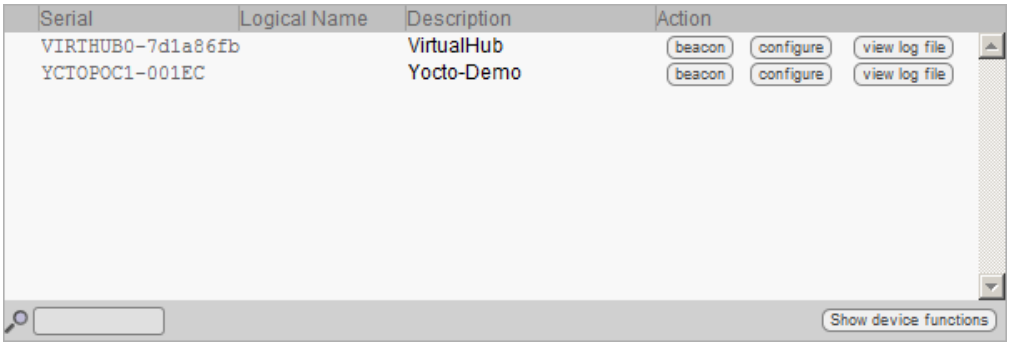
Le module Yocto-Demo ne sert pratiquement à rien, il est simplement conçu pour vous permettre de tester les API de programmation de Yoctopuce. C'est pourquoi le Yocto-Demo est un module de base équipé d'une seule LED verte en plus de la YoctoLED. Une fois que vous aurez compris comment piloter cette LED depuis votre langage de programmation préféré, il y a de bonnes chances pour que vous sachiez contrôler n'importe quel autre module Yoctopuce.

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

3. Premiers pas

Arrivé à ce chapitre votre Yocto-Demo 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.



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

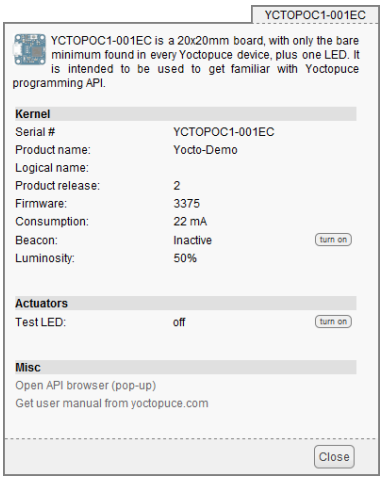
3.1. Localisation

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-Demo.

[1] www.yoctopuce.com/FR/virtualhub.php
[2] L'interface a été testée avec FireFox 3+, IE 6+, Safari et Chrome.

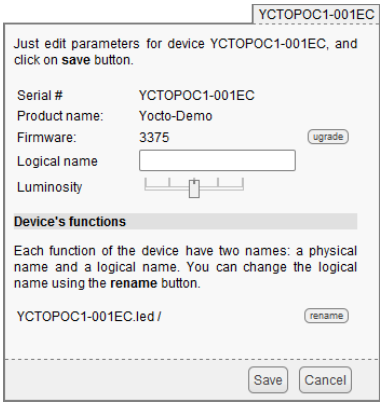


Propriétés du module Yocto-Demo.

Cette fenêtre vous permet entre autres de jouer avec la led de test. Vous remarquerez que la consommation du module varie légèrement en fonction de l'état de la led de test.

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-Demo.

Firmware

Le firmware du module peut être facilement être 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 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 commande [4].

Nom logique du module

Le nom logique est un nom choisi par vous, qui vous permettra d'accéder à votre module, de la même matière qu'un nom de fichier vous permet d'accéder à son contenu. Un nom logique doit faire

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

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 plus un peu 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

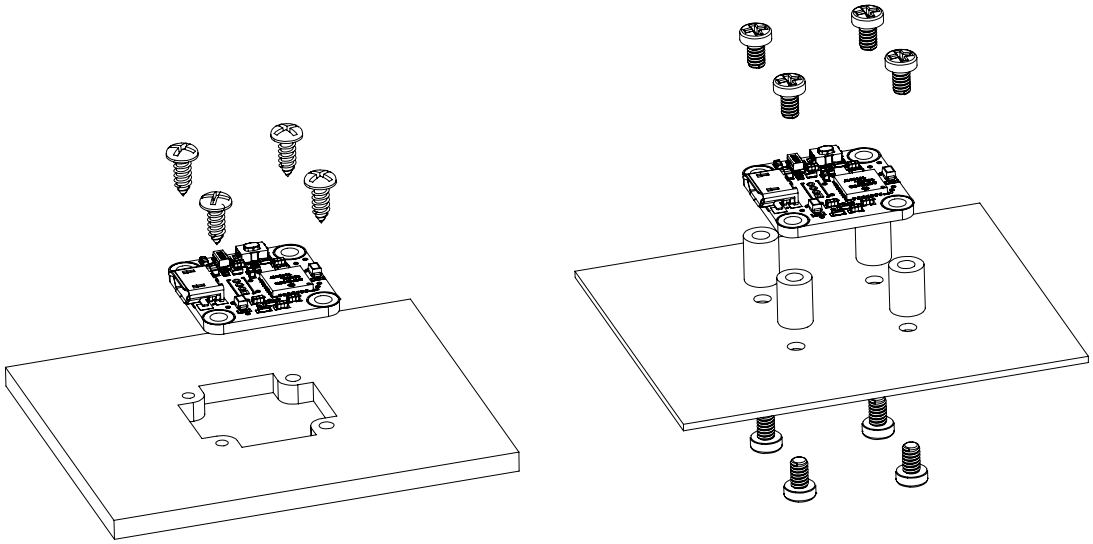
La seule fonction du module Yocto-Demo est la led de test, dont le nom hardware est `led`.

4. Montage et connectique

Ce chapitre fournit des explications importantes pour utiliser votre module Yocto-Demo 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-Demo 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 très joli mais ça tiendra.

Si vous comptez visser votre module directement contre une paroi conductrice, un chassis métallique par exemple, intercalez une couche isolante entre les deux. Sinon vous aller à coup sûr

provoquer un court-circuit: il y a des pads à nu sous votre module. Du simple ruban adhésif d'emballage devrait faire l'affaire.

4.2. Contraintes d'alimentation par USB

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

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

En pratique, USB n'aurait pas eu le succès qu'il a si il était si contraignant. Il se trouve que par économie, les fabricants de hubs omettent presque toujours d'implémenter la limitation de courant sur les ports: ils se contentent de connecter l'alimentation de tous les ports directement à l'ordinateur, tout en se déclarant comme *hub alimenté* même lorsqu'ils ne le sont pas (afin de désactiver tous les contrôles de consommation dans le système d'exploitation). C'est assez malpropre, mais dans la mesure ou 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.

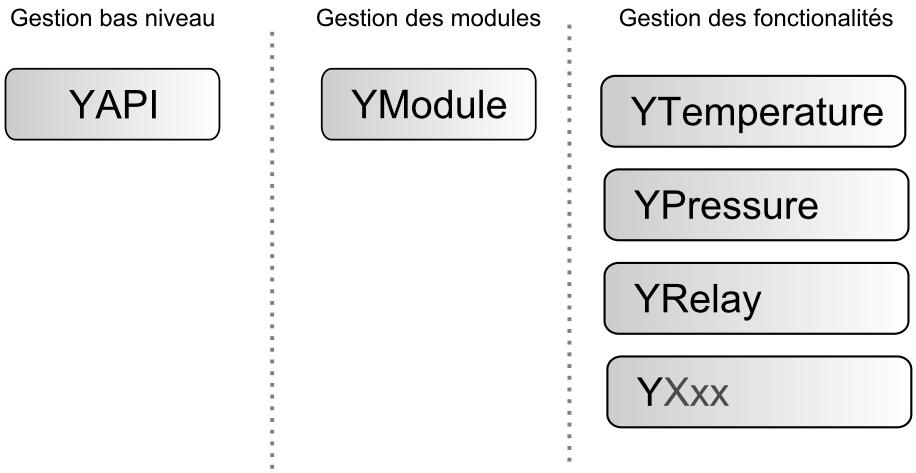
5. Programmation, concepts généraux

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-Demo 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.

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-Demo 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-Demo

Le module Yocto-Demo n'offre qu'une seule instance de la fonction Led, correspondant à l'unique led de test présente sur le module.

module : Module

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

led : Led

attribut	type	modifiable ?
logicalName	Texte	modifiable
advertisedValue	Texte	lecture seule
power	On/Off	modifiable
luminosity	0..100%	modifiable
blinking	Type énuméré	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

Chaine de caractères contenant le numéro de série, unique et préprogrammé en usine. Pour un module Yocto-Demo, ce numéro de série commence toujours par YCTOPOC1. Il peut servir comme point de départ pour accéder par programmation à un module particulier.

logicalName

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

productId

Identifiant USB du module, préprogrammé à la valeur 9 en usine.

productRelease

Numéro de révision du module hardware, preprogrammed at the factory.

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 Led

La librairie de programmation Yoctopuce permet non seulement d'allumer la led à une intensité donnée, mais aussi de la faire osciller à plusieurs fréquences.

logicalName

Chaîne de caractères contenant le nom logique de la led, initialement vide. Cet attribut peut être changé au bon vouloir de l'utilisateur. Un fois initialisé à une valeur non vide, il peut servir de point de départ pour accéder à directement à la led. Si deux leds 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, et qui sera publiée automatiquement jusqu'au hub parent. Pour une led, la valeur publiée est l'état de la led (OFF, ON) ou le mode de signalisation si elle clignote (RELAX, AWARE, RUN, CALL ou PANIC).

power

Etat actuel de la led de test. Les valeurs possibles sont: OFF (led éteinte) et ON (led allumée).

luminosity

Intensité lumineuse maximale de la led (pourcentage). Cet attribut permet de contrôler précisément à quelle force la led de test s'allume, lorsqu'un simple état OFF/ON n'est pas suffisant. La led passe à l'état OFF lorsque la luminosité est mise à 0, et à l'état ON si la luminosité est réglée à une valeur non-nulle.

blinking

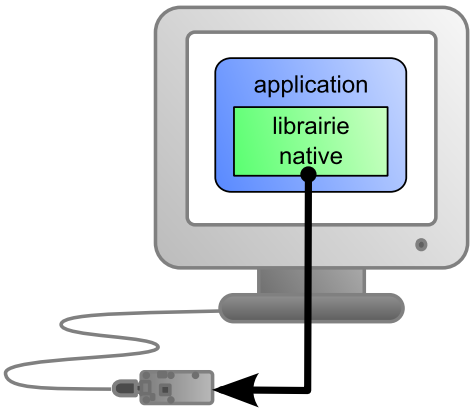
Mode de signalisation de la led: constamment allumée, ou fréquence de clignotement. Les valeurs possibles sont: STILL (pas de clignotement), RELAX (led oscillant à toutes les 4 secondes), AWARE (toutes les 2 secondes), RUN (toutes les secondes), CALL (2 fois par seconde) ou PANIC (4 fois par seconde).

5.5. Quelle interface: Native, DLL ou Service?

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

Contrôle natif

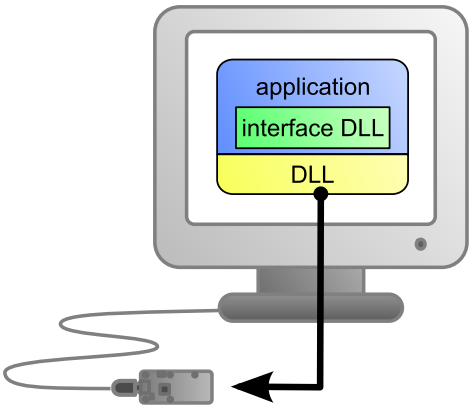
Dans ce cas de figure le programme pilotant votre projet est directement compilé avec une librairie qui offre le contrôle des modules. C'est objectivement la solution la plus simple et la plus élégante pour l'utilisateur final. Il lui suffira de brancher le câble USB et de lancer votre programme pour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



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

Contrôle natif par DLL

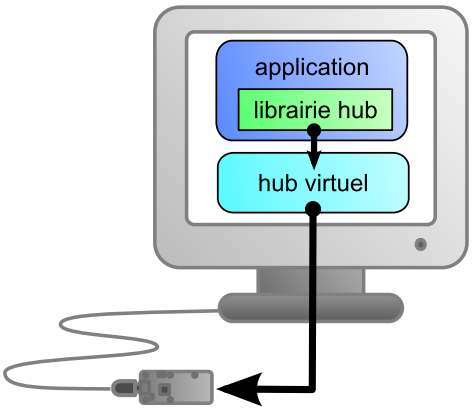
Ici l'essentiel du code permettant de contrôler les modules se trouve dans une DLL, et le programme est compilé avec une petite librairie permettant de contrôler cette DLL. C'est la manière la plus rapide pour coder le support des modules dans un language particulier. En effet la partie "utile" du code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



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

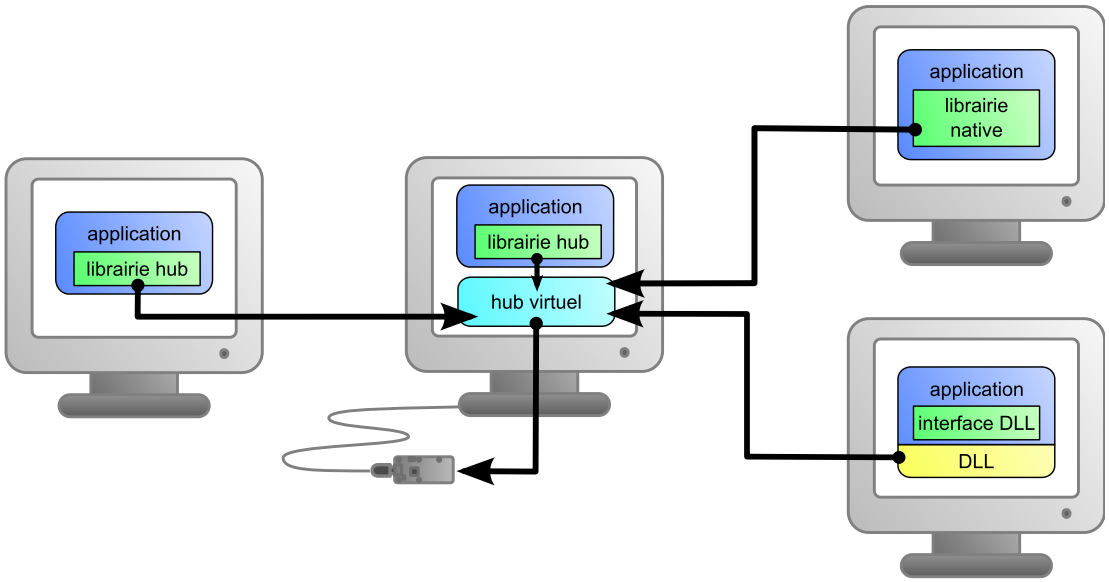
Contrôle par un service

Certain langages ne permettent tout simplement pas d'accéder facilement au niveau matériel de la machine. C'est le cas de Javascript par exemple. Pour gérer ce cas Yoctopuce offre la solution sous la forme d'un petit programme, appelé Hub Virtuel 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 hub virtuel. L'utilisateur final se verra obligé de lancer le hub virtuel avant de lancer le programme de contrôle du projet proprement dit, à moins qu'il ne décide d'installer le hub sous la forme d'un service/démon, auquel cas le hub virtuel se lancera automatiquement au démarrage de la machine..



L'application se connecte au virtual hub pour connecter le module.

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



Lorsqu'on utilise un hub virtuel, 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	-	•	•
Javascript	-	-	•
Node.js	-	-	•
PHP	-	-	•
Java	-	-	•
Java pour Android	•	-	•
Ligne de commande	•	-	•

Méthode de support pour les différents langages.

Limitation des librairies Yoctopuce

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

5.6. Programmation, par où commencer?

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

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

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

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

6. Utilisation du Yocto-Demo en ligne de commande

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

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

6.1. Installation

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

```
C:\>YLed any set_power ON

C:\>YLed any set_blinking RELAX
```

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

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

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

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

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

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

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

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

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

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

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

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

6.3. Contrôle de la fonction Led

Pour contrôler la fonction Led de votre Yocto-Demo, vous avez besoin de l'exécutable YLed.

Vous pouvez par exemple lancer:

```
C:\>YLed any set_power ON

C:\>YLed any set_blinking RELAX
```

Cet exemple utilise la cible "*any*" pour signifier que l'on désire travailler sur la première fonction Led 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-Demo avec le numéros de série *YCTOPOC1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *led* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté).

```
C:\>YLed YCTOPOC1-123456.led describe

C:\>YLed YCTOPOC1-123456.MaFonction describe

C:\>YLed MonModule.led describe

C:\>YLed MonModule.MaFonction describe

C:\>YLed MaFonction describe
```

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

```
C:\>YLed all describe
```

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

```
C:\>YLed /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 YCTOPOC1-12346 set_logicalName MonPremierModule

C:\>YModule YCTOPOC1-12346 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 YCTOPOC1-12346 set_logicalName MonPremierModule
C:\>YModule YCTOPOC1-12346 saveToFlash
```

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

```
C:\>YModule -s  YCTOPOC1-12346 set_logicalName MonPremierModule
```

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

6.5. Limitations

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

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

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

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

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

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

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

7. Utilisation du Yocto-Demo en Javascript

Javascript n'est probablement pas le premier langage qui vous serait venu à l'esprit pour contrôler du matériel, mais il présente l'immense avantage d'être très facile à mettre en oeuvre: avec Javascript, il ne vous faudra qu'un éditeur de texte et un browser internet pour réaliser vos premiers essais.

Javascript fait 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 la passerelle de Yoctopuce appelée VirtualHub sur la machine à laquelle sont branchés les modules

7.1. Préparation

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

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

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

7.2. Contrôle de la fonction Led

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

```
<SCRIPT type="text/javascript" src="yocto_api.js">;</SCRIPT>
<SCRIPT type="text/javascript" src="yocto_led.js"></SCRIPT>

// On récupère l'objet représentant le module, à travers le VirtualHub local
yRegisterHub('http://127.0.0.1:4444/');
var led = yFindLed("YCTOPOC1-123456.led");

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

Voyons maintenant en détail ce que font ces quelques lignes.

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

yocto_api.js et yocto_led.js

Ces deux includes Javascript permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api.js` doit toujours être inclus, `yocto_led.js` est nécessaire pour gérer les modules contenant une led, comme le Yocto-Demo.

yRegisterHub

La fonction `yRegisterHub` permet d'indiquer sur quelle machine se trouve 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.

yFindLed

La fonction `yFindLed`, permet de retrouver une led 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-Demo avec le numéros de série *YCTOPOC1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *led* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
var led = yFindLed("YCTOPOC1-123456.led");
var led = yFindLed("YCTOPOC1-123456.MaFonction");
var led = yFindLed("MonModule.led");
var led = yFindLed("MonModule.MaFonction");
var led = yFindLed("MaFonction");
</
```

`yFindLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la led.

isOnline

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

set_power

La fonction `set_power()` de l'objet renvoyé par `yFindLed` permet d'allumer et d'éteindre la led. L'argument est Y_POWER_ON ou Y_POWER_OFF. Vous trouverez dans la référence de l'interface de programmation d'autres méthodes permettant de contrôler précisément la luminosité et de faire clignoter automatiquement la led.

Un exemple réel

Ouvrez votre éditeur de texte préféré[3], recopiez le code ci-dessous, sauvez-le dans le même répertoire que les fichiers de la librairie, et ouvrez-le avec votre browser favori (sauf Opera). Vous trouverez aussi ce code dans le répertoire **Examples/Doc-GettingStarted-Yocto-Demo** 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.

L'exemple est codé pour être utilisé soit depuis un serveur web, soit en ouvrant directement le fichier localement sur la machine. Notez que cette dernière solution n'est pas possible avec certaines versions de Internet Explorer (en particulier IE 9 de Windows 7), qui refuse d'ouvrir des connections réseau lorsqu'il travaille sur un fichier local. Pour utiliser Internet Explorer, vous devez donc mettre les pages sur un serveur web. Aucun problème par contre avec Chrome, Firefox ou Safari.

Si le Yocto-Demo 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-Demo et où vous avez lancé le VirtualHub.

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

```
<HTML>
<HEAD>
 <TITLE>Hello World</TITLE>
 <SCRIPT type="text/javascript" src="yocto_api.js"></SCRIPT>
 <SCRIPT type="text/javascript" src="yocto_led.js"></SCRIPT>
 <SCRIPT language='javascript1.5' type='text/JavaScript'>
 <!--
 // 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/') != YAPI_SUCCESS) {
     alert("Cannot contact VirtualHub on 127.0.0.1");
 }

 var led;

 function refresh()
 {
     var serial = document.getElementById('serial').value;
     if(serial == '') {
         // Detect any conected module suitable for the demo
         led = yFirstLed();
         if(led) {
             serial = led.module().get_serialNumber();
             document.getElementById('serial').value = serial;
         }
     }

     led = yFindLed(serial+".led");
     if(led.isOnline()) {
         document.getElementById('msg').value = '';
     } else {
         document.getElementById('msg').value = 'Module not connected';
     }
     setTimeout('refresh()',500);
 }

 function switchIt(state)
 {
     if (state) led.set_power(Y_POWER_ON);
           else led.set_power(Y_POWER_OFF);
 }
 -->
 </SCRIPT>
</HEAD>
<BODY onload='refresh();'>
 Module to use: <input id='serial'>
 <input id='msg' style='color:red;border:none;' readonly><br>
 <a href='javascript:switchIt(true);'>ON</a><br>
 <a href='javascript:switchIt(false);'>OFF</a>
</BODY>
</HTML>
```

7.3. 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>
 <SCRIPT type="text/javascript" src="yocto_api.js"></SCRIPT>
 <SCRIPT language='javascript1.5'  type='text/JavaScript'>
 <!--
 // 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/') != YAPI_SUCCESS) {
     alert("Cannot contact VirtualHub on 127.0.0.1");
 }
```

```
var module;

function refresh()
{
    var serial = document.getElementById('serial').value;
    if(serial == '') {
        // Detect any conected module suitable for the demo
        module = yFirstModule().nextModule();
        if(module) {
            serial = module.get_serialNumber();
            document.getElementById('serial').value = serial;
        }
    }

    module = yFindModule(serial);
    if(module.isOnline()) {
        document.getElementById('msg').value = '';
        var html = 'serial: '+module.get_serialNumber()+'<br>';
        html += 'logical name: '+module.get_logicalName()+'<br>';
        html += 'luminosity:'+module.get_luminosity()+'%<br>';
        html += 'beacon:';
        if (module.get_beacon()==Y_BEACON_ON)
            html+="ON <a href='javascript:beacon(Y_BEACON_OFF)'>switch off</a><br>";
        else
            html+="OFF <a href='javascript:beacon(Y_BEACON_ON)'>switch on</a><br>";

        html += 'upTime: '+parseInt(module.get_upTime()/1000)+' sec<br>';
        html += 'USB current: '+module.get_usbCurrent()+' mA<br>';
        html += 'logs:<br><pre>'+module.get_lastLogs()+'</pre><br>';
        document.getElementById('data').innerHTML = html;
    } else {
        document.getElementById('msg').value = 'Module not connected';
    }
    setTimeout('refresh()',1000);
}

function beacon(state)
{
    module.set_beacon(state);
}
-->
</SCRIPT>
</HEAD>
<BODY onload='refresh();'>
Module to use: <input id='serial'>
<input id='msg' style='color:red;border:none;' readonly><br>
<span id='data'></span>
</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>Change module settings</TITLE>
 <SCRIPT type="text/javascript" src="yocto_api.js"></SCRIPT>
 <SCRIPT    type='text/JavaScript'>
 <!--
 // 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/') != YAPI_SUCCESS) {
    alert("Cannot contact VirtualHub on 127.0.0.1");
}

var module;

function refresh()
{
    var serial = document.getElementById('serial').value;
    if(serial == '') {
        // Detect any conected module suitable for the demo
        module = yFirstModule().nextModule();
        if(module) {
            serial = module.get_serialNumber();
            document.getElementById('serial').value = serial;
        }
    }

    module = yFindModule(serial);
    if(module.isOnline()) {
        document.getElementById('msg').value = '';
        document.getElementById('curName').value = module.get_logicalName();
    } else {
        document.getElementById('msg').value = 'Module not connected';
    }
    setTimeout('refresh()',1000);
}

function save()
{
    var newname = document.getElementById('newName').value;
    if (!yCheckLogicalName(newname)) {
        alert('invalid logical name');
        return;
    }
    module.set_logicalName(newname);
    module.saveToFlash();
}
-->
</SCRIPT>
</HEAD>
<BODY onload='refresh();'>
Module to use: <input id='serial'>
<input id='msg' style='color:red;border:none;' readonly><br>
Current name: <input id='curName' readonly><br>
New logical name: <input id='newName'>
<a href='javascript:save();'>Save</a>
</BODY>
</HTML>
```

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

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 `NULL`. Ci-dessous un petit exemple listant les module connectés

```
<HTML>
<HEAD>
<TITLE>Modules inventory</TITLE>
<SCRIPT type="text/javascript" src="yocto_api.js"></SCRIPT>
<SCRIPT    type='text/JavaScript'>
<!--
// 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/') != YAPI_SUCCESS) {
     alert("Cannot contact VirtualHub on 127.0.0.1");
 }

 function refresh()
 {
     yUpdateDeviceList();

     var htmlcode = '';
     var module = yFirstModule();
     while(module) {
         htmlcode += module.get_serialNumber()
                     +'('+module.get_productName()+")<br>";
         module = module.nextModule();
     }
     document.getElementById('list').innerHTML=htmlcode;
     setTimeout('refresh()',500);
 }
 -->
 </SCRIPT>
</HEAD>
<BODY onload='refresh();'>
 <H1>Device list</H1>
 <tt><span id='list'></span></tt>
</BODY>
</HTML>
```

7.4. 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 `yDisableExceptions()` 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.

8. Utilisation du Yocto-Demo en PHP

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

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

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

8.1. Préparation

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

- La librairie de programmation pour PHP[2]
- Le programme VirtualHub[3] pour Windows, Mac OS X ou Linux selon l'OS que vous utilisez

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

8.2. Contrôle de la fonction Led

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

```
include('yocto_api.php');
include('yocto_led.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);
$led = yFindLed("YCTOPOC1-123456.led");

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.php et yocto_led.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_led.php` est nécessaire pour gérer les modules contenant une led, comme le Yocto-Demo.

yRegisterHub

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

yFindLed

La fonction `yFindLed`, permet de retrouver une led 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-Demo avec le numéros de série *YCTOPOC1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *led* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
$led = yFindLed("YCTOPOC1-123456.led");
$led = yFindLed("YCTOPOC1-123456.MaFonction");
$led = yFindLed("MonModule.led");
$led = yFindLed("MonModule.MaFonction");
$led = yFindLed("MaFonction");
```

`yFindLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la led.

isOnline

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

set_power

La fonction `set_power()` de l'objet renvoyé par `yFindLed` permet d'allumer et d'éteindre la led. L'argument est Y_POWER_ON ou Y_POWER_OFF. Vous trouverez dans la référence de l'interface de programmation d'autres méthodes permettant de contrôler précisément la luminosité et de faire clignoter automatiquement la led.

Un exemple réel

Ouvrez votre éditeur de texte préféré[4], recopiez le code ci dessous, sauvez-le dans un répertoire accessible par votre serveur web/PHP avec les fichiers de la librairie, et ouvrez-la page avec votre browser favori. Vous trouverez aussi ce code dans le répertoire **Examples/Doc-GettingStarted-Yocto-Demo** 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.

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

```
<HTML>
<HEAD>
 <TITLE>Hello World</TITLE>
</HEAD>
<BODY>
<FORM method='get'>
<?php
  include('yocto_api.php');
  include('yocto_led.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
      $led = yFindLed("$serial.led");
      if (!$led->isOnline()) {
          die("Module not connected (check serial and USB cable)");
      }
  } else {
      // or use any connected module suitable for the demo
      $led = yFirstLed();
      if(is_null($led)) {
          die("No module connected (check USB cable)");
      } else {
          $serial = $led->module()->get_serialnumber();
      }
  }
  Print("Module to use: <input name='serial' value='$serial'><br>");

  // Drive the selected module
  if (isset($_GET['state'])) {
      $state = $_GET['state'];
      if ($state=='OFF') $led->set_power(Y_POWER_OFF);
      if ($state=='ON')  $led->set_power(Y_POWER_ON);
  }
?>
<input type='radio' name='state' value='ON'>Turn led ON
<input type='radio' name='state' value='OFF'>Turn led OFF
<br><input type='submit'>
</FORM>
</BODY>
</HTML>
```

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());
?>
<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>");
?>
<input type='submit'>
</FORM>
</BODY>
</HTML>
```

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

Enumération des modules

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

```
<HTML>
<HEAD>
 <TITLE>inventory</TITLE>
</HEAD>
<BODY>
<H1>Device list</H1>
<TT>
<?php
    include('yocto_api.php');
    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();
    }
?>
</TT>
</BODY>
```

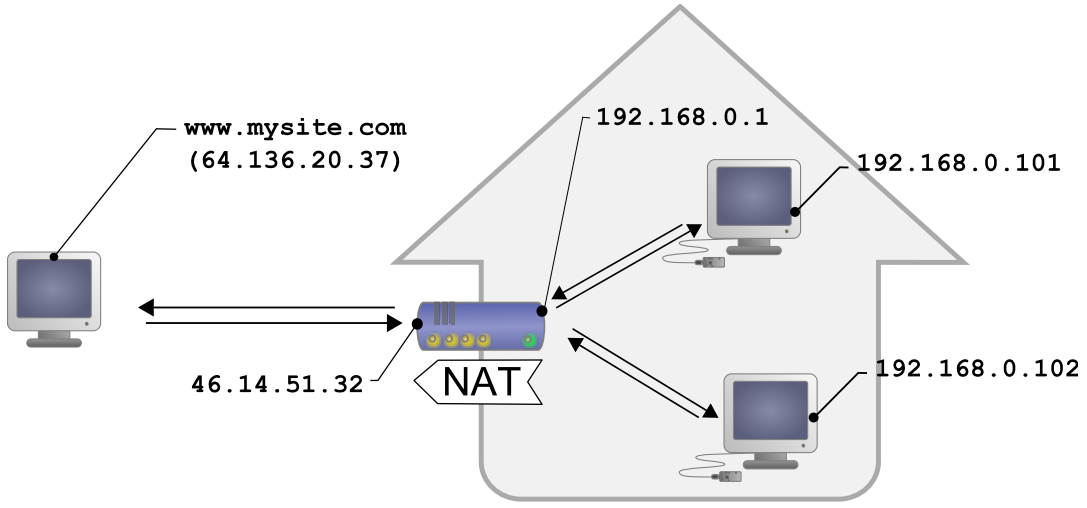
```
</HTML>
```

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

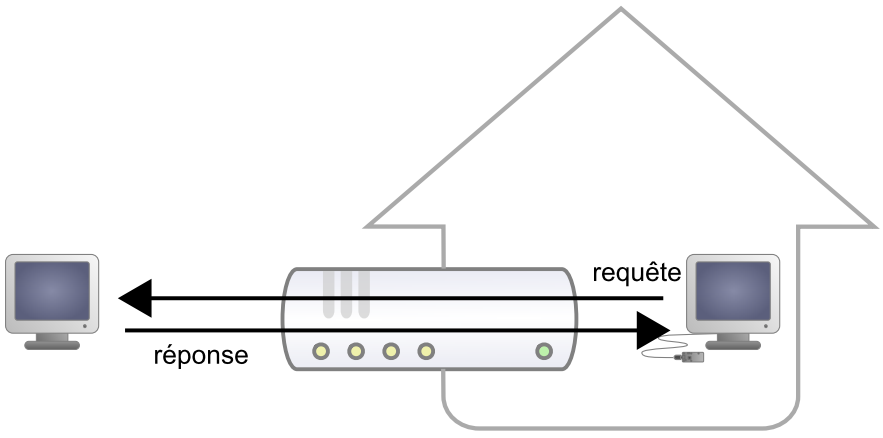
Le filtre NAT, avantages et inconvénients

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

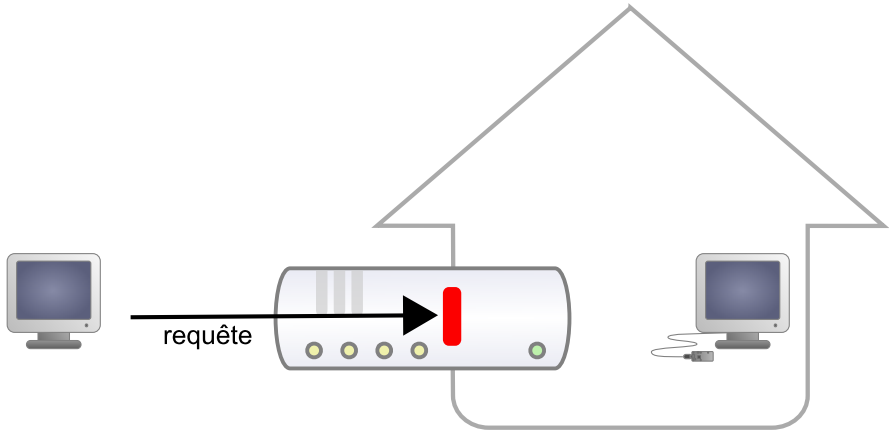


Configuration DSL typique, les machines du LAN sont isolées de l'extérieur par le router DSL

Ce qui, transposé en terme de réseau, donne : les appareils connectés sur un réseau domestique peuvent communiquer entre eux en utilisant une adresse IP locale (du genre 192.168.xxx.yyy), et contacter des serveurs sur Internet par leur adresse publique, mais vu de l'extérieur, il n'y a qu'une seule adresse IP officielle, attribuée au routeur DSL exclusivement. Les différents appareils réseau ne sont pas directement atteignables depuis l'extérieur. C'est assez contraignant, mais c'est une protection relativement efficace contre les intrusions.



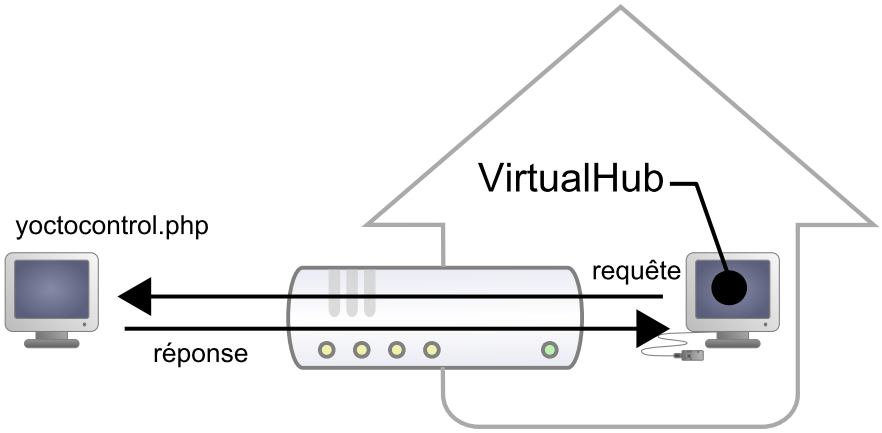
Les réponses aux requêtes venant des machines du LAN sont routées.



Mais les requêtes venant de l'extérieur sont bloquées.

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

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



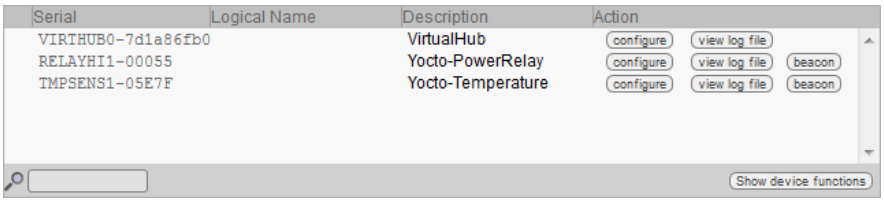
L'API par callback HTTP utilise le VirtualHub, et c'est lui qui initie les requêtes.

Configuration

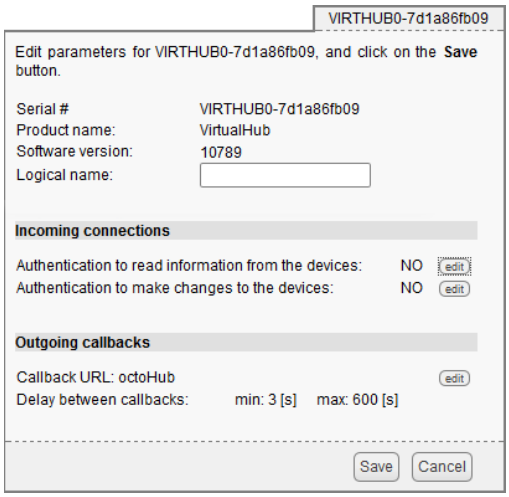
L'API callback se sert donc du *Virtual Hub* comme passerelle. Toutes les communications sont initiées par le *Virtual Hub*, ce sont donc des communication sortantes, et par conséquent parfaitement autorisée par le routeur DSL.

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

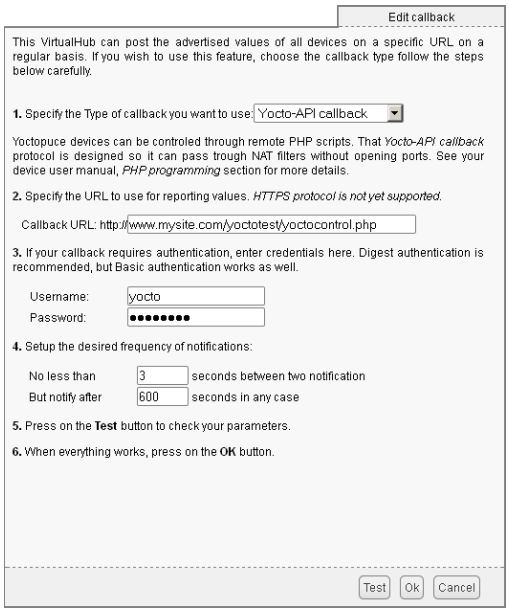
1. Lancer un *VirtualHub*
2. Accéder à son interface, généralement 127.0.0.1:4444
3. Cliquer sur le bouton **configure** de la ligne correspondant au *VirtualHub* lui-même
4. Cliquer sur le bouton **edit** de la section **Outgoing callbacks**



Cliquer sur le bouton "configure" de la première ligne



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



Et choisir "Yocto-API callback".

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

Utilisation

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

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

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

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

Problèmes courants

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

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

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

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

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

Limitations

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

Le mode *Yocto-API par callback HTTP* n'est pour l'instant disponible qu'en PHP 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 `yDisableExceptions()` 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.

9. Utilisation du Yocto-Demo en C++

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

Les librairies Yoctopuce[2] pour C++ vous sont fournies au format source dans leur intégralité. Une partie de la librairie de bas-niveau est écrite en C pur sucre, mais vous n'aurez à priori pas besoin d'interagir directement avec elle: tout a été fait pour que l'interaction soit le plus simple possible depuis le C++. La librairie vous est fournie bien entendu aussi sous forme binaire, de sorte à pouvoir la linker directement si vous le préférez.

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

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

9.1. Contrôle de la fonction Led

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

```
#include "yocto_api.h"
#include "yocto_led.h"

[...]
String  errmsg;
YLed *led;
```

[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);
led = yFindLed("YCTOPOC1-123456.led");

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

Voyons maintenant en détail ce que font ces quelques lignes.

yocto_api.h et yocto_led.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_led.h` est nécessaire pour gérer les modules contenant une led, comme le Yocto-Demo.

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.

yFindLed

La fonction `yFindLed`, permet de retrouver une led 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-Demo avec le numéros de série *YCTOPOC1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *led* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YLed *led = yFindLed("YCTOPOC1-123456.led");
YLed *led = yFindLed("YCTOPOC1-123456.MaFonction");
YLed *led = yFindLed("MonModule.led");
YLed *led = yFindLed("MonModule.MaFonction");
YLed *led = yFindLed("MaFonction");
```

`yFindLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la led.

isOnline

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

set_power

La fonction `set_power()` de l'objet renvoyé par `yFindLed` permet d'allumer et d'éteindre la led. L'argument est Y_POWER_ON ou Y_POWER_OFF. Vous trouverez dans la référence de l'interface de programmation d'autres méthodes permettant de contrôler précisément la luminosité et de faire clignoter automatiquement la led.

Un exemple réel

Lancez votre environnement C++ et ouvrez le projet exemple correspondant, fourni dans le répertoire **Examples/Doc-GettingStarted-Yocto-Demo** 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_led.h"
```

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

using namespace std;

static void usage(void)
{
    cout << "usage: demo <serial_number>  [ on | off ]" << endl;
    cout << "       demo <logical_name> [ on | off ]" << endl;
    cout << "       demo any [ on | off ]                (use any discovered device)" <<
endl;
    u64 now = yGetTickCount();    // dirty active wait loop
        while (yGetTickCount()-now<3000);
    exit(1);
}

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

    if(argc < 3) {
        usage();
    }
    target    = (string) argv[1];
    on_off = (string) argv[2];

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

    if(target == "any"){
        led =  yFirstLed();
    }else{
        led =  yFindLed(target + ".led");
    }
    if (led && led->isOnline()) {
        led->set_power(on_off == "on" ? Y_POWER_ON : Y_POWER_OFF);
    } else {
        cout << "Module not connected (check identification and USB cable)" << endl;
    }

    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;
    }
    return 0;
}
```

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

Modifications des réglages du module

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

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

#include "yocto_api.h"

using namespace std;

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

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

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

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

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

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

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

Enumeration des modules

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

```
#include <iostream>

#include "yocto_api.h"

using namespace std;

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

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

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

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

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

- Elle ne requiert pas l'installation d'une librairie dynamique spécifique à Yoctopuce sur le système final, tout est dans l'exécutable.

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

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

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

Intégration en librairie statique

L'intégration de de la librairie Yoctopuce sous forme de librairie statique 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 -lusb-1.0 -lstdc++
```

Intégration en librairie dynamique

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

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

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

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

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

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

10. Utilisation du Yocto-Demo en Objective-C

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

Les librairies Yoctopuce[1] pour Objective-C vous sont fournies au format source dans leur intégralité. Une partie de la librairie de bas-niveau est écrite en C pur sucre, mais vous n'aurez à priori pas besoin d'interagir directement avec elle: tout a été fait pour que l'interaction soit le plus simple possible depuis Objective-C.

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

Afin des les garder simples, tous les exemples fournis dans cette documentation sont des applications consoles. Il va de soit que que les fonctionnement des librairies est strictement identiques si vous les intégrez dans une application dotée d'une interface graphique. Vous trouverez sur le blog de Yoctopuce un exemple détaillé[2] avec des séquences vidéo montrant comment intégrer les fichiers de la librairie à vos projets.

10.1. Contrôle de la fonction Led

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

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


static void usage(void)
{
    NSLog(@"usage: demo <serial_number>  [ on | off ]");
    NSLog(@"       demo <logical_name> [ on | off ]");
    NSLog(@"       demo any [ on | off ]                (use any discovered device)");
    exit(1);
```

[1] www.yoctopuce.com/FR/libraries.php
[2] www.yoctopuce.com/FR/article/nouvelle-librairie-objective-c-pour-mac-os-x

```
}


int main(int argc, const char * argv[])
{
    NSError *error;
    if(argc < 3) {
        usage();
    }

    @autoreleasepool {
        NSString *target = [NSString stringWithUTF8String:argv[1]];
        NSString *on_off = [NSString stringWithUTF8String:argv[2]];
        YLed     *led;

        if([YAPI RegisterHub:@"usb": &error] != YAPI_SUCCESS) {
            NSLog(@"RegisterHub error: %@", [error localizedDescription]);
            return 1;
        }
        if([target isEqualToString:@"any"]){
            led =  [YLed FirstLed];
        }else{
            led =  [YLed FindLed:[target stringByAppendingString:@".led"]];
        }
        if ([led isOnline]) {
            if ([on_off isEqualToString:@"on"])
                [led set_power:Y_POWER_ON];
            else
                [led set_power:Y_POWER_OFF];
        } else {
            NSLog(@"Module not connected (check identification and USB cable)\n");
        }
    }
    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_led.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_led.h` est nécessaire pour gérer les modules contenant une led, comme le Yocto-Demo.

[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.

[Led FindLed]

La fonction `[Led FindLed]`, permet de retrouver une led 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-Demo avec le numéros de série *YCTOPOC1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *led* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YLed *led = [YLed FindLed:@"YCTOPOC1-123456.led"];
YLed *led = [YLed FindLed:@"YCTOPOC1-123456.MaFonction"];
YLed *led = [YLed FindLed:@"MonModule.led"];
YLed *led = [YLed FindLed:@"MonModule.MaFonction"];
YLed *led = [YLed FindLed:@"MaFonction"];
```

`[YLed FindLed]` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la led.

isOnline

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

set_power

La fonction `set_power()` de l'objet renvoyé par `YLed.FindLed` permet d'allumer et d'éteindre la led. L'argument est YLed.POWER_ON ou YLed.POWER_OFF. Vous trouverez dans la référence de l'interface de programmation d'autres méthodes permettant de contrôler précisément la luminosité et de faire clignoter automatiquement la led.

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);
        }
    }
    return 0;
}
```

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

Modifications des réglages du module

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

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

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


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

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

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

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

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

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

Enumeration des modules

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

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

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

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

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

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

10.3. Gestion des erreurs

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

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

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

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

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `yDisableExceptions()` 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.

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

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 modules[3]. Les fichiers sources en Visual Basic gèrent la partie haut niveau de l'API. Vous avez donc besoin de cette DLL et des fichiers .vb du répertoire `Sources` pour créer un projet gérant des modules Yoctopuce.

Configuration d'un projet Visual Basic

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

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

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

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

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

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

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

11.3. Contrôle de la fonction Led

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

```
[...]
Dim errmsg As String
Dim led As YLed

REM On récupère l'objet représentant le module (ici connecté en local sur USB)
yRegisterHub("usb", errmsg)
led = yFindLed("YCTOPOC1-123456.led")

REM Pour gérer le hot-plug, on vérifie que le module est là
If (led.isOnline()) Then
   REM Utiliser led.set_power(), ...
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.

yFindLed

La fonction `yFindLed`, permet de retrouver une led 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-Demo avec le numéros de série *YCTOPOC1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *led* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
led = yFindLed("YCTOPOC1-123456.led")
led = yFindLed("YCTOPOC1-123456.MaFonction")
led = yFindLed("MonModule.led")
led = yFindLed("MonModule.MaFonction")
led = yFindLed("MaFonction")
```

`yFindLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la led.

[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 `yFindLed` permet de savoir si le module correspondant est présent et en état de marche.

set_power

La fonction `set_power()` de l'objet renvoyé par `yFindLed` permet d'allumer et d'éteindre la led. L'argument est Y_POWER_ON ou Y_POWER_OFF. Vous trouverez dans la référence de l'interface de programmation d'autres méthodes permettant de contrôler précisément la luminosité et de faire clignoter automatiquement la led.

Un exemple réel

Lancez Microsoft VisualBasic et ouvrez le projet exemple correspondant, fourni dans le répertoire **Examples/Doc-GettingStarted-Yocto-Demo** 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>  [ on | off ]")
        Console.WriteLine(execname+" <logical_name> [ on | off ]")
        Console.WriteLine(execname+" any [ on | off ] ")
        System.Threading.Thread.Sleep(2500)

        End
    End Sub

    Sub Main()
        Dim argv() As String = System.Environment.GetCommandLineArgs()
        Dim errmsg As String = ""
        Dim target As String
        Dim led As YLed

        Dim on_off As String


        If argv.Length < 3 Then Usage()

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


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

        If target = "any" Then
            led = yFirstLed()
            If led Is Nothing Then
                Console.WriteLine("No module connected (check USB cable) ")
                End
            End If

        Else
            led = yFindLed(target + ".led")

        End If

        If (led.isOnline()) Then
            If on_off = "ON" Then led.set_power(Y_POWER_ON) Else led.set_power(Y_POWER_OFF)
        Else
            Console.WriteLine("Module not connected (check identification and USB cable)")
        End If
    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 errmsg As String = ""
    Dim m As ymodule

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

    If argv.Length < 2 Then usage()

    m = yFindModule(argv(1)) REM use serial or logical name

    If (m.isOnline()) Then
      If argv.Length > 2 Then
        If argv(2) = "ON" Then m.set_beacon(Y_BEACON_ON)
        If argv(2) = "OFF" Then m.set_beacon(Y_BEACON_OFF)
      End If
      Console.WriteLine("serial:       " + m.get_serialNumber())
      Console.WriteLine("logical name: " + m.get_logicalName())
      Console.WriteLine("luminosity:   " + Str(m.get_luminosity()))
      Console.Write("beacon:       ")
      If (m.get_beacon() = Y_BEACON_ON) Then
        Console.WriteLine("ON")
      Else
        Console.WriteLine("OFF")
      End If
      Console.WriteLine("upTime:       " + Str(m.get_upTime() / 1000) + " sec")
      Console.WriteLine("USB current:  " + Str(m.get_usbCurrent()) + " mA")
      Console.WriteLine("Logs:")
      Console.WriteLine(m.get_lastLogs())
    Else
      Console.WriteLine(argv(1) + " not connected (check identification and USB cable)")
    End If



  End Sub

End Module
```

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

Modifications des réglages du module

Lorsque que vous souhaitez modifier les réglages d'un module, il suffit d'appeler la fonction `set_xxx()` correspondante, cependant cette modification n'a lieu que dans la mémoire vive du module: si le module redémarre, les modifications seront perdues. Pour qu'elle soient mémorisées

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

```
Module Module1


  Sub usage()

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

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

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

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

    m = yFindModule(argv(1)) REM use serial or logical name
    If m.isOnline() Then

      newname = argv(2)
      If (Not yCheckLogicalName(newname)) Then
        Console.WriteLine("Invalid name (" + newname + ")")
        End
      End If
      m.set_logicalName(newname)
      m.saveToFlash() REM do not forget this

      Console.Write("Module: serial= " + m.get_serialNumber)
      Console.Write(" / name= " + m.get_logicalName())
    Else
      Console.Write("not connected (check identification and USB cable")
    End If

  End Sub

End Module
```

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

Enumeration des modules

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

```
Module Module1

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

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

```
      Console.WriteLine("RegisterHub error: " + errmsg)
      End
    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

  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 `yDisableExceptions()` 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.

12. Utilisation du Yocto-Demo en C#

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

12.1. Installation

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

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 modules[3]. Les fichiers sources en Visual C# gèrent la partie haut niveau de l'API. Vous avez donc besoin de cette DLL et des fichiers .cs du répertoire `Sources` pour créer un projet gérant des modules Yoctopuce.

Configuration d'un projet Visual C#

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

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

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 Led

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

```
[...]
string errmsg = "";
YLed led;

// On récupère l'objet représentant le module (ici connecté en local sur USB)
YAPI.RegisterHub("usb", errmsg);
led = YLed.FindLed("YCTOPOC1-123456.led");

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

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.

YLed.FindLed

La fonction `YLed.FindLed`, permet de retrouver une led 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-Demo avec le numéros de série *YCTOPOC1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *led* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
led = YLed.FindLed("YCTOPOC1-123456.led");
led = YLed.FindLed("YCTOPOC1-123456.MaFonction");
led = YLed.FindLed("MonModule.led");
led = YLed.FindLed("MonModule.MaFonction");
led = YLed.FindLed("MaFonction");
```

`YLed.FindLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la led.

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

isOnline

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

set_power

La fonction `set_power()` de l'objet renvoyé par `YLed.FindLed` permet d'allumer et d'éteindre la led. L'argument est YLed.POWER_ON ou YLed.POWER_OFF. Vous trouverez dans la référence de l'interface de programmation d'autres méthodes permettant de contrôler précisément la luminosité et de faire clignoter automatiquement la led.

Un exemple réel

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

    static void Main(string[] args)
    {
      string errmsg = "";
      string target;
      YLed led;
      string on_off;

      if (args.Length < 2) usage();
      target = args[0].ToUpper();
      on_off = args[1].ToUpper();

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

      if (target == "ANY")
      {
        led = YLed.FirstLed();
        if (led == null)
        {
          Console.WriteLine("No module connected (check USB cable) ");
          Environment.Exit(0);
        }
      }
      else led = YLed.FindLed(target + ".led");

      if (led.isOnline())
      {
        if (on_off == "ON") led.set_power(YLed.POWER_ON); else led.set_power(YLed.POWER_OFF
);
      }
      else Console.WriteLine("Module not connected (check identification and USB cable)");
    }
```

```
  }
}
```

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:\r\n"+ m.get_lastLogs());

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

   }
  }
}
```

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");
    }
  }
}
```

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

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

Enumeration des modules

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

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

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

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

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

    }
  }
}
```

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

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

13. Utilisation du Yocto-Demo en Delphi

Delphi est l'héritier de Turbo-Pascal. A l'origine, Delphi était produit par Borland, mais c'est maintenant Embarcadero qui l'édite. Sa force réside dans sa facilité d'utilisation, il permet à quiconque ayant des notions de Pascal de programmer une application Windows en deux temps trois mouvements. Son seul défaut est d'être payant[1].

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

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

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

13.1. Préparation

Connectez-vous sur le site de Yoctopuce et téléchargez la la librairie Yoctopuce pour Delphi[3]. Décompressez le tout dans le répertoire de votre choix, et ajoutez le sous-répertoire *sources* de l'archive dans la liste des répertoires des librairies de Delphi[4].

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

13.2. Contrôle de la fonction Led

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'environement**

```
  SysUtils,
  yocto_api,
  yocto_led;

Procedure  Usage();
  var
   exe : string;

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

procedure setLedState(led:TYLed; state:boolean);
  begin
    if (led.isOnline()) then
     begin
      if state then led.set_power(Y_POWER_ON)
               else led.set_power(Y_POWER_OFF);
     end
    else Writeln('Module not connected (check identification and USB cable)');
  end;

var
  c         : char;
  led       : TYLed;
  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;

  if paramstr(1)='any' then
    begin
      // use the first available led
      led := yFirstLed();
      if led=nil then
         begin
           writeln('No module connected (check USB cable)');
           halt;
         end
       end
   else // or use the one specified on the command line
    led:= YFindLed(paramstr(1)+'.led');

  // make sure it is connected
  if not(led.isOnline()) then
     begin
        Writeln('Module not connected (check identification and USB cable)');
       halt;
     end;

  // minimalist UI
  Writeln('0: turn test led OFF');
  Writeln('1: turn test led ON');
  Writeln('x: exit');
  repeat
    read(c);
    case c  of
      '0' : setLedState(led,false);
      '1' : setLedState(led,true);
    end;
  until c='x';

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_led

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

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.

yFindLed

La fonction `yFindLed`, permet de retrouver une led 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-Demo avec le numéros de série *YCTOPOC1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *led* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
led := yFindLed("YCTOPOC1-123456.led");
led := yFindLed("YCTOPOC1-123456.MaFonction");
led := yFindLed("MonModule.led");
led := yFindLed("MonModule.MaFonction");
led := yFindLed("MaFonction");
```

`yFindLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la led.

isOnline

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

set_power

La fonction `set_power()` de l'objet renvoyé par `yFindLed` permet d'allumer et d'éteindre la led. L'argument est Y_POWER_ON ou Y_POWER_OFF. Vous trouverez dans la référence de l'interface de programmation d'autres méthodes permettant de contrôler précisément la luminosité et de faire clignoter automatiquement la led.

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 = 'YCTOPOC1-123456'; // 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';
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 = 'YCTOPOC1-123456'; // 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();

  Writeln('logical name is now : '+module.get_logicalName());
end.
```

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

Énumération des modules

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

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

var
  module : TYModule;
  errmsg : string;

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

  Writeln('Device list');

  module := yFirstModule();
  while module<>nil  do
   begin
     Writeln( module.get_serialNumber()+' ('+module.get_productName()+')');
     module := module.nextModule();
   end;

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 `yDisableExceptions()` 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-Demo en Python

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

14.1. Fichiers sources

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

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 Led

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

```
[...]

errmsg=YRefParam()
#On récupère l'objet représentant le module (ici connecté en local sur USB)
YAPI.RegisterHub("usb",errmsg)
led = YLed.FindLed("YCTOPOC1-123456.led")

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

[...]
```

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.

YLed.FindLed

La fonction `YLed.FindLed`, permet de retrouver une led 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-Demo avec le numéros de série *YCTOPOC1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *led* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
led = YLed.FindLed("YCTOPOC1-123456.led")
led = YLed.FindLed("YCTOPOC1-123456.MaFonction")
led = YLed.FindLed("MonModule.led")
led = YLed.FindLed("MonModule.MaFonction")
led = YLed.FindLed("MaFonction")
```

`YLed.FindLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la led.

isOnline

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

set_power

La fonction `set_power()` de l'objet renvoyé par `YLed.FindLed` permet d'allumer et d'éteindre la led. L'argument est YLed.POWER_ON ou YLed.POWER_OFF. Vous trouverez dans la référence de l'interface de programmation d'autres méthodes permettant de contrôler précisément la luminosité et de faire clignoter automatiquement la led.

Un exemple réel

Lancez votre interpréteur Python et ouvrez le script correspondant, fourni dans le répertoire **Examples/Doc-GettingStarted-Yocto-Demo** 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_led 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 setLedState(led,state):
    if led.isOnline():
        if state :
            led.set_power(YLed.POWER_ON)
        else:
            led.set_power(YLed.POWER_OFF)
    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':
    # retreive any RGB led
    led = YLed.FirstLed()
    if led is None :
        die('No module connected')
else:
    led= YLed.FindLed(target + '.led')

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

print('0: turn test led OFF')
print('1: turn test led ON')
print('x: exit')

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

c= input("command:")

while c!='x':
    if c=='0' :  setLedState(led,False);
    elif c=='1' :setLedState(led,True);
    c= input("command:")
```

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)")
```

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")
```

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

Enumeration des modules

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

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

from yocto_api import *

errmsg=YRefParam()

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

print('Device list')

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

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 `yDisableExceptions()` 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.

15. Utilisation du Yocto-Demo 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 Java[1]
- Le programme VirtualHub[2] pour Windows, Mac OS X ou Linux selon l'OS que vous utilisez

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

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

15.2. Contrôle de la fonction Led

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

```
[...]

// On récupère l'objet représentant le module (ici connecté en local sur USB)
YAPI.RegisterHub("127.0.0.1");
led = YLed.FindLed("YCTOPOC1-123456.led");

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

[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.

YLed.FindLed

La fonction `YLed.FindLed`, permet de retrouver une led 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-Demo avec le numéros de série *YCTOPOC1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *led* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
led = YLed.FindLed("YCTOPOC1-123456.led")
led = YLed.FindLed("YCTOPOC1-123456.MaFonction")
led = YLed.FindLed("MonModule.led")
led = YLed.FindLed("MonModule.MaFonction")
led = YLed.FindLed("MaFonction")
```

`YLed.FindLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la led.

isOnline

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

set_power

La fonction `set_power()` de l'objet renvoyé par `YLed.FindLed` permet d'allumer et d'éteindre la led. L'argument est YLed.POWER_ON ou YLed.POWER_OFF. Vous trouverez dans la référence de l'interface de programmation d'autres méthodes permettant de contrôler précisément la luminosité et de faire clignoter automatiquement la led.

Un exemple réel

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

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
import com.yoctopuce.YoctoAPI.*;

/**
 *
 * @author yocto
 */
public class Demo {
    /**
     * @param args the command line arguments
     */
    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);
        }

        YLed led;
        if (args.length > 0) {
            led = YLed.FindLed(args[0]);
        } else {
            led = YLed.FirstLed();
            if (led == null) {
                System.out.println("No module connected (check USB cable)");
                System.exit(1);
            }
        }

        try {
            System.out.println("Switch led ON");
            led.set_power(YLed.POWER_ON);
            YAPI.Sleep(1000);
            System.out.println("Switch led OFF");
            led.set_power(YLed.POWER_OFF);
        } catch (YAPI_Exception ex) {
            System.out.println("Module "+led.describe()+" not connected (check
identification and USB cable)");
        }

        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 aux chapitre API

Modifications des réglages du module

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

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

public class Demo {

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

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

        YModule m;
        String newname;

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

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

            m.set_logicalName(newname);
```

```
            m.saveToFlash(); // do not forget this

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

        YAPI.FreeAPI();
    }
}
```

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

Enumeration des modules

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

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

public class Demo {

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

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

        YAPI.FreeAPI();
    }
}
```

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 des que vous débrancherez un module.

16. Utilisation du Yocto-Demo 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 Android[1]. La librairie est disponible en fichiers sources, mais elle aussi disponible sous la forme d'un fichier jar. Branchez vos modules, décompressez les fichiers de la librairie dans le répertoire de votre choix. Et configurez votre environnement de programmation Android pour qu'il puisse les trouver.

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

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 YoctoHub[2].

16.4. Activer le port USB sous Android

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

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

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

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

```
</manifest>
```

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

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

Lancement automatique

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

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

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

</manifest>
```

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

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

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

16.5. Contrôle de la fonction Led

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

```
[...]

// On récupère l'objet représentant le module (ici connecté en local sur USB)
YAPI.EnableUSBHost(this);
YAPI.RegisterHub("usb");
led = YLed.FindLed("YCTOPOC1-123456.led");

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

[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

YAPI.EnableUSBHost

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

YAPI.RegisterHub

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

YLed.FindLed

La fonction `YLed.FindLed` permet de retrouver une led 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-Demo avec le numéros de série *YCTOPOC1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *led* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
led = YLed.FindLed("YCTOPOC1-123456.led")
led = YLed.FindLed("YCTOPOC1-123456.MaFonction")
led = YLed.FindLed("MonModule.led")
led = YLed.FindLed("MonModule.MaFonction")
led = YLed.FindLed("MaFonction")
```

`YLed.FindLed` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler la led.

isOnline

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

set_power

La fonction `set_power()` de l'objet renvoyé par `YLed.FindLed` permet d'allumer et d'éteindre la led. L'argument est YLed.POWER_ON ou YLed.POWER_OFF. Vous trouverez dans la référence de l'interface de programmation d'autres méthodes permettant de contrôler précisément la luminosité et de faire clignoter automatiquement la led.

Un exemple réel

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

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

```
package com.yoctopuce.doc_examples;

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

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

public class GettingStarted_Yocto_Demo extends Activity implements OnItemSelectedListener
{

    private YLed led = null;
    private ArrayAdapter<String> aa;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.gettingstarted_yocto_demo);
        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");
            YLed r = YLed.FirstLed();
            while (r != null) {
                String hwid = r.get_hardwareId();
                aa.add(hwid);
                r = r.nextLed();
            }
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
        // refresh Spinner with detected relay
        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();
        led = YLed.FindLed(hwid);
    }
```

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

    /** Called when the user touches the button State A */
    public void setLedOn(View view)
    {
        // Do something in response to button click
        if (led != null)
            try {
                led.setPower(YLed.POWER_ON);
            } catch (YAPI_Exception e) {
                e.printStackTrace();
            }
    }

    /** Called when the user touches the button State B */
    public void setLedOff(View view)
    {
        // Do something in response to button click
        if (led != null)
            try {
                led.setPower(YLed.POWER_OFF);
            } catch (YAPI_Exception e) {
                e.printStackTrace();
            }
    }

}
```

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.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Switch;
import android.widget.TextView;

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

public class ModuleControl extends Activity implements OnItemSelectedListener
{

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

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

    @Override
```

```
    protected void onStart()
    {
        super.onStart();

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

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

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

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

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

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

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

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

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

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

Modifications des réglages du module

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

```
package com.yoctopuce.doc_examples;

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

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

public class SaveSettings extends Activity implements OnItemSelectedListener
{

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

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

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

        try {
            aa.clear();
            YAPI.EnableUSBHost(this);
            YAPI.RegisterHub("usb");
```

```
            YModule r = YModule.FirstModule();
            while (r != null) {
                String hwid = r.get_hardwareId();
                aa.add(hwid);
                r = r.nextModule();
            }
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
        // refresh Spinner with detected relay
        aa.notifyDataSetChanged();
    }

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

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

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

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

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

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

}
```

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

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

Enumeration des modules

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

```
package com.yoctopuce.doc_examples;

import android.app.Activity;
import android.os.Bundle;
import android.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);
                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 des que vous débrancherez un module.

17. Programmation avancée

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

Afin de rester le plus concis possible, les exemples donnés dans ce chapitre ne font aucune gestion d'erreur. Ne les copiez pas tels-quels dans une application de production.

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 est n'est pas la plus efficace ni la plus réactive. C'est pourquoi l'API de programmation Yoctopuce propose aussi un modèle de programmation par événements. Cette technique consiste à demander à l'API de signaler elle-même les changements importants dès qu'ils sont détectés. A chaque fois qu'un paramètre clé change, l'API appelle une fonction de callback que vous avez prédéfinie.

Détecter l'arrivée et le départ des modules

La gestion du *hot-plug* est importante lorsque l'on travaille avec des modules USB, car tôt ou tard vous serez amené à brancher et débrancher un module après le lancement de votre programme. L'API a été conçue pour gérer l'arrivée et le départ inopinés des modules de manière transparente, mais votre application doit en général en tenir compte si elle veut éviter de prétendre utiliser un module qui a été débranché.

La programmation par événements est particulièrement utile pour détecter les branchements/ débranchements de modules. Il est en effet plus simple de se faire signaler les branchements, que de devoir lister en permanence les modules branchés pour en déduire ceux qui sont arrivés et ceux qui sont partis. Pour pouvoir être prévenu dès qu'un module arrive, vous avez besoin de trois morceaux de code.

Le callback

Le callback est la fonction qui sera appelée à chaque fois qu'un nouveau module Yoctopuce sera branché. Elle prend en paramètre le module concerné.

```
static void deviceArrival(YModule m)
```

```
{
    Console.WriteLine("Nouveau module  : " + m.get_serialNumber());
}
```

L'initialisation
Vous devez ensuite signaler à l'API qu'il faut appeler votre callback quand un nouveau module est branché.

```
YAPI.RegisterDeviceArrivalCallback(deviceArrival);
```

Notez que si des modules sont déjà branchés lorsque le callback est enregistré, le callback sera appelé pour chacun de ces modules déjà branchés.

Déclenchement des callbacks
Un problème classique de la programmation par callbacks est que ces callbacks peuvent être appelés n'importe quand, y compris à des moments où le programme principal n'est pas prêt à les recevoir, ce qui peut avoir des effets de bords indésirables comme des dead-locks et autres conditions de course. C'est pourquoi dans l'API Yoctopuce, les callbacks d'arrivée/départs de modules ne sont appelés que pendant l'exécution de la fonction `UpdateDeviceList()`. Il vous suffit d'appeler `UpdateDeviceList()` à intervalle régulier depuis un timer ou un thread spécifique pour controller précisément quand les appels à ces callbacks auront lieu:

```
// boucle d'attente gérant les callback
while (true)
{
    // callback d'arrivée / départ de modules
    YAPI.UpdateDeviceList(ref errmsg);
    // attente non active gérant les autres callbacks
    YAPI.Sleep(500, ref errmsg);
}
```

De manière similaire, il est possible d'avoir un callback quand un module est débranché. Vous trouverez un exemple concret démontrant toutes ces techniques dans la librairie de programmation Yoctopuce de chaque langage. L'exemple se trouve dans le répertoire *Examples/Prog-EventBased*.

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

18. 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.

18.1. 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.

18.2. 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 uvre, il suffit d'un *Virtual Hub* et d'un simple browser Web. Ainsi, si vous copiez l'URL suivante dans votre browser favori, alors que le *Virtual Hub* est en train de tourner, vous obtiendrez la liste des modules présents.

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

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

Contrôle d'un module par l'interface REST

Chaque module Yoctopuce a sa propre interface REST disponible sous différentes formes. Imaginons un Yocto-Demo avec le numéro de de série *YCTOPOC1-12345* et le nom logique *monModule*. l'URL suivante permettra de connaître l'état du module.

```
http://127.0.0.1:4444/bySerial/YCTOPOC1-12345/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/YCTOPOC1-12345/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/YCTOPOC1-12345/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 led1, il suffit de construire l'URL suivante.

```
http://127.0.0.1:4444/bySerial/YCTOPOC1-12345/api/led1.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/YCTOPOC1-12345/api/led1/logicalName
```

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

```
http://127.0.0.1:4444/bySerial/YCTOPOC1-12345/api/led1?logicalName=maFonction
```

Vous trouverez la liste des attributs disponibles pour votre Yocto-Demo 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/YCTOPOC1-12345/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/YCTOPOC1-12345/dataLogger.json?id=led1&utc=1389801080
```

18.3. Utilisation des librairies dynamiques

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

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

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

Contrôle d'un module

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

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

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

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

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

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

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

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

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

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

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

const
  serial      = 'YCTOPOC1-12345';
  getValue = 'GET /api/module/luminosity HTTP/1.1'#13#10#13#10;
  setValue = 'GET /api/module?luminosity=100 HTTP/1.1'#13#10#13#10;

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

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

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

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

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

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

end.
```

Inventaire des modules

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

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

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

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

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

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

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

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

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

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

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

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


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

begin
  errmsg  :=  @errmsgBuffer;

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

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

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

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

end.
```

VB6 et yapi.dll

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

18.4. 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.

19. Référence de l'API de haut niveau

Ce chapitre résume les fonctions de l'API de haut niveau pour commander votre Yocto-Demo. La syntaxe et les types précis peuvent varier d'un langage à l'autre mais, sauf avis contraire toutes sont disponibles dans chaque language. Pour une information plus précise sur les types des arguments et des valeurs de retour dans un langage donné, veuillez vous référer au fichier de définition pour ce langage (`yocto_api.*` ainsi que les autres fichiers `yocto_*` définissant les interfaces des fonctions).

Dans les langages qui supportent les exceptions, toutes ces fonctions vont par défaut générer des exceptions en cas d'erreur plutôt que de retourner la valeur d'erreur documentée pour chaque fonction, afin de faciliter le déboguage. Il est toutefois possible de désactiver l'utilisation d'exceptions à l'aide de la fonction `yDisableExceptions()`, si l'on préfère travailler avec des valeurs de retour d'erreur.

Ce chapitre ne reprend pas en détail les concepts de programmation décrits plus tôt, afin d'offrir une référence plus concise. En cas de doute, n'hésitez pas à retourner au chapitre décrivant en détail de chaque attribut configurable.

19.1. 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      <script type='text/javascript' src='yocto_api.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YAPI = yoctolib.YAPI;
        var YModule = yoctolib.YModule;
php     require_once('yocto_api.php');
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;
py      from yocto_api import *
```

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 tolerante 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 appellée à chaque fois que l'API a quelque chose à dire.

ySelectArchitecture(**arch**)

Sélectionne manuellement l'architecture de la libraire 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**, **arguments**)

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

ySleep(**ms_duration**, **errmsg**)

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

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() YAPI
yCheckLogicalName()

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

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

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

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

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

YAPI.EnableExceptions() YAPI
yEnableExceptions()

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

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

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

YAPI.EnableUSBHost() yEnableUSBHost()

YAPI

Cette fonction est utilisée uniquement sous Android.

java void **EnableUSBHost**(Object **osContext**)

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

Paramètres :

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

YAPI.FreeAPI() YAPI
yFreeAPI()

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

js	function **yFreeAPI**( )
nodejs	function **FreeAPI**( )
php	function **yFreeAPI**( )
cpp	void **yFreeAPI**( )
m	+(void) **FreeAPI**
pas	procedure **yFreeAPI**( )
vb	procedure **yFreeAPI**( )
cs	void **FreeAPI**( )
java	void **FreeAPI**( )
py	def **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() YAPI
yGetAPIVersion()

Retourne la version de la librairie Yoctopuce utilisée.

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

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

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

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

Retourne :
un long entier contenant la valeur du compteur de millisecondes.

YAPI.HandleEvents() YAPI
yHandleEvents()

Maintient la communication de la librairie avec les modules Yoctopuce.

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

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

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

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

YAPI.InitAPI() YAPI
yInitAPI()

Initialise la librairie de programmation de Yoctopuce explicitement.

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

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

Lorsque cette fonctin est utilisée avec comme `mode` la valeur `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() YAPI
yPreregisterHub()

Alternative plus tolerante à RegisterHub().

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

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

Paramètres :

url une chaîne de caractères contenant **"usb"**,**"callback",** ou l'URL racine du VirtualHub à utiliser.

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

Retourne :

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

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

YAPI.RegisterDeviceArrivalCallback() YAPI yRegisterDeviceArrivalCallback()

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

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

Le callback sera appelé pendant l'éxecution de la fonction `yHandleDeviceList`, que vous devrez appeler régulièrement.

Paramètres :

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

YAPI.RegisterDeviceRemovalCallback() YAPI yRegisterDeviceRemovalCallback()

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

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

Le callback sera appelé pendant l'éxecution de la fonction `yHandleDeviceList`, que vous devrez appeler régulièrement.

Paramètres :

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

YAPI.RegisterHub() YAPI yRegisterHub()

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

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

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

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

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

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

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

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

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

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

Paramètres :

url une chaîne de caractères contenant **"usb"**,**"callback",** ou l'URL racine du VirtualHub à utiliser.

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

YAPI.RegisterHubDiscoveryCallback() YAPI yRegisterHubDiscoveryCallback()

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**)
py	def **RegisterHubDiscoveryCallback**(**hubDiscoveryCallback**)

la fonction de callback reçois deux chaînes de caractères en paramètre La première chaîne contient le numéro de série du hub réseau et la deuxième chaîne contient l'URL du hub. L'URL peut être passée directement en argument à la fonction `yRegisterHub`. Le callback sera appelé pendant l'exécution de la fonction `yHandleDeviceList`, que vous devrez appeler régulièrement.

Paramètres :

hubDiscoveryCallback une procédure qui prend deux chaîne de caractères en paramètre, ou `null`

YAPI.RegisterLogFunction() YAPI
yRegisterLogFunction()

Enregistre une fonction de callback qui sera appellée 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)
py     def RegisterLogFunction( logfun)
```

Utile pour débugger le fonctionnement de l'API.

Paramètres :

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

YAPI.SelectArchitecture() YAPI ySelectArchitecture()

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

py def **SelectArchitecture**(**arch**)

Par défaut, la libraire Python détecte automatiquement la version de la libraire dynamique à utiliser pour accéder au port USB. Sous Linux ARM il n'est pas possible de détecter de manière fiable si il s'agit d'une installation Soft float (armel) ou Hard float (armhf). Dans ce cas, il est donc recommendé d'appeler `SelectArchitecture()` avant tout autre appel à la librairie pour forcer l'utilisation d'une architecture spécifiée.

Paramètres :

arch une chaîne de caractère spécifiant l'architecture à utiliser. Les valeurs possibles sont `"armhf"`,`"armel"`, `"i386"`,`"x86_64"`,`"32bit"`, `"64bit"`

Retourne :

rien.

En cas d'erreur, déclenche une exception.

YAPI.SetDelegate() YAPI ySetDelegate()

(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 `yHandleDeviceList`, que vous devrez appeler régulièrement.

Paramètres :

object un objet qui soit se conformer au protocole `YAPIDelegate`, ou `nil`

YAPI.SetTimeout() YAPI
ySetTimeout()

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

```
js      function ySetTimeout( callback, ms_timeout, arguments)
nodejs  function SetTimeout( callback, ms_timeout, arguments)
```

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
arguments	des arguments supplémentaires peuvent être fournis, pour être passés à la fonction de callback si nécessaire (pas supporté sous Microsoft Internet Explorer).

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.Sleep() YAPI
ySleep()

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

```
js      function ySleep( ms_duration, errmsg)
nodejs  function Sleep( ms_duration, errmsg)
php     function ySleep( $ms_duration, &$errmsg)
cpp     YRETCODE ySleep( unsigned ms_duration, string& errmsg)
m       +(YRETCODE) Sleep :(unsigned) ms_duration :(NSError **) errmsg
pas     function ySleep( ms_duration: integer, var errmsg: string): integer
vb      function ySleep( ByVal ms_duration As Integer,
                         ByRef errmsg As String) As Integer
cs      int Sleep( int ms_duration, ref string errmsg)
java    int Sleep( long ms_duration)
py      def Sleep( ms_duration, errmsg=None)
```

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.TriggerHubDiscovery() YAPI
yTriggerHubDiscovery()

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( )
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() YAPI yUnregisterHub()

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

Paramètres :

url une chaîne de caractères contenant **"usb"** ou

YAPI.UpdateDeviceList() YAPI yUpdateDeviceList()

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

```
js      function yUpdateDeviceList( errmsg)
nodejs  function UpdateDeviceList( errmsg)
php     function yUpdateDeviceList( &$errmsg)
cpp     YRETCODE yUpdateDeviceList( string& errmsg)
m       +(YRETCODE) UpdateDeviceList :(NSError**) errmsg
pas     function yUpdateDeviceList( var errmsg: string): integer
vb      function yUpdateDeviceList( ByRef errmsg As String) As YRETCODE
cs      YRETCODE UpdateDeviceList( ref string errmsg)
java    int UpdateDeviceList( )
py      def UpdateDeviceList( errmsg=None)
```

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() YAPI
yUpdateDeviceList_async()

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-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

19.2. 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;
php     require_once('yocto_api.php');
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;
py      from yocto_api import *
```

Fonction globales
yFindModule(**func**) Permet de retrouver un module d'après son numéro de série ou son nom logique.
yFirstModule() Commence l'énumération des modules accessibles par la librairie.
Méthodes des objets YModule
module→checkFirmware(**path**, **onlynew**) Test si le fichié byn est valid pour le module.
module→describe() Retourne un court texte décrivant le module.
module→download(**pathname**) Télécharge le fichier choisi du module et retourne son contenu.
module→functionCount() Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.
module→functionId(**functionIndex**) Retourne l'identifiant matériel de la *n*ième fonction du module.
module→functionName(**functionIndex**) Retourne le nom logique de la *n*ième fonction du module.
module→functionValue(**functionIndex**) Retourne la valeur publiée par la *n*ième fonction du module.
module→get_allSettings() Retourne tous les paramètres 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_hardwareId()
Retourne l'identifiant unique du module.

module→get_icon2d()
Retourne l'icône du module.

module→get_lastLogs()
Retourne une chaine de charactère contenant les derniers logs du module.

module→get_logicalName()
Retourne le nom logique du module.

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

module→get_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_upTime()
Retourne le numbre de millisecondes écoulées depuis la mise sous tension du 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→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→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**)

Restore tous les paramètres du 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**)

Retourne la valeur entière précédemment stockée dans cet attribut.

module→triggerFirmwareUpdate(**secBeforeReboot**)

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

module→updateFirmware(**path**)

Prepare une mise à jour de firmware du module.

module→wait_async(**callback**, **context**)

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

YModule.FindModule() YModule
yFindModule()

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

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

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

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.FirstModule() YModule
yFirstModule()

Commence l'énumération des modules accessibles par la librairie.

```
js      function yFirstModule( )
nodejs  function FirstModule( )
php     function yFirstModule( )
cpp     YModule* yFirstModule( )
m       +(YModule*) FirstModule
pas     function yFirstModule( ): TYModule
vb      function yFirstModule( ) As YModule
cs      YModule FirstModule( )
java    YModule FirstModule( )
py      def 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

Test si le fichié byn est valid 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)
py      def checkFirmware( path, onlynew)
cmd     YModule target checkFirmware path onlynew
```

Cette methode est utile pour tester 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 firmware équivalent ou plus ancien au firmware installé sont ignorés.

Paramètres :

path le path sur un fichier byn ou un répertoire contenant plusieurs fichier byn

onlynew retourne uniquement les fichier strictement plus récent

Retourne :

: le path du fichier byn a utiliser ou une chaine vide si aucun firmware plus récent est disponible En cas d'erreur, déclenche une exception ou retourne une chaine de caractère qui comment par "error:".

module→describe() YModule

Retourne un court texte décrivant le module.

```
js       function describe( )
nodejs   function describe( )
php      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( )
```

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)
php     function download( $pathname)
cpp     string download( string pathname)
m       -(NSMutableData*) download : (NSString*) pathname
pas     function download( pathname: string): TByteArray
vb      function download( ) As Byte
py      def 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→functionCount() YModule

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

```
js      function functionCount( )
nodejs  function functionCount( )
php     function functionCount( )
cpp     int functionCount( )
m       -(int) functionCount
pas     function functionCount( ): integer
vb      function functionCount( ) As Integer
cs      int functionCount( )
py      def functionCount( )
```

Retourne :
le nombre de fonctions sur le module

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

module→functionId() YModule

Retourne l'identifiant matériel de la *n*ième fonction du module.

```
js      function functionId( functionIndex)
nodejs  function functionId( functionIndex)
php     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)
py      def functionId( functionIndex)
```

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant à l'identifiant matériel unique de la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

module→functionName() YModule

Retourne le nom logique de la *n*ième fonction du module.

```
js      function functionName( functionIndex)
nodejs  function functionName( functionIndex)
php     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)
py      def 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→functionValue() YModule

Retourne la valeur publiée par la *n*ième fonction du module.

```
js      function functionValue( functionIndex)
nodejs  function functionValue( functionIndex)
php     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)
py      def functionValue( functionIndex)
```

Paramètres :

functionIndex l'index de la fonction pour laquelle l'information est désirée, en commençant à 0 pour la première fonction.

Retourne :

une chaîne de caractères correspondant à la valeur publiée par la fonction désirée

En cas d'erreur, déclenche une exception ou retourne un chaîne vide.

module→get_allSettings() YModule
module→allSettings()

Retourne tous les paramètres du module.

```
js      function get_allSettings( )
nodejs  function get_allSettings( )
php     function get_allSettings( )
cpp     string get_allSettings( )
m       -(NSMutableData*) allSettings
pas     function get_allSettings( ): TByteArray
vb      function get_allSettings( ) As Byte
py      def get_allSettings( )
cmd     YModule target get_allSettings
```

Utile pour sauvgarder les noms logiques et les calibrations du module.

Retourne :

un buffer binaire avec touts les paramètres En cas d'erreur, déclenche une exception ou retourne `YAPI_INVALID_STRING`.

module→get_beacon() YModule
module→beacon()

Retourne l'état de la balise de localisation.

```
js      function get_beacon( )
nodejs  function get_beacon( )
php     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( )
py      def get_beacon( )
cmd     YModule target get_beacon
```

Retourne :

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

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

module→get_errorMessage() YModule
module→errorMessage()

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

```
js      function get_errorMessage( )
nodejs  function get_errorMessage( )
php     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( )
```

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( )
php     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( )
```

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

Retourne :
un nombre correspondant au code de la dernière erreur qui s'est produit lors de l'utilisation du module

module→get_firmwareRelease() YModule
module→firmwareRelease()

Retourne la version du logiciel embarqué du module.

```
js      function get_firmwareRelease( )
nodejs  function get_firmwareRelease( )
php     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( )
py      def 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_hardwareId() YModule
module→hardwareId()

Retourne l'identifiant unique du module.

```
js      function get_hardwareId( )
nodejs  function get_hardwareId( )
php     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( )
```

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( )
php     function get_icon2d( )
cpp     string get_icon2d( )
m       -(NSMutableData*) icon2d
pas     function get_icon2d( ): TByteArray
vb      function get_icon2d( ) As Byte
py      def get_icon2d( )
cmd     YModule target get_icon2d
```

L'icone est au format PNG et a une taille maximale de 1536 octets.

Retourne :
un buffer binaire contenant l'icone, au format png. En cas d'erreur, déclenche une exception ou retourne `YAPI_INVALID_STRING`.

module→get_lastLogs() YModule
module→lastLogs()

Retourne une chaine de charactère contenant les derniers logs du module.

```
js      function get_lastLogs( )
nodejs  function get_lastLogs( )
php     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( )
py      def get_lastLogs( )
cmd     YModule target get_lastLogs
```

Cette methode retourne les derniers logs qui sont encore stocké dans le module.

Retourne :

une chaine de charactère 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( )
php     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( )
py      def get_logicalName( )
cmd     YModule target get_logicalName
```

Retourne :
une chaîne de caractères représentant le nom logique du module

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

module→get_luminosity() YModule
module→luminosity()

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

```
js      function get_luminosity( )
nodejs  function get_luminosity( )
php     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( )
py      def 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_persistentSettings() YModule
module→persistentSettings()

Retourne l'état courant des réglages persistents du module.

```
js      function get_persistentSettings( )
nodejs  function get_persistentSettings( )
php     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( )
py      def 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( )
php     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( )
py      def 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( )
php     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( )
py      def 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( )
php     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( )
py      def 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( )
php     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( )
py      def 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( )
php     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( )
py      def 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_upTime() YModule
module→upTime()

Retourne le numbre de millisecondes écoulées depuis la mise sous tension du module

```
js      function get_upTime( )
nodejs  function get_upTime( )
php     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( )
py      def get_upTime( )
cmd     YModule target get_upTime
```

Retourne :

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

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

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( )
php     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( )
py      def 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( )
php     function get_userData( )
cpp     void * get_userData( )
m       -(void*) userData
pas     function get_userData( ): Tobject
vb      function get_userData( ) As Object
cs      object get_userData( )
java    Object get_userData( )
py      def get_userData( )
```

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

Retourne :
l'objet stocké précédemment par l'appelant.

module→get_userVar() YModule
module→userVar()

Retourne la valeur entière précédemment stockée dans cet attribut.

```
js      function get_userVar( )
nodejs  function get_userVar( )
php     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( )
py      def 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→isOnline() YModule

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

```
js      function isOnline( )
nodejs  function isOnline( )
php     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( )
```

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)
php     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( int msValidity)
java    int load( long msValidity)
py      def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

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 occasionellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la 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→nextModule() YModule

Continue l'énumération des modules commencée à l'aide de `yFirstModule()`.

```
js      function nextModule( )
nodejs  function nextModule( )
php     function nextModule( )
cpp     YModule * nextModule( )
m       -(YModule*) nextModule
pas     function nextModule( ): TYModule
vb      function nextModule( ) As YModule
cs      YModule nextModule( )
java    YModule nextModule( )
py      def 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)
php     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)
py      def 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    void registerLogCallback( YModuleLogCallback callback)
m      -(void) registerLogCallback : (YModuleLogCallback) callback
vb     function registerLogCallback( ByVal callback As YModuleLogCallback) As Integer
cs     int registerLogCallback( LogCallback callback)
java   void registerLogCallback( LogCallback callback)
py     def registerLogCallback( callback)
```

Utile pour débugger le fonctionnement d'un module Yoctopuce.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l?objet module qui a produit un log, un chaîne de caractère qui contiens le log

module→revertFromFlash() YModule

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

```
js      function revertFromFlash( )
nodejs  function revertFromFlash( )
php     function revertFromFlash( )
cpp     int revertFromFlash( )
m       -(int) revertFromFlash
pas     function revertFromFlash( ): LongInt
vb      function revertFromFlash( ) As Integer
cs      int revertFromFlash( )
java    int revertFromFlash( )
py      def 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( )
php     function saveToFlash( )
cpp     int saveToFlash( )
m       -(int) saveToFlash
pas     function saveToFlash( ): LongInt
vb      function saveToFlash( ) As Integer
cs      int saveToFlash( )
java    int saveToFlash( )
py      def saveToFlash( )
cmd     YModule target saveToFlash
```

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→set_allSettings() YModule
module→setAllSettings()

Restore tous les paramètres du module.

```
js      function set_allSettings( settings)
nodejs  function set_allSettings( settings)
php     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( )
py      def set_allSettings( settings)
cmd     YModule target set_allSettings settings
```

Utile pour restorer les noms logiques et les calibrations du module depuis un sauvgarde.

Paramètres :

settings un buffer binaire avec touts les paramètres

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

module→set_beacon() YModule
module→setBeacon()

Allume ou éteint la balise de localisation du module.

```
js      function set_beacon( newval)
nodejs  function set_beacon( newval)
php     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)
py      def set_beacon( newval)
cmd     YModule target set_beacon newval
```

Paramètres :

newval soit `Y_BEACON_OFF`, soit `Y_BEACON_ON`

Retourne :

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

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

module→set_logicalName() YModule
module→setLogicalName()

Change le nom logique du module.

```
js      function set_logicalName( newval)
nodejs  function set_logicalName( newval)
php     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)
py      def 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)
php     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)
py      def set_luminosity( newval)
cmd     YModule target set_luminosity newval
```

Le paramêtre 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)
php     function set_userData( $data)
cpp     void set_userData( void* data)
m       -(void) setUserData : (void*) 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)
```

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

Paramètres :

data objet quelconque à mémoriser

module→set_userVar() YModule
module→setUserVar()

Retourne la valeur entière précédemment stockée dans cet attribut.

```
js      function set_userVar( newval)
nodejs  function set_userVar( newval)
php     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)
py      def set_userVar( newval)
cmd     YModule target set_userVar newval
```

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)
php     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)
py      def 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)
php     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)
py      def updateFirmware( path)
cmd     YModule target updateFirmware path
```

Cette methode un object `YFirmwareUpdate` qui est utilisé pour mettre à jour le firmware du module.

Paramètres :

path le path sur un fichier byn

Retourne :

: Un object `YFirmwareUpdate`

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)
```

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.

19.3. Interface de la fonction Led

La librairie de programmation Yoctopuce permet non seulement d'allumer la led à une intensité donnée, mais aussi de la faire osciller à plusieurs fréquences.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js      <script type='text/javascript' src='yocto_led.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YLed = yoctolib.YLed;
php     require_once('yocto_led.php');
cpp     #include "yocto_led.h"
m       #import "yocto_led.h"
pas     uses yocto_led;
vb      yocto_led.vb
cs      yocto_led.cs
java    import com.yoctopuce.YoctoAPI.YLed;
py      from yocto_led import *
```

Fonction globales
yFindLed(**func**) Permet de retrouver une led d'après un identifiant donné.
yFirstLed() Commence l'énumération des leds accessibles par la librairie.
Méthodes des objets YLed
led→describe() Retourne un court texte décrivant de manière non-ambigüe l'instance de la led au format `TYPE(NAME)=SERIAL.FUNCTIONID`.
led→get_advertisedValue() Retourne la valeur courante de la led (pas plus de 6 caractères).
led→get_blinking() Retourne le mode de signalisation de la led.
led→get_errorMessage() Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la led.
led→get_errorType() Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la led.
led→get_friendlyName() Retourne un identifiant global de la led au format `NOM_MODULE.NOM_FONCTION`.
led→get_functionDescriptor() Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.
led→get_functionId() Retourne l'identifiant matériel de la led, sans référence au module.
led→get_hardwareId() Retourne l'identifiant matériel unique de la led au format `SERIAL.FUNCTIONID`.
led→get_logicalName() Retourne le nom logique de la led.
led→get_luminosity() Retourne l'intensité de la led en pour cent.
led→get_module()

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

led→get_module_async(callback, context)

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

led→get_power()

Retourne l'état courant de la led.

led→get_userData()

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

led→isOnline()

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

led→isOnline_async(callback, context)

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

led→load(msValidity)

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

led→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

led→nextLed()

Continue l'énumération des leds commencée à l'aide de `yFirstLed()`.

led→registerValueCallback(callback)

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

led→set_blinking(newval)

Modifie le mode de signalisation de la led.

led→set_logicalName(newval)

Modifie le nom logique de la led.

led→set_luminosity(newval)

Modifie l'intensité lumineuse de la led (en pour cent).

led→set_power(newval)

Modifie l'état courant de la led.

led→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`.

led→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.

YLed.FindLed() YLed
yFindLed()

Permet de retrouver une led d'après un identifiant donné.

```
js      function yFindLed( func)
nodejs  function FindLed( func)
php     function yFindLed( $func)
cpp     YLed* yFindLed( const string& func)
m       +(YLed*) FindLed :(NSString*) func
pas     function yFindLed( func: string): TYLed
vb      function yFindLed( ByVal func As String) As YLed
cs      YLed FindLed( string func)
java    YLed FindLed( String func)
py      def FindLed( 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 soit en ligne au moment ou elle est appelée, l'objet retourné sera néanmoins valide. Utiliser la méthode `YLed.isOnline()` pour tester si la led 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 sans ambiguïté

Retourne :

un objet de classe `YLed` qui permet ensuite de contrôler la led.

YLed.FirstLed() YLed
yFirstLed()

Commence l'énumération des leds accessibles par la librairie.

```
js      function yFirstLed( )
nodejs  function FirstLed( )
php     function yFirstLed( )
cpp     YLed* yFirstLed( )
m       +(YLed*) FirstLed
pas     function yFirstLed( ): TYLed
vb      function yFirstLed( ) As YLed
cs      YLed FirstLed( )
java    YLed FirstLed( )
py      def FirstLed( )
```

Utiliser la fonction `YLed.nextLed()` pour itérer sur les autres leds.

Retourne :

un pointeur sur un objet `YLed`, correspondant à la première led accessible en ligne, ou `null` si il n'y a pas de leds disponibles.

led→describe() YLed

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

```
js      function describe( )
nodejs  function describe( )
php     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( )
```

Plus précisément, TYPE correspond au type de fonction, NAME correspond au nom utilsé lors du premier accès a la fonction, SERIAL correspond au numéro de série du module si le module est connecté, ou "unresolved" sinon, et FUNCTIONID correspond à l'identifiant matériel de la fonction si le module est connecté. Par exemple, La methode va retourner Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1 si le module est déjà connecté ou Relay(BadCustomeName.relay1)=unresolved si le module n'est pas déjà connecté. Cette methode ne declenche aucune transaction USB ou TCP et peut donc être utilisé dans un debuggeur.

Retourne :

une chaîne de caractères décrivant la led (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

led→get_advertisedValue() YLed
led→advertisedValue()

Retourne la valeur courante de la led (pas plus de 6 caractères).

```
js      function get_advertisedValue( )
nodejs  function get_advertisedValue( )
php     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( )
py      def get_advertisedValue( )
cmd     YLed target get_advertisedValue
```

Retourne :
une chaîne de caractères représentant la valeur courante de la led (pas plus de 6 caractères).

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

led→get_blinking() YLed
led→blinking()

Retourne le mode de signalisation de la led.

```
js      function get_blinking( )
nodejs  function get_blinking( )
php     function get_blinking( )
cpp     Y_BLINKING_enum get_blinking( )
m       -(Y_BLINKING_enum) blinking
pas     function get_blinking( ): Integer
vb      function get_blinking( ) As Integer
cs      int get_blinking( )
java    int get_blinking( )
py      def get_blinking( )
cmd     YLed target get_blinking
```

Retourne :

une valeur parmi `Y_BLINKING_STILL`, `Y_BLINKING_RELAX`, `Y_BLINKING_AWARE`, `Y_BLINKING_RUN`, `Y_BLINKING_CALL` et `Y_BLINKING_PANIC` représentant le mode de signalisation de la led

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

led→get_errorMessage() YLed
led→errorMessage()

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

```
js      function get_errorMessage( )
nodejs  function get_errorMessage( )
php     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( )
```

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.

led→get_errorType() YLed
led→errorType()

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

```
js      function get_errorType( )
nodejs  function get_errorType( )
php     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( )
```

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.

led→get_friendlyName() YLed
led→friendlyName()

Retourne un identifiant global de la led au format NOM_MODULE.NOM_FONCTION.

```
js      function get_friendlyName( )
nodejs  function get_friendlyName( )
php     function get_friendlyName( )
cpp     string get_friendlyName( )
m       -(NSString*) friendlyName
cs      string get_friendlyName( )
java    String get_friendlyName( )
py      def get_friendlyName( )
```

Le chaîne retournée utilise soit les noms logiques du module et de la led si ils sont définis, soit respectivement le numéro de série du module et l'identifant matériel de la led (par exemple: MyCustomName.relay1)

Retourne :

une chaîne de caractères identifiant la led en utilisant les noms logiques (ex: MyCustomName.relay1)

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

led→get_functionDescriptor() YLed
led→functionDescriptor()

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

js	function **get_functionDescriptor**( )
nodejs	function **get_functionDescriptor**( )
php	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**( )

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

led→get_functionId() YLed
led→functionId()

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

```
js      function get_functionId( )
nodejs  function get_functionId( )
php     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( )
```

Par example `relay1`.

Retourne :

une chaîne de caractères identifiant la led (ex: `relay1`)

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

led→get_hardwareId() YLed
led→hardwareId()

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

```
js      function get_hardwareId( )
nodejs  function get_hardwareId( )
php     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( )
```

L'identifiant unique est composé du numéro de série du module et de l'identifiant matériel de la led (par example RELAYLO1-123456.relay1).

Retourne :

une chaîne de caractères identifiant la led (ex: RELAYLO1-123456.relay1)

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

led→get_logicalName() YLed
led→logicalName()

Retourne le nom logique de la led.

```
js      function get_logicalName( )
nodejs  function get_logicalName( )
php     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( )
py      def get_logicalName( )
cmd     YLed target get_logicalName
```

Retourne :
une chaîne de caractères représentant le nom logique de la led.

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

led→get_luminosity() YLed
led→luminosity()

Retourne l'intensité de la led en pour cent.

```
js      function get_luminosity( )
nodejs  function get_luminosity( )
php     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( )
py      def get_luminosity( )
cmd     YLed target get_luminosity
```

Retourne :
un entier représentant l'intensité de la led en pour cent

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

led→get_module() YLed
led→module()

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

```
js      function get_module( )
nodejs  function get_module( )
php     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( )
```

Si la fonction ne peut être trouvée sur aucun module, l'instance de YModule retournée ne sera pas joignable.

Retourne :
une instance de YModule

led→get_module_async() YLed
led→module_async()

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.

led→get_power() YLed
led→power()

Retourne l'état courant de la led.

```
js      function get_power( )
nodejs  function get_power( )
php     function get_power( )
cpp     Y_POWER_enum get_power( )
m       -(Y_POWER_enum) power
pas     function get_power( ): Integer
vb      function get_power( ) As Integer
cs      int get_power( )
java    int get_power( )
py      def get_power( )
cmd     YLed target get_power
```

Retourne :

soit `Y_POWER_OFF`, soit `Y_POWER_ON`, selon l'état courant de la led

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

led→get_userData() YLed
led→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( )
php     function get_userData( )
cpp     void * get_userData( )
m       -(void*) userData
pas     function get_userData( ): Tobject
vb      function get_userData( ) As Object
cs      object get_userData( )
java    Object get_userData( )
py      def get_userData( )
```

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

Retourne :
l'objet stocké précédemment par l'appelant.

led→isOnline() YLed

Vérifie si le module hébergeant la led est joignable, sans déclencher d'erreur.

```
js      function isOnline( )
nodejs  function isOnline( )
php     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( )
```

Si les valeurs des attributs en cache de la led 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 est joignable, `false` sinon

led→isOnline_async() YLed

Vérifie si le module hébergeant la led 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 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.

led→load() YLed

Met en cache les valeurs courantes de la led, avec une durée de validité spécifiée.

```
js      function load( msValidity)
nodejs  function load( msValidity)
php     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( int msValidity)
java    int load( long msValidity)
py      def load( msValidity)
```

Par défaut, lorsqu'on accède à un module, tous les attributs des fonctions du module sont automatiquement mises en cache pour la durée standard (5 ms). Cette méthode peut être utilisée pour marquer occasionellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Paramètres :

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

led→load_async() YLed

Met en cache les valeurs courantes de la led, 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 occasionellement les données cachées comme valides pour une plus longue période, par exemple dans le but de réduire le trafic réseau.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la machine virtuelle Javascript avec une attente active.

Paramètres :

msValidity	un entier correspondant à la durée de validité attribuée aux les paramètres chargés, en millisecondes
callback	fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou YAPI_SUCCESS)
context	contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

rien du tout : le résultat sera passé en paramètre à la fonction de callback.

led→nextLed() YLed

Continue l'énumération des leds commencée à l'aide de `yFirstLed()`.

```
js      function nextLed( )
nodejs  function nextLed( )
php     function nextLed( )
cpp     YLed * nextLed( )
m       -(YLed*) nextLed
pas     function nextLed( ): TYLed
vb      function nextLed( ) As YLed
cs      YLed nextLed( )
java    YLed nextLed( )
py      def nextLed( )
```

Retourne :

un pointeur sur un objet `YLed` accessible en ligne, ou `null` lorsque l'énumération est terminée.

led→registerValueCallback() YLed

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

js	function **registerValueCallback**(**callback**)
nodejs	function **registerValueCallback**(**callback**)
php	function **registerValueCallback**(**$callback**)
cpp	int **registerValueCallback**(YLedValueCallback **callback**)
m	-(int) **registerValueCallback** : (YLedValueCallback) **callback**
pas	function **registerValueCallback**(**callback**: TYLedValueCallback): LongInt
vb	function **registerValueCallback**( ) As Integer
cs	int **registerValueCallback**(ValueCallback **callback**)
java	int **registerValueCallback**(UpdateCallback **callback**)
py	def **registerValueCallback**(**callback**)

Ce callback n'est appelé que durant l'exécution de `ySleep` ou `yHandleEvents`. Cela permet à l'appelant de contrôler quand les callback peuvent se produire. Il est important d'appeler l'une de ces deux fonctions périodiquement pour garantir que les callback ne soient pas appelés trop tard. Pour désactiver un callback, il suffit d'appeler cette méthode en lui passant un pointeur nul.

Paramètres :

callback la fonction de callback à rappeler, ou un pointeur nul. La fonction de callback doit accepter deux arguments: l'object fonction dont la valeur a changé, et la chaîne de caractère décrivant la nouvelle valeur publiée.

led→set_blinking() YLed
led→setBlinking()

Modifie le mode de signalisation de la led.

```
js      function set_blinking( newval)
nodejs  function set_blinking( newval)
php     function set_blinking( $newval)
cpp     int set_blinking( Y_BLINKING_enum newval)
m       -(int) setBlinking : (Y_BLINKING_enum) newval
pas     function set_blinking( newval: Integer): integer
vb      function set_blinking( ByVal newval As Integer) As Integer
cs      int set_blinking( int newval)
java    int set_blinking( int newval)
py      def set_blinking( newval)
cmd     YLed target set_blinking newval
```

Paramètres :

newval une valeur parmi `Y_BLINKING_STILL`, `Y_BLINKING_RELAX`, `Y_BLINKING_AWARE`, `Y_BLINKING_RUN`, `Y_BLINKING_CALL` et `Y_BLINKING_PANIC` représentant le mode de signalisation de la led

Retourne :

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

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

led→set_logicalName() YLed
led→setLogicalName()

Modifie le nom logique de la led.

```
js      function set_logicalName( newval)
nodejs  function set_logicalName( newval)
php     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)
py      def set_logicalName( newval)
cmd     YLed 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.

Retourne :

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

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

led→set_luminosity() YLed
led→setLuminosity()

Modifie l'intensité lumineuse de la led (en pour cent).

```
js      function set_luminosity( newval)
nodejs  function set_luminosity( newval)
php     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)
py      def set_luminosity( newval)
cmd     YLed target set_luminosity newval
```

Paramètres :

newval un entier représentant l'intensité lumineuse de la led (en pour cent)

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur.

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

led→set_power() YLed
led→setPower()

Modifie l'état courant de la led.

```
js      function set_power( newval)
nodejs  function set_power( newval)
php     function set_power( $newval)
cpp     int set_power( Y_POWER_enum newval)
m       -(int) setPower : (Y_POWER_enum) newval
pas     function set_power( newval: Integer): integer
vb      function set_power( ByVal newval As Integer) As Integer
cs      int set_power( int newval)
java    int set_power( int newval)
py      def set_power( newval)
cmd     YLed target set_power newval
```

Paramètres :

newval soit `Y_POWER_OFF`, soit `Y_POWER_ON`, selon l'état courant de la led

Retourne :

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

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

led→set_userData() YLed
led→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)
php     function set_userData( $data)
cpp     void set_userData( void* data)
m       -(void) setUserData : (void*) 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)
```

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

Paramètres :

data objet quelconque à mémoriser

led→wait_async() YLed

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)
```

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. Problèmes courants

20.1. 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 éviterons de devoir partir de rien.

Exemple 1: 51-yoctopuce.rules

Cette règle va autoriser tous les utilisateurs à accéder en lecture et en écriture aux périphériques Yoctopuce USB. Les droits d'accès pour tous les autres périphériques ne seront pas modifiés. Si ce scénario vous convient il suffit de copier le fichier "`51-yoctopuce_all.rules`" dans le répertoire "`/etc/udev/rules.d`" et de redémarrer votre système.

```
# 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"
```

20.2. Plateformes ARM: HF et EL

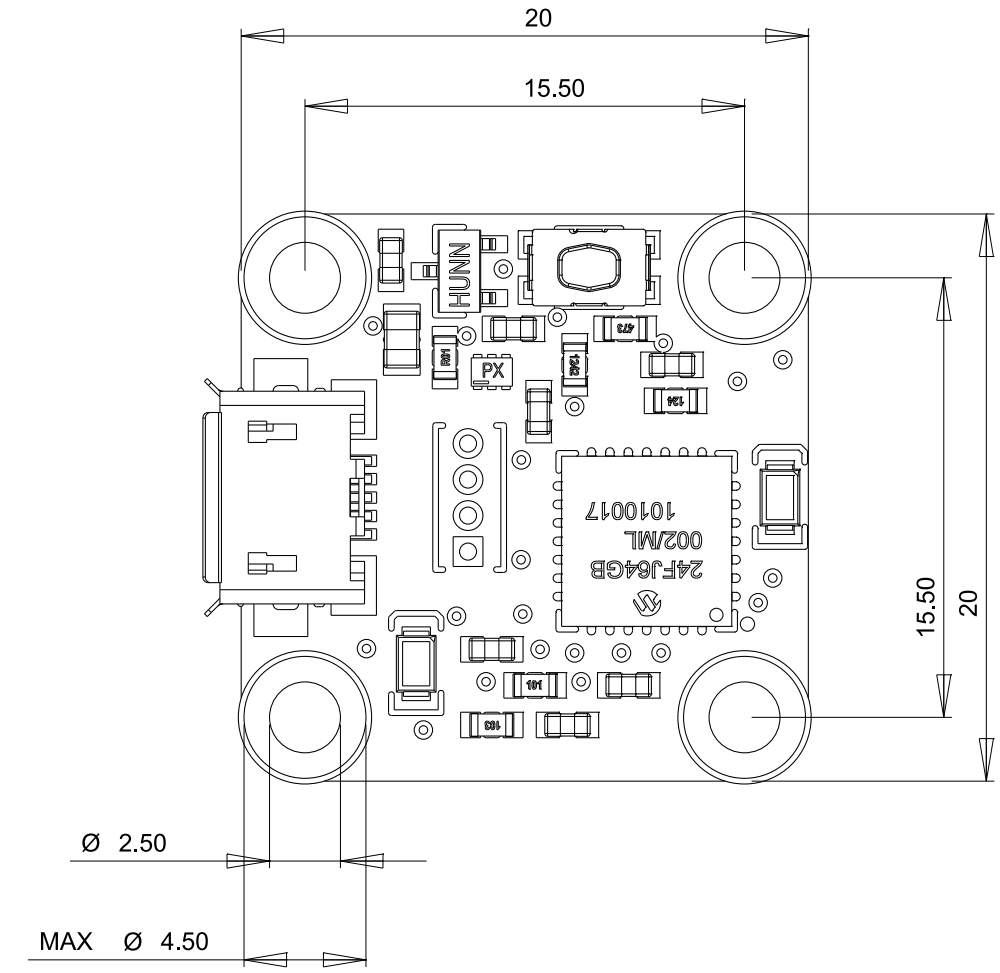
Sur ARM il existe deux grandes familles d'executables: HF (Hard Float) et EL (EABI Little Endian). Ces deux familles ne sont absolument pas compatibles entre elles. La capacité d'une machine ARM à faire tourner des exécutables de l'une ou l'autre de ces familles dépend du hardware et du système d'exploitation. Les problèmes de compatibilité entre ArmHL et ArmEL sont assez difficiles à diagnostiquer, souvent même l'OS se révèle incapable de distinguer un exécutable HF d'un exécutable EL.

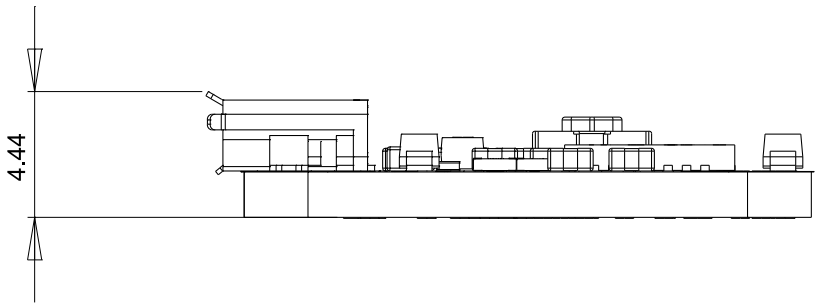
Tous les binaires Yoctopuce pour ARM sont fournis pré-compilée pour ArmHF et ArmEL, si vous ne savez à quelle famille votre machine ARM apartient, essayez simplement de lancer un exécutable de chaque famille.

21. Caractéristiques

Vous trouverez résumées ci dessous les principales caractéristiques techniques de votre module Yocto-Demo

Largeur	20 mm
Longueur	20 mm
Poids	2 g
Connecteur USB	micro-B
Sytème 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	0x0009





All dimensions are in mm
Toutes les dimensions sont en mm

Yocto-Demo

A4

Scale
4:1
Echelle

Index

A
Accès 89
Accessoires 3
Activer 90
Alimentation 12
Android 89, 90
Avancée 101

B
Basic 57
Blueprint 221

C
C# 63
C++ 43, 48
Callback 38
Caractéristiques 219
checkFirmware, YModule 140
CheckLogicalName, YAPI 111
Commande 21, 103
Commencer 19
Compatibilité 89
Concepts 13
Configuration 8
Connectique 11
Contraintes 12
Contrôle 15, 22, 23, 25, 27, 33, 35, 43, 45, 51, 53, 58, 60, 64, 66, 71, 73, 78, 79, 83, 85, 92, 94, 136
Courants 217

D
Delphi 71
describe, YLed 189
describe, YModule 141
Description 21
DisableExceptions, YAPI 112
download, YModule 142
Dynamique 77
Dynamiques 105

E
Éléments 5, 6
EnableExceptions, YAPI 113
EnableUSBHost, YAPI 114
Erreurs 30, 41, 47, 55, 62, 68, 76, 81, 87, 99
Événements 101

F
Fichiers 77
Filtres 38
FindLed, YLed 187
FindModule, YModule 138
FirstLed, YLed 188
FirstModule, YModule 139
Fixation 11
Fonctions 110
FreeAPI, YAPI 115
functionCount, YModule 143
functionId, YModule 144
functionName, YModule 145
functionValue, YModule 146

G
get_advertisedValue, YLed 190
get_allSettings, YModule 147
get_beacon, YModule 148
get_blinking, YLed 191
get_errorMessage, YLed 192
get_errorMessage, YModule 149
get_errorType, YLed 193
get_errorType, YModule 150
get_firmwareRelease, YModule 151
get_friendlyName, YLed 194
get_functionDescriptor, YLed 195
get_functionId, YLed 196
get_hardwareId, YLed 197
get_hardwareId, YModule 152
get_icon2d, YModule 153
get_lastLogs, YModule 154
get_logicalName, YLed 198
get_logicalName, YModule 155
get_luminosity, YLed 199
get_luminosity, YModule 156
get_module, YLed 200
get_module_async, YLed 201
get_persistentSettings, YModule 157
get_power, YLed 202
get_productId, YModule 158
get_productName, YModule 159
get_productRelease, YModule 160
get_rebootCountdown, YModule 161
get_serialNumber, YModule 162
get_upTime, YModule 163
get_usbCurrent, YModule 164
get_userData, YLed 203
get_userData, YModule 165
get_userVar, YModule 166
GetAPIVersion, YAPI 116
GetTickCount, YAPI 117

H
HandleEvents, YAPI 118
HTTP 38, 103
Hub 89

I

InitAPI, YAPI 119
Installation 21, 57, 63
Intégration 48
Interface 136, 186
Introduction 1
isOnline, YLed 204
isOnline, YModule 167
isOnline_async, YLed 205
isOnline_async, YModule 168

J

Java 83
Javascript 25

L

Langages 103
Librairie 48, 77, 108
Librairies 105
Limitations 23
Linux 217
load, YLed 206
load, YModule 169
load_async, YLed 207
load_async, YModule 170
Localisation 7

M

Module 7, 15, 23, 27, 35, 45, 53, 60, 66, 73, 79, 85, 94, 136
Montage 11

N

Natif 89
Native 17
.NET 57
nextLed, YLed 208
nextModule, YModule 171
Niveau 108, 109

O

Objective-C 51

P

Paradigme 13
Plateformes 218
Port 90, 108
Préparation 25, 33, 71, 83, 89
PreregisterHub, YAPI 120
Prérequis 1
Présentation 5
Problèmes 217
Programmation 13, 19, 101
Projet 57, 63
Python 77

R

reboot, YModule 172
Référence 109
RegisterDeviceArrivalCallback, YAPI 121
RegisterDeviceRemovalCallback, YAPI 122
RegisterHub, YAPI 123
RegisterHubDiscoveryCallback, YAPI 125
registerLogCallback, YModule 173
RegisterLogFunction, YAPI 126
registerValueCallback, YLed 209
revertFromFlash, YModule 174

S

saveToFlash, YModule 175
SelectArchitecture, YAPI 127
Service 17
set_allSettings, YModule 176
set_beacon, YModule 177
set_blinking, YLed 210
set_logicalName, YLed 211
set_logicalName, YModule 178
set_luminosity, YLed 212
set_luminosity, YModule 179
set_power, YLed 213
set_userData, YLed 214
set_userData, YModule 180
set_userVar, YModule 181
SetDelegate, YAPI 128
SetTimeout, YAPI 129
Sleep, YAPI 130
Sources 77
Supportés 103

T

Test 7
triggerFirmwareUpdate, YModule 182
TriggerHubDiscovery, YAPI 131

U

UnregisterHub, YAPI 132
UpdateDeviceList, YAPI 133
UpdateDeviceList_async, YAPI 134
updateFirmware, YModule 183

V

Virtual 89, 103
Visual 57, 63
VisualBasic 57

W

wait_async, YLed 215
wait_async, YModule 184

Y

YAPI 125-134
yCheckLogicalName 111
yDisableExceptions 112
yEnableExceptions 113
yEnableUSBHost 114
yFindLed 187
yFindModule 138
yFirstLed 188
yFirstModule 139
yFreeAPI 115
yGetAPIVersion 116
yGetTickCount 117
yHandleEvents 118
yInitAPI 119
YLed 187-215
YModule 138-184
Yocto-Demo 15, 21, 25, 33, 43, 51, 57, 63, 71, 77, 83, 89
yPreregisterHub 120
yRegisterDeviceArrivalCallback 121
yRegisterDeviceRemovalCallback 122
yRegisterHub 123
yRegisterHubDiscoveryCallback 125
yRegisterLogFunction 126
ySelectArchitecture 127
ySetDelegate 128
ySetTimeout 129
ySleep 130
yTriggerHubDiscovery 131
yUnregisterHub 132
yUpdateDeviceList 133
yUpdateDeviceList_async 134