

Yocto-Display

Mode d'emploi

---



# Table des matières

<b>1. Introduction</b>	<b>1</b>
1.1. Informations de sécurité	2
1.2. Conditions environnementales	3
<b>2. Présentation</b>	<b>5</b>
2.1. Les éléments communs	5
2.2. Les éléments spécifiques	6
2.3. Accessoires optionnels	7
<b>3. Principes de fonctionnement</b>	<b>9</b>
3.1. Processeur et mémoire embarqués	9
3.2. Orientation	9
3.3. Système de couche	9
3.4. Routines graphiques	10
3.5. Affichage de texte	11
3.6. Format des fichiers de fontes	12
3.7. Séquences et animations	13
3.8. Optimisations	14
<b>4. Le système de fichiers embarqué</b>	<b>15</b>
4.1. Utilisation	15
4.2. Limitations	16
<b>5. Premiers pas</b>	<b>17</b>
5.1. Prérequis	17
5.2. Test de la connectivité USB	18
5.3. Localisation	19
5.4. Test du module	19
5.5. Configuration	19
<b>6. Montage et connectique</b>	<b>21</b>
6.1. Fixation	21
6.2. Contraintes d'alimentation par USB	22
6.3. Compatibilité électromagnétique (EMI)	22

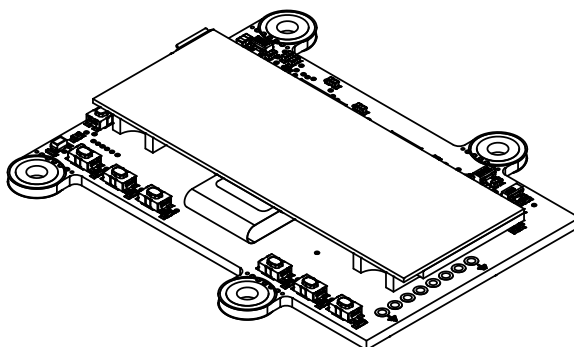
<b>7. Programmation, concepts généraux</b>	<b>25</b>
7.1. <i>Paradigme de programmation</i>	25
7.2. <i>Le module Yocto-Display</i>	27
7.3. <i>Module</i>	28
7.4. <i>Display</i>	29
7.5. <i>AnButton</i>	30
7.6. <i>Files</i>	32
7.7. <i>Quelle interface: Native, DLL ou Service?</i>	32
7.8. <i>Accéder aux modules à travers un hub</i>	35
7.9. <i>Programmation, par où commencer?</i>	35
<b>8. Utilisation du Yocto-Display en ligne de commande</b>	<b>37</b>
8.1. <i>Installation</i>	37
8.2. <i>Utilisation: description générale</i>	37
8.3. <i>Contrôle de la fonction Display</i>	38
8.4. <i>Contrôle de la partie module</i>	39
8.5. <i>Limitations</i>	39
<b>9. Utilisation du Yocto-Display en Python</b>	<b>41</b>
9.1. <i>Fichiers sources</i>	41
9.2. <i>Librairie dynamique</i>	41
9.3. <i>Contrôle de la fonction Display</i>	41
9.4. <i>Contrôle de la partie module</i>	44
9.5. <i>Gestion des erreurs</i>	46
<b>10. Utilisation du Yocto-Display en C++</b>	<b>49</b>
10.1. <i>Contrôle de la fonction Display</i>	49
10.2. <i>Contrôle de la partie module</i>	52
10.3. <i>Gestion des erreurs</i>	54
10.4. <i>Intégration de la librairie Yoctopuce en C++</i>	55
<b>11. Utilisation du Yocto-Display en C#</b>	<b>57</b>
11.1. <i>Installation</i>	57
11.2. <i>Utilisation l'API yoctopuce dans un projet Visual C#</i>	57
11.3. <i>Contrôle de la fonction Display</i>	58
11.4. <i>Contrôle de la partie module</i>	60
11.5. <i>Gestion des erreurs</i>	63
<b>12. Utilisation du Yocto-Display avec LabVIEW</b>	<b>65</b>
12.1. <i>Architecture</i>	65
12.2. <i>Compatibilité</i>	66
12.3. <i>Installation</i>	66
12.4. <i>Présentation des VIs Yoctopuce</i>	71
12.5. <i>Fonctionnement et utilisation des VIs</i>	74
12.6. <i>Utilisation des objets</i>	76
12.7. <i>Gestion du datalogger</i>	78
12.8. <i>Énumération de fonctions</i>	79
12.9. <i>Un mot sur les performances</i>	80
12.10. <i>Un exemple complet de programme LabVIEW</i>	80
12.11. <i>Différences avec les autres API Yoctopuce</i>	81

<b>13. Utilisation du Yocto-Display en Java</b> .....	<b>83</b>
13.1. Préparation .....	83
13.2. Contrôle de la fonction <i>Display</i> .....	83
13.3. Contrôle de la partie module .....	86
13.4. Gestion des erreurs .....	88
<b>14. Utilisation du Yocto-Display avec Android</b> .....	<b>91</b>
14.1. Accès Natif et <i>VirtualHub</i> .....	91
14.2. Préparation .....	91
14.3. Compatibilité .....	91
14.4. Activer le port USB sous Android .....	92
14.5. Contrôle de la fonction <i>Display</i> .....	93
14.6. Contrôle de la partie module .....	96
14.7. Gestion des erreurs .....	101
<b>15. Utilisation du Yocto-Display en TypeScript</b> .....	<b>103</b>
15.1. Utiliser la librairie <i>Yoctopuce</i> pour <i>TypeScript</i> .....	104
15.2. Petit rappel sur les fonctions asynchrones en <i>JavaScript</i> .....	104
15.3. Contrôle de la fonction <i>Display</i> .....	105
15.4. Contrôle de la partie module .....	109
15.5. Gestion des erreurs .....	111
<b>16. Utilisation du Yocto-Display en JavaScript / EcmaScript</b> .....	<b>113</b>
16.1. Fonctions bloquantes et fonctions asynchrones en <i>JavaScript</i> .....	114
16.2. Utiliser la librairie <i>Yoctopuce</i> pour <i>JavaScript / EcmaScript 2017</i> .....	115
16.3. Contrôle de la fonction <i>Display</i> .....	117
16.4. Contrôle de la partie module .....	121
16.5. Gestion des erreurs .....	123
<b>17. Utilisation du Yocto-Display en PHP</b> .....	<b>125</b>
17.1. Préparation .....	125
17.2. Contrôle de la fonction <i>Display</i> .....	126
17.3. Contrôle de la partie module .....	128
17.4. API par callback HTTP et filtres NAT .....	130
17.5. Gestion des erreurs .....	134
<b>18. Utilisation du Yocto-Display en VisualBasic .NET</b> .....	<b>135</b>
18.1. Installation .....	135
18.2. Utilisation l'API <i>yoctopuce</i> dans un projet <i>Visual Basic</i> .....	135
18.3. Contrôle de la fonction <i>Display</i> .....	136
18.4. Contrôle de la partie module .....	138
18.5. Gestion des erreurs .....	140
<b>19. Utilisation du Yocto-Display en Delphi / Lazarus</b> .....	<b>143</b>
19.1. Préparation .....	143
19.2. Contrôle de la fonction <i>Display</i> .....	144
19.3. Contrôle de la partie module .....	147
19.4. Gestion des erreurs .....	149
<b>20. Utilisation du Yocto-Display avec Universal Windows Platform</b> .....	<b>151</b>
20.1. Fonctions bloquantes et fonctions asynchrones .....	151

20.2. <i>Installation</i> .....	152
20.3. <i>Utilisation l'API Yoctopuce dans un projet Visual Studio</i> .....	152
20.4. <i>Contrôle de la fonction Display</i> .....	153
20.5. <i>Un exemple concret</i> .....	154
20.6. <i>Contrôle de la partie module</i> .....	155
20.7. <i>Gestion des erreurs</i> .....	158
<b>21. Utilisation du Yocto-Display en Objective-C</b> .....	<b>159</b>
21.1. <i>Contrôle de la fonction Display</i> .....	159
21.2. <i>Contrôle de la partie module</i> .....	162
21.3. <i>Gestion des erreurs</i> .....	164
<b>22. Utilisation avec des langages non supportés</b> .....	<b>167</b>
22.1. <i>Utilisation en ligne de commande</i> .....	167
22.2. <i>Assembly .NET</i> .....	167
22.3. <i>Virtual Hub et HTTP GET</i> .....	169
22.4. <i>Utilisation des bibliothèques dynamiques</i> .....	171
22.5. <i>Port de la bibliothèque haut niveau</i> .....	174
<b>23. Programmation avancée</b> .....	<b>175</b>
23.1. <i>Programmation par événements</i> .....	175
<b>24. Référence de l'API de haut niveau</b> .....	<b>177</b>
24.1. <i>La classe YAPI</i> .....	178
24.2. <i>La classe YModule</i> .....	182
24.3. <i>La classe YDisplay</i> .....	189
24.4. <i>La classe YDisplayLayer</i> .....	196
24.5. <i>La classe YAnButton</i> .....	199
24.6. <i>La classe YFiles</i> .....	205
<b>25. Problèmes courants</b> .....	<b>211</b>
25.1. <i>Par où commencer ?</i> .....	211
25.2. <i>Linux et USB</i> .....	211
25.3. <i>Plateformes ARM: HF et EL</i> .....	212
25.4. <i>Les exemples de programmation n'ont pas l'air de marcher</i> .....	212
25.5. <i>Module alimenté mais invisible pour l'OS</i> .....	212
25.6. <i>Another process named xxx is already using yAPI</i> .....	212
25.7. <i>Déconnexions, comportement erratique</i> .....	213
25.8. <i>Le module ne marche plus après une mise à jour ratée</i> .....	213
25.9. <i>RegisterHub d'une instance de VirtualHub déconnecte la précédente</i> .....	213
25.10. <i>Commandes ignorées</i> .....	213
25.11. <i>Module endommagé</i> .....	213
<b>26. Caractéristiques</b> .....	<b>215</b>
<i>Blueprint</i> .....	217

# 1. Introduction

Le Yocto-Display est un module électronique de 66x58mm qui permet de piloter un écran OLED monochrome de 128 x 32px ainsi que 6 canaux qui permettant de mesurer l'état d'interrupteurs, boutons poussoir, ou encore de potentiomètres. Cet écran vous permettra d'afficher facilement quelques informations très lisibles depuis une machine qui n'est normalement pas munie d'un moniteur.



*Le module Yocto-Display*

Le Yocto-Display n'est pas en lui-même un produit complet. C'est un composant destiné à être intégré dans une solution d'automatisation en laboratoire, ou pour le contrôle de procédés industriels, ou pour des applications similaires en milieu résidentiel ou commercial. Pour pouvoir l'utiliser, il faut au minimum l'installer à l'intérieur d'un boîtier de protection et le raccorder à un ordinateur de contrôle.

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

Adresse e-mail: [support@yoctopuce.com](mailto:support@yoctopuce.com)

Site Internet: [www.yoctopuce.com](http://www.yoctopuce.com)

Adresse postale: Route de Cartigny 33

Localité: 1236 Cartigny

Pays: Suisse

## 1.1. Informations de sécurité

Le Yocto-Display est conçu pour respecter la norme de sécurité IEC 61010-1:2010. Il ne causera pas de danger majeur pour l'opérateur et la zone environnante, même en condition de premier défaut, pour autant qu'il soit intégré et utilisé conformément aux instructions contenues dans cette documentation, et en particulier dans cette section.

### Boîtier de protection

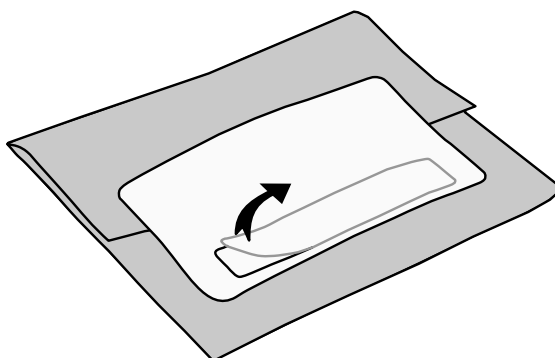
Le Yocto-Display ne doit pas être utilisé sans boîtier de protection, en raison des composants électriques à nu. Pour une sécurité optimale, il devrait être mis dans un boîtier non métallique, non-inflammable, résistant à un choc de 5 J, par exemple en polycarbonate (LEXAN ou autre) d'indice de protection IK08 et classifié V-1 ou mieux selon la norme IEC 60695-11-10. L'utilisation d'un boîtier de qualité inférieure peut nécessiter des avertissements spécifiques pour l'utilisateur et/ou compromettre la conformité avec la norme de sécurité.

### Entretien

Si un dégat est constaté sur le circuit électronique ou sur le boîtier, il doit être remplacé afin de ne pas compromettre la sécurité d'utilisation et d'éviter d'endommager d'autres parties du système par les surcharges éventuelles que pourrait causer un court-circuit.

### Identification

Pour faciliter l'entretien du circuit et l'identification des risques lors de la maintenance, vous devriez coller l'étiquette autocollante synthétique identifiant le Yocto-Display, fournie avec le circuit électronique, à proximité immédiate du module. Si le module est dans un boîtier dédié, l'étiquette devrait être collée sur la surface extérieur du boîtier. L'étiquette est résistante à l'humidité et au frottement usuel qui peut survenir durant un entretien normal.



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

### Applications

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

En particulier, le Yocto-Display n'est *pas* certifié pour utilisation dans un environnement médical, ni pour les applications critiques à la santé, ni pour toute autre application menaçant la vie humaine.

### Environnement

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



## 1.2. Conditions environnementales

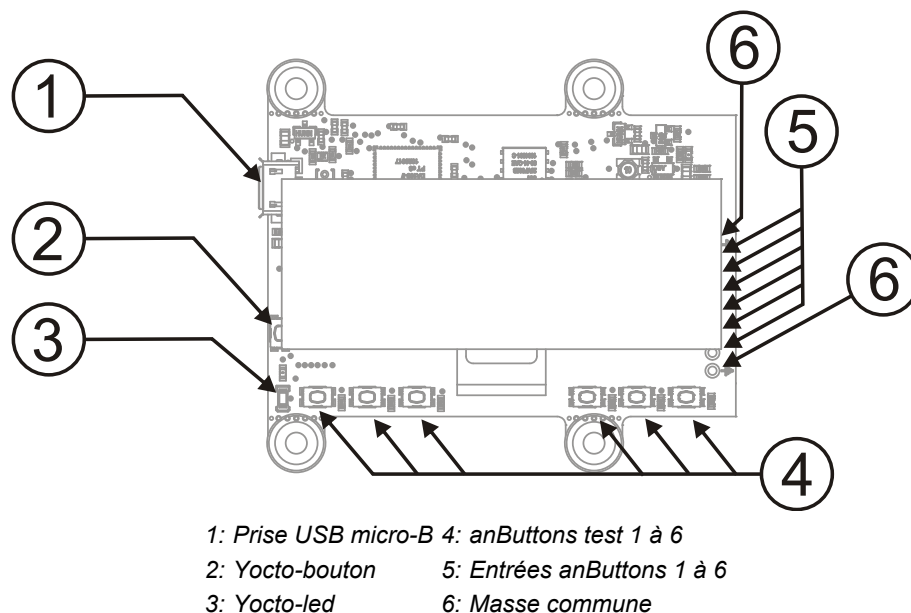
Les produits Yoctopuce sont conçus pour une utilisation intérieure dans un environnement usuel de bureau ou de laboratoire (*degré de pollution 2* selon IEC 60664): la pollution de l'air doit être faible et essentiellement non conductrice. L'humidité relative prévue est de 10% à 90% RH, sans condensation.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

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

La plage de température de fonctionnement étendue du Yocto-Display est -25...70°C. Cette plage de température a été déterminée en fonction des recommandations officielles des fabricants des composants utilisés dans le Yocto-Display, et par des tests de durée limitée (1h) dans les conditions extrêmes, en environnement contrôlé. Si vous envisagez d'utiliser le Yocto-Display dans des conditions de température extrêmes pour une période prolongée, il est recommandé de faire des tests extensifs avant la mise en production.



## 2. Présentation



### 2.1. Les éléments communs

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

#### Le connecteur USB

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

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

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

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

### Le Yocto-bouton

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

### La Yocto-Led

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

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

La Yocto-Led a une troisième fonctionnalité moins plaisante: lorsque ce logiciel interne qui contrôle le module rencontre une erreur fatale, elle se met à flasher SOS en morse<sup>1</sup>. 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<sup>2</sup>.

### 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-Display ce numéro commence par YD128X32. Le module peut être piloté par logiciel en utilisant ce numéro de série. Ce numéro de série ne peut pas être changé.

### Le nom logique

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

## 2.2. Les éléments spécifiques

### L'écran

L'écran est un écran OLED fabriqué par WiseChip sous la référence UG-2832ALBCG01. Etant en verre, il est assez fragile, ne laissez pas tomber votre Yocto-Display. Lorsque vous installez votre Yocto-Display veillez à ce que l'écran ne soit soumis à aucune contrainte mécanique. De plus la nappe qui sort de l'écran est particulièrement fragile au niveau de la jonction à l'écran, veillez à ce

---

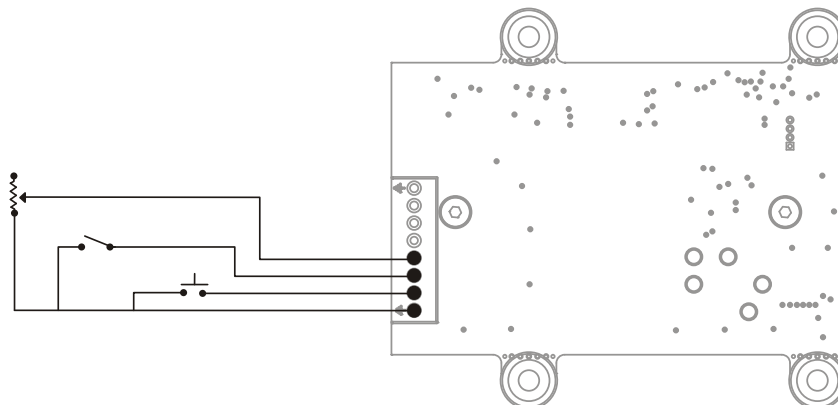
<sup>1</sup> court-court-court long-long-long court-court-court

<sup>2</sup> support@yoctopuce.com

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

### Les entrées 1 à 6

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



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

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

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

### Potentiomètres et calibration

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

### Les boutons poussoirs de test

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

## 2.3. Accessoires optionnels

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

## Micro-hub USB

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

## YoctoHub-Ethernet, YoctoHub-Wireless and YoctoHub-GSM

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

## Connecteurs 1.27mm (ou 1.25mm)

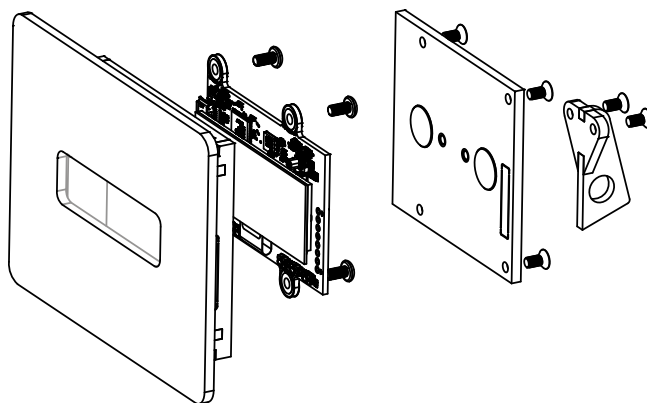
Si vous désirez raccorder le module Yocto-Display à un Micro-hub USB ou à un YoctoHub en évitant l'encombrement d'un vrai câble USB, vous pouvez utiliser les 4 pads au pas 1.27mm juste derrière le connecteur USB. Vous avez alors deux possibilités.

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

Vous pouvez aussi utiliser un petit câble à 4 fils doté de connecteurs au pas 1.27mm (ou 1.25mm, la différence est négligeable pour 4 pins), ce qui vous permet de déporter le module d'une dizaine de centimètres. N'allongez pas trop la distance si vous utilisez ce genre de câble, car il n'est pas blindé et risque donc de provoquer des émissions électromagnétiques indésirables.

## Boîtier

Votre Yocto-Display a été conçu pour pouvoir être installé tel quel dans votre projet. Néanmoins, Yoctopuce commercialise des boîtiers spécialement conçus pour les modules Yoctopuce. Le boîtier recommandé pour votre Yocto-Display est le modèle YoctoBox-Display. Il a un petit pied amovible qui lui permet de tenir debout, il dispose aussi d'aimants intégrés puissants qui lui permettent de tenir sur des surfaces ferromagnétiques. Vous trouverez plus d'informations à propos de ce boîtier sur le site de Yoctopuce<sup>3</sup>.



*Vous pouvez installer votre Yocto-Display dans un boîtier optionnel.*

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

## 3. Principes de fonctionnement

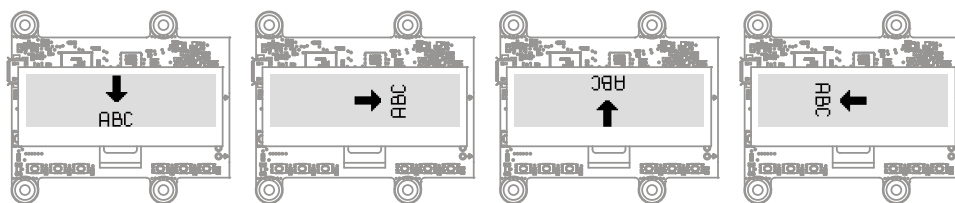
### 3.1. Processeur et mémoire embarqués

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

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

### 3.2. Orientation

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

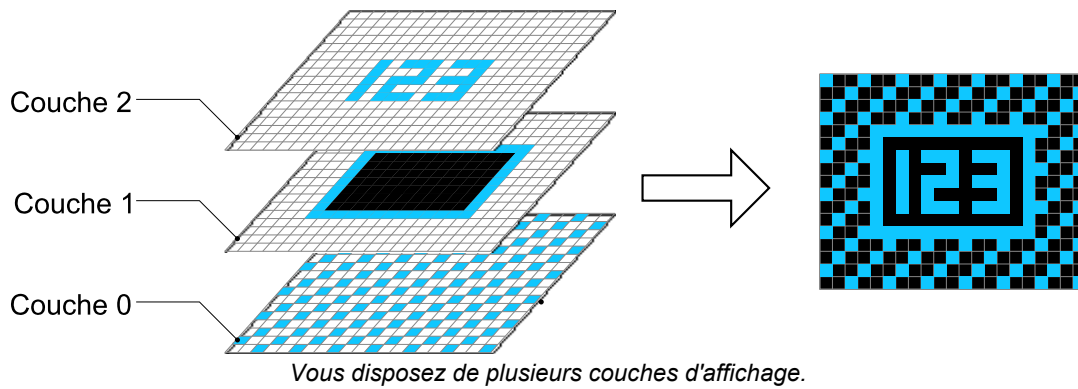


Effet des paramètres LEFT, UP, RIGHT et DOWN sur l'orientation de l'affichage.

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

### 3.3. Système de couche

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



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

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

Les primitives permettant de manipuler directement les couches sont:

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

## 3.4. Routines graphiques

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

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

Les primitives graphiques de base sont:

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

<sup>1</sup> [http://en.wikipedia.org/wiki/Multiple\\_buffering](http://en.wikipedia.org/wiki/Multiple_buffering)



## Couleurs

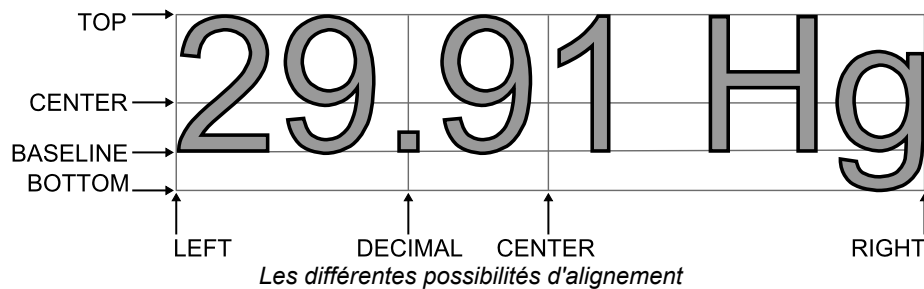
L'écran du Yocto-Display est purement monochrome. Vous ne pouvez donc pas afficher de niveaux de gris, ni bénéficier d'anti-aliasing. Vous pouvez dessiner en trois "couleurs": la couleur d'affichage de l'écran (que nous appellerons "blanc" dans cette documentation, même si il s'agit de bleu clair par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

Les primitives permettant de changer de couleur de dessin sont:

- `selectGrayPen`
- `selectEraser`

## 3.5. Affichage de texte

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



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

- `selectFont`
- `drawText`

Les polices de caractères préchargées sur le module sont:

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

## Mode console

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

Les primitives permettant de gérer la console sont:

- `clearConsole`
- `consoleOut`
- `setConsoleMargins`
- `setConsoleBackground`
- `setConsoleWordWrap`

## Internationalisation et caractères régionaux

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

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

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

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

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

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

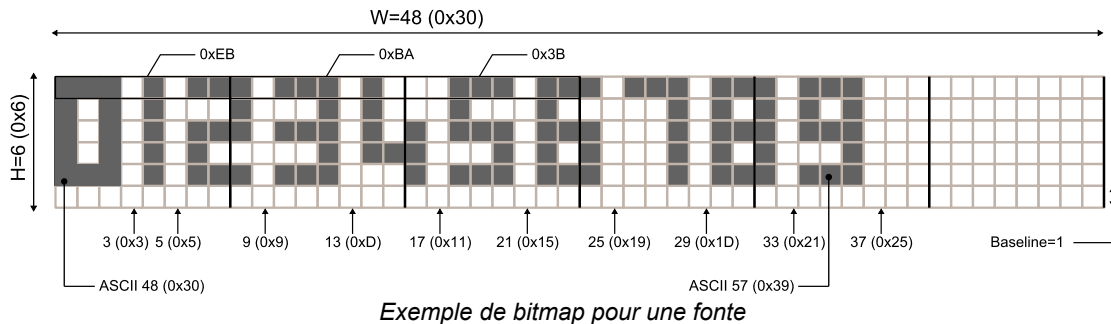
## 3.6. Format des fichiers de fontes

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

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

## Exemple pratique

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



Le fichier de fonte contiendra alors les données suivantes:

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

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

Vous trouverez dans la librairie Delphi un petit utilitaire Windows<sup>2</sup> vous permettant de générer des fichiers de fontes à partir des fontes système.

## 3.7. Séquences et animations

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

Vous trouverez dans les librairies un exemple de code<sup>3</sup> illustrant le fonctionnement des séquences. Dès que cet exemple sera exécuté, l'écran se mettra à jouer la séquence indéfiniment.

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

### Séquence de démarrage

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

<sup>2</sup> Examples\Display-font-generator

<sup>3</sup> Prog-Display-Sequences

## 3.8. Optimisations

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

### Écriture dans les couche cachées

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

### Double buffering

La technique appelée *double buffering* permet d'afficher des animations sans que les artefacts lié à la construction des graphismes ne soient visibles. Elle consiste à travailler sur deux couches, une visible et une seconde cachée. Les images sont construites dans la couche invisible et à la fin de chaque construction, le contenu des deux couches est permuté. Vous trouverez dans les librairies un exemple<sup>4</sup> utilisant cette technique pour animer un flocon de Von Koch.

### Utilisation de bitmap

A partir d'une certaine complexité graphique, il sera plus probablement plus efficace de calculer un bitmap directement sur la machine hôte et de l'envoyer à l'écran. Pour cela il suffit d'utiliser la fonction *drawBitmap*. Les données d'un bitmap sont encodées dans un tableau de bytes, lignes par lignes en partant du coin supérieur gauche. Dans chaque byte, le bit de point le plus fort représente le pixel le plus à gauche. Vous trouverez dans les librairies un exemple<sup>5</sup> calculant un ensemble de Mandelbrot basé sur ce principe.

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

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

---

<sup>4</sup> Prog-Display-DoubleBuffering

<sup>5</sup> Prog-Display-DrawBitmap

## 4. Le système de fichiers embarqué

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

### 4.1. Utilisation

#### Utilisation interactive avec le virtual hub

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

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

#### Utilisation programmée

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

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

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

système de fichier Yoctopuce est la même que celle d'Ethernet, Gzip, PNG, etc. Sa valeur caractéristique pour la chaîne de neuf caractères "123456789" est 0xCBf43926.

### Utilisation par HTTP

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

## 4.2. Limitations

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

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

## 5. Premiers pas

Par design, tous les modules Yoctopuce se pilotent de la même façon, c'est pourquoi les documentations des modules de la gamme sont très semblables. Si vous avez déjà épluché la documentation d'un autre module Yoctopuce, vous pouvez directement sauter à la description de sa configuration.

### 5.1. Prérequis

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

#### Un ordinateur

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

Yoctopuce fournit les bibliothèques logicielles permettant de piloter ses modules pour les systèmes d'exploitation suivants: **Windows, Linux, macOS et Android**. Les modules Yoctopuce ne nécessitent pas l'installation de driver (ou pilote) spécifiques, car ils utilisent le driver HID<sup>1</sup> fourni en standard dans tous les systèmes d'exploitation.

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

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

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

- Windows: Intel 64 bits et 32 bits

---

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

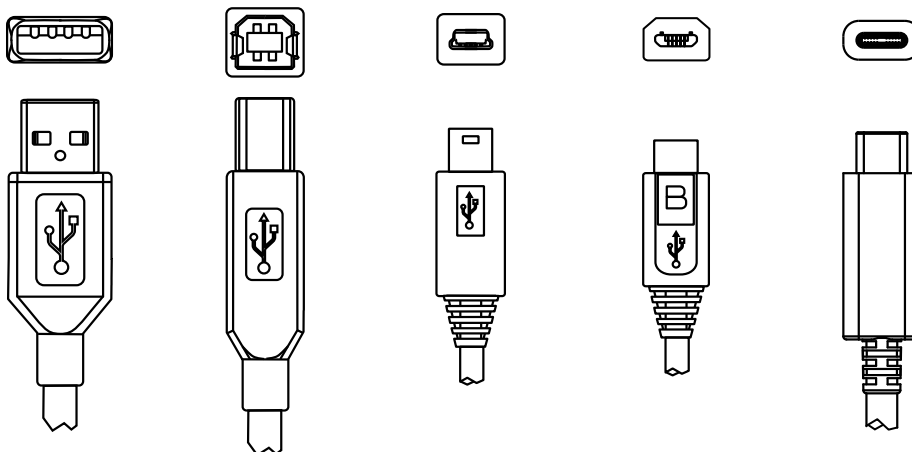
- Linux: Intel 64 bits et 32 bits, ARM 64 bits et 32 bits, y compris Raspberry Pi OS.
- macOS: Intel 64 bits et Apple Silicon (ARM)

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

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

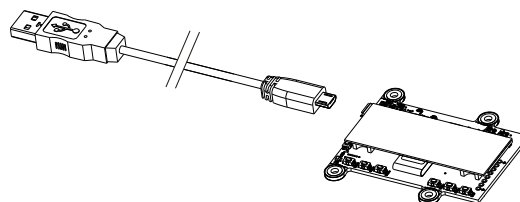
## Un câble USB 2.0 de type A-micro B

Il existe plusieurs formes de connecteurs USB. La taille "normale" correspond à celle que vous utilisez probablement pour brancher votre imprimante. La taille "mini" a plus ou moins disparu. La taille "micro" était la plus petite au moment où les modules Yoctopuce sont apparus, et c'est toujours celle que nous utilisons. Depuis quelques années, des connecteurs USB-C sont apparus, mais pour ne pas multiplier le type de connecteurs dans notre gamme de produit, nous sommes pour l'instant resté au standard "micro-B".



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

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



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

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

## 5.2. Test de la connectivité USB

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



Rendez-vous sur le site de Yoctopuce et téléchargez le programme *VirtualHub*<sup>2</sup>, Il est disponible pour Windows, Linux et macOS. En temps normal le programme VirtualHub sert de couche d'abstraction pour les langages qui ne peuvent pas accéder aux couches matérielles de votre ordinateur. Mais il offre aussi une interface sommaire pour configurer vos modules et tester les fonctions de base, on accède à cette interface à l'aide d'un simple browser web<sup>3</sup>. Lancez VirtualHub en ligne de commande, ouvrez votre browser préféré et tapez l'adresse `http://127.0.0.1:4444`. Vous devriez voir apparaître la liste des modules Yoctopuce raccordés à votre ordinateur.

Serial	Logical Name	Description	Action
VIRTHUB0-698728048		VirtualHub	<input type="button" value="configure"/> <input type="button" value="view log file"/>
YD128X32-0AF8D		Yocto-Display	<input type="button" value="configure"/> <input type="button" value="view log file"/> <input type="button" value="beacon"/>

Search:

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

### 5.3. Localisation

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

### 5.4. Test du module

La première chose à vérifier est le bon fonctionnement de votre module: cliquez sur le numéro de série correspondant à votre module, et une fenêtre résumant les propriétés de votre Yocto-Display.

YD128X32-0AF8D

Edit parameters for device YD128X32-0AF8D, and click on the **Save** button.

Serial # YD128X32-0AF8D  
 Product name: Yocto-Display  
 Firmware: 10325   
 Logical name:   
 Luminosity:  (signal leds only)

**Device functions**

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

YD128X32-0AF8D.anButton1 /    
 YD128X32-0AF8D.anButton2 /    
 YD128X32-0AF8D.anButton3 /    
 YD128X32-0AF8D.anButton4 /    
 YD128X32-0AF8D.anButton5 /    
 YD128X32-0AF8D.anButton6 /    
 YD128X32-0AF8D.display /

Orientation:

Brightness:

Start up sequence:

YD128X32-0AF8D.files /

User files: 0 file, 3808 KB available

Propriétés du module Yocto-Display.

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

### 5.5. Configuration

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

<sup>2</sup> [www.yoctopuce.com/FR/virtualhub.php](http://www.yoctopuce.com/FR/virtualhub.php)

<sup>3</sup> L'interface est testée avec Chrome, FireFox, Safari, Edge et IE 11.

Configuration du module Yocto-Display.

## Firmware

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

Pour mettre à jour un firmware, cliquez simplement sur le bouton **upgrade** de la fenêtre de configuration et suivez les instructions. Si pour une raison ou une autre, la mise à jour venait à échouer, débranchez puis rebranchez le module. Recommencer la procédure devrait résoudre alors le problème. Si le module a été débranché alors qu'il était en cours de reprogrammation, il ne fonctionnera probablement plus et ne sera plus listé dans l'interface. Mais il sera toujours possible de le reprogrammer correctement en utilisant le programme VirtualHub<sup>4</sup> en ligne de commande<sup>5</sup>.

## Nom logique du module

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

## Luminosité

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

## Nom logique des fonctions

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

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

<sup>4</sup> [www.yoctopuce.com/FR/virtualhub.php](http://www.yoctopuce.com/FR/virtualhub.php)

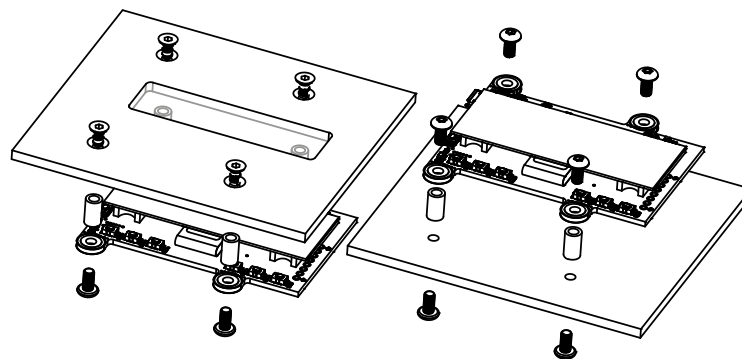
<sup>5</sup> Consultez la documentation de VirtualHub pour plus de détails

## 6. Montage et connectique

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

### 6.1. Fixation

Pendant la mise au point de votre projet vous pouvez vous contenter de laisser le module se promener au bout de son câble. Veillez simplement à ce qu'il ne soit pas en contact avec quoi que soit de conducteur (comme vos outils). Une fois votre projet pratiquement terminé il faudra penser à faire en sorte que vos modules ne puissent pas se promener à l'intérieur.



*Exemples de montage sur un support.*

Le module Yocto-Display dispose de trous de montage 3mm. Vous pouvez utiliser ces trous pour y passer des vis. Le diamètre de la tête de ces vis ne devra pas dépasser 8mm, sous peine d'endommager les circuits du module. Veillez à ce que la surface inférieure du module ne soit pas en contact avec le support. La méthode recommandée consiste à utiliser des entretoises, mais il en existe d'autres. Rien ne vous empêche de le fixer au pistolet à colle; ça ne sera pas très joli mais ça tiendra.

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

## 6.2. Contraintes d'alimentation par USB

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

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

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

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

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

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

## 6.3. Compatibilité électromagnétique (EMI)

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

Les mesures de référence que nous effectuons pour valider la conformité avec la norme IEC CISPR 11 sont faites sans aucun boîtier, mais en raccordant les modules par un câble USB blindé, conforme à la spécification USB 2.0: le blindage du câble est relié au blindage des deux connecteurs, et la résistance totale entre le blindage des deux connecteurs est inférieure 0.6Ω. Le câble utilisé fait 3m, de sorte à exposer un segment d'un mètre horizontal, un segment d'un mètre vertical et de garder le dernier mètre le plus proche de l'ordinateur hôte à l'intérieur d'un bloc de ferrite.

Si vous utilisez un câble non blindé ou incorrectement blindé, votre système fonctionnera sans problème mais vous risquez de n'être pas conforme à la norme. Dans le cadre de systèmes composés de plusieurs modules raccordés par des câbles au pas 1.27mm, ou de capteurs déportés,

<sup>1</sup> [www.yoctopuce.com/FR/article/cables-usb-la-taille-compte](http://www.yoctopuce.com/FR/article/cables-usb-la-taille-compte)

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

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



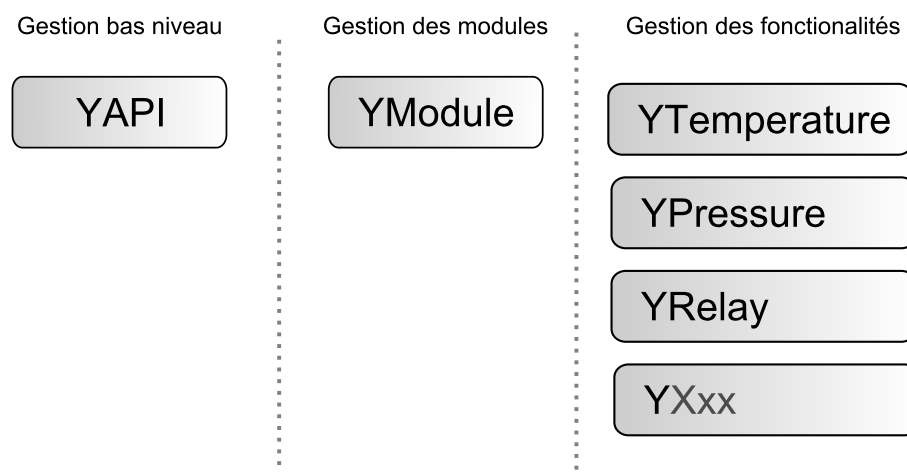
## 7. Programmation, concepts généraux

L'API Yoctopuce a été pensée pour être à la fois simple à utiliser, et suffisamment générique pour que les concepts utilisés soient valables pour tous les modules de la gamme Yoctopuce et ce dans tous les langages de programmation disponibles. Ainsi, une fois que vous aurez compris comment piloter votre Yocto-Display dans votre langage de programmation favori, il est très probable qu'apprendre à utiliser un autre module, même dans un autre langage, ne vous prendra qu'un minimum de temps.

### 7.1. Paradigme de programmation

L'API Yoctopuce est une API orientée objet. Mais, dans un souci de simplicité, seules les bases de la programmation objet ont été utilisées. Même si la programmation objet ne vous est pas familière, il est peu probable que cela vous soit un obstacle à l'utilisation des produits Yoctopuce. Notez que vous n'aurez jamais à allouer ou désallouer un objet lié à l'API Yoctopuce: cela est géré automatiquement.

Il existe une classe par type de fonctionnalité Yoctopuce. Le nom de ces classes commence toujours par un Y suivi du nom de la fonctionnalité, par exemple *YTemperature*, *YRelay*, *YPressure*, etc.. Il existe aussi une classe *YModule*, dédiée à la gestion des modules en temps que tels, et enfin il existe la classe statique *YAPI*, qui supervise le fonctionnement global de l'API et gère les communications à bas niveau.



Structure de l'API Yoctopuce.

## La classe YSensor

A chaque fonctionnalité d'un module Yoctopuce, correspond une classe: YTemperature pour mesurer la température, YVoltage pour mesurer une tension, YRelay pour contrôler un relais, etc. Il existe cependant une classe spéciale qui peut faire plus: YSensor.

Cette classe YSensor est la classe parente de tous les senseurs Yoctopuce, elle permet de contrôler n'importe quel senseur, quel que soit son type, en donnant accès aux fonctions communes à tous les senseurs. Cette classe permet de simplifier la programmation d'applications qui utilisent beaucoup de senseurs différents. Mieux encore, si vous programmez une application basée sur la classe YSensor, elle sera compatible avec tous les senseurs Yoctopuce, y compris ceux qui n'existent pas encore.

## Programmation

Dans l'API Yoctopuce, la priorité a été mise sur la facilité d'accès aux fonctionnalités des modules en offrant la possibilité de faire abstraction des modules qui les implémentent. Ainsi, il est parfaitement possible de travailler avec un ensemble de fonctionnalités sans jamais savoir exactement quel module les héberge au niveau matériel. Cela permet de considérablement simplifier la programmation de projets comprenant un nombre important de modules.

Du point de vue programmation, votre Yocto-Display 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 Fonction / Module

Du point de vue de l'API, les modules et leurs fonctionnalités sont donc fortement décorrélés à dessein. Mais l'API offre néanmoins la possibilité de passer de l'un à l'autre. Ainsi la méthode *get\_module()*, disponible dans chaque classe de fonctionnalité, permet de retrouver l'objet correspondant au module hébergeant cette fonctionnalité. Inversement, la classe *YModule* dispose d'un certain nombre de méthodes permettant d'énumérer les fonctionnalités disponibles sur un module.

## 7.2. Le module Yocto-Display

Le module Yocto-Display est un écran OLED de 128x32 pixels. Il inclut un filesystem permettant de stocker des fichiers (images, polices, séquences) et six instances de la fonction *AnButton*, correspondant aux six entrées analogiques (lecture de potentiomètre ou de bouton) présentes sur le module.

### module : Module

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

### display : Display

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

**files : Files**

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

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

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

## 7.3. Module

Interface de contrôle des paramètres généraux des modules Yoctopuce

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

### productName

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

### serialNumber

Chaîne de caractères contenant le numéro de série, unique et préprogrammé en usine. Pour un module Yocto-Display, ce numéro de série commence toujours par YD128X32. Il peut servir comme point de départ pour accéder par programmation à un module particulier.

### logicalName

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

### productId

Identifiant USB du module, préprogrammé à la valeur 45 en usine.

### **productRelease**

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

### **firmwareRelease**

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

### **persistentSettings**

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

### **luminosity**

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

### **beacon**

Etat de la balise de localisation du module.

### **upTime**

Temps écoulé depuis la dernière mise sous tension du module.

### **usbCurrent**

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

### **rebootCountdown**

Compte à rebours pour déclencher un redémarrage spontané du module.

### **userVar**

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

## **7.4. Display**

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

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

### **logicalName**

Chaîne de caractères contenant le nom logique de l'écran, initialement vide. Cet attribut peut être changé au bon vouloir de l'utilisateur. Un fois initialisé à une valeur non vide, il peut servir de point de départ pour accéder directement à l'écran. Si deux écrans portent le même nom logique dans un projet, il n'y a pas moyen de déterminer lequel va répondre si l'on tente un accès par ce nom logique. Le nom logique du module est limité à 19 caractères parmi A..Z, a..z, 0..9, \_ et -.

### **advertisedValue**

Courte chaîne de caractères résumant l'état actuel de l'écran, et qui sera publiée automatiquement jusqu'au hub parent. Pour un écran, la valeur publiée est son état (ON ou OFF).

### **enabled**

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

### **startupSeq**

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

### **brightness**

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

### **orientation**

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

### **displayWidth**

Largeur de l'écran, en pixels.

### **displayHeight**

Hauteur de l'écran, en pixels.

### **displayType**

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

### **layerWidth**

Largeur des couches affichables, en pixels.

### **layerHeight**

Hauteur des couches affichables, en pixels.

### **layerCount**

Nombre des couches affichables disponibles.

### **command**

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

## **7.5. AnButton**

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

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

**logicalName**

Chaîne de caractères contenant le nom logique de l'entrée analogique, initialement vide. Cet attribut peut être changé au bon vouloir de l'utilisateur. Un fois initialisé à une valeur non vide, il peut servir de point de départ pour accéder à directement à l'entrée analogique. Si deux entrées analogiques portent le même nom logique dans un projet, il n'y a pas moyen de déterminer lequel va répondre si l'on tente un accès par ce nom logique. Le nom logique du module est limité à 19 caractères parmi A..Z,a..z,0..9,\_ et -.

**advertisedValue**

Courte chaîne de caractères résumant l'état actuel de l'entrée analogique, et qui sera publiée automatiquement jusqu'au hub parent. Pour une entrée analogique, la valeur publiée est la valeur mesurée recalibrée (a number between 0 and 1000).

**calibratedValue**

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

**rawValue**

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

**analogCalibration**

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

**calibrationMax**

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

**calibrationMin**

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

**sensitivity**

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

**isPressed**

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

**lastTimePressed**

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

### **lastTimeReleased**

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

### **pulseCounter**

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

### **pulseTimer**

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

### **inputType**

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

## **7.6. Files**

Interface pour interagir avec les systèmes de fichier, disponibles par exemple dans le Yocto-Color-V2, le Yocto-SPI, le YoctoHub-Ethernet et le YoctoHub-GSM-4G

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

### **logicalName**

Chaîne de caractères contenant le nom logique du système de fichier, initialement vide. Cet attribut peut être changé au bon vouloir de l'utilisateur. Un fois initialisé à une valeur non vide, il peut servir de point de départ pour accéder à directement au système de fichier. Si deux systèmes de fichier portent le même nom logique dans un projet, il n'y a pas moyen de déterminer lequel va répondre si l'on tente un accès par ce nom logique. Le nom logique du module est limité à 19 caractères parmi `A..Z,a..z,0..9,_` et `-`.

### **advertisedValue**

Courte chaîne de caractères résumant l'état actuel du système de fichier, et qui sera publiée automatiquement jusqu'au hub parent. Pour un système de fichier, la valeur publiée est le nombre de fichiers présents.

### **filesCount**

Nombre de fichiers présents dans le système de fichier.

### **freeSpace**

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

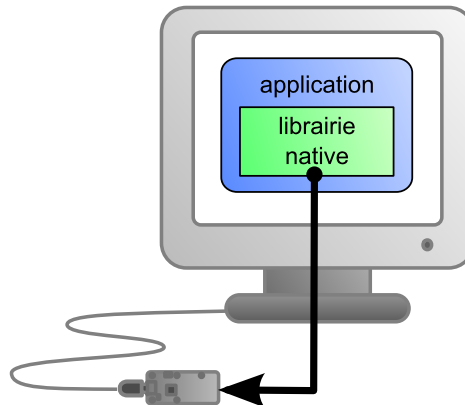
## **7.7. Quelle interface: Native, DLL ou Service?**

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

### **Contrôle natif**

Dans ce cas de figure le programme pilotant votre projet est directement compilé avec une librairie qui offre le contrôle des modules. C'est objectivement la solution la plus simple et la plus élégante

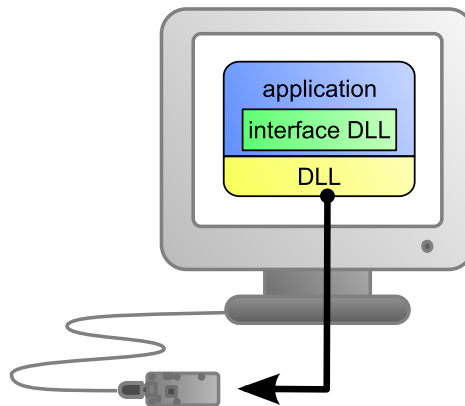
pour l'utilisateur final. Il lui suffira de brancher le câble USB et de lancer votre programme pour que tout fonctionne. Malheureusement, cette technique n'est pas toujours disponible ou même possible.



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

### Contrôle natif par DLL

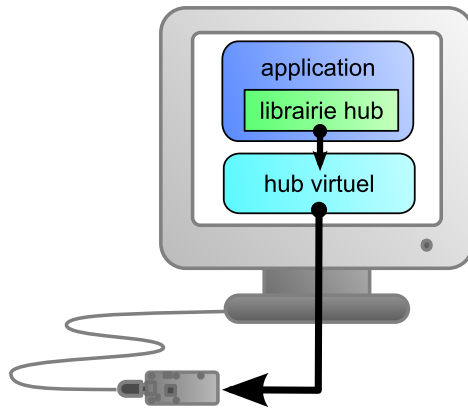
Ici l'essentiel du code permettant de contrôler les modules se trouve dans une DLL, et le programme est compilé avec une petite bibliothèque permettant de contrôler cette DLL. C'est la manière la plus rapide pour coder le support des modules dans un langage particulier. En effet la partie "utile" du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



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

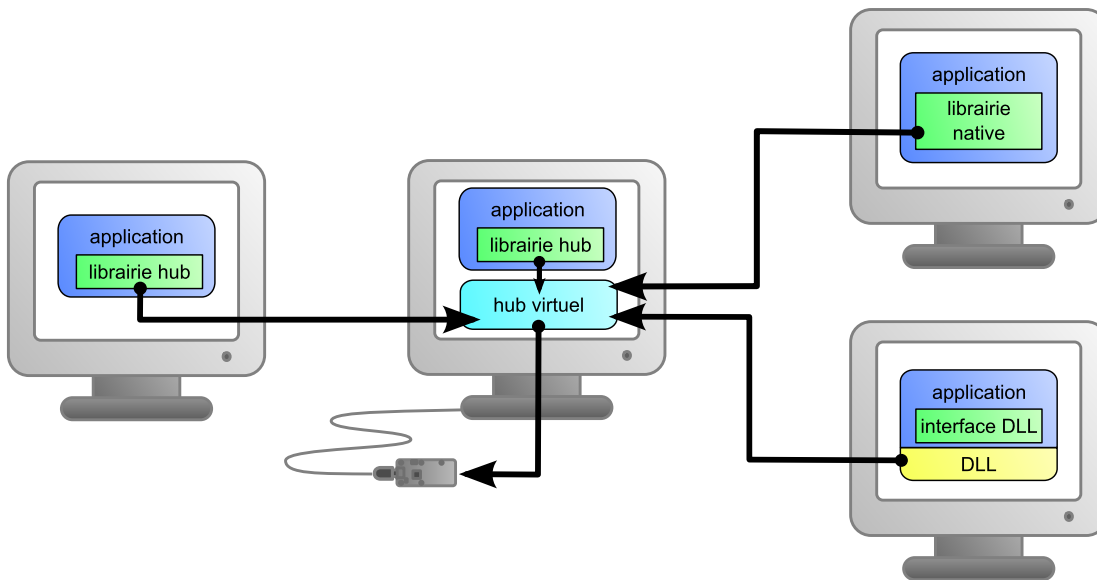
### Contrôle par un service

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



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

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



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

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

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

Méthode de support pour les différents langages.



## 7.8. Accéder aux modules à travers un hub

### VirtualHub pour contourner la limitation d'accès à USB

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

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

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

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

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

### Avec un YoctoHub

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

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

```
YAPI.RegisterHub("usb",errmsg); // utilisation en mode natif USB
YAPI.RegisterHub("127.0.0.1",errmsg); // utilisation en mode réseau local avec VirtualHub
YAPI.RegisterHub("192.168.0.10",errmsg); // utilisation avec YoctoHub dont l'adresse IP est 192.168.0.10
```

## 7.9. Programmation, par où commencer?

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

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

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

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



## 8. Utilisation du Yocto-Display en ligne de commande

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

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

### 8.1. Installation

Téléchargez l'API en ligne de commande<sup>2</sup>. 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-Display, ouvrir un shell et commencer à travailler en tapant par exemple:

```
C:\>YDisplay any -layer 0 drawText 64 32 CENTER "Hello world!"
```

Sous Linux, pour utiliser l'API en ligne de commande, vous devez soit être root, soit définir une règle *udev* pour votre système. Vous trouverez plus de détails au chapitre *Problèmes courants*.

### 8.2. Utilisation: description générale

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

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

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

---

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

<sup>2</sup> <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.

### 8.3. Contrôle de la fonction Display

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

Vous pouvez par exemple lancer:

```
C:\>YDisplay any -layer 0 drawText 64 32 CENTER "Hello world!"
```

Cet exemple utilise la cible "any" pour signifier que l'on désire travailler sur la première fonction Display trouvée parmi toutes celles disponibles sur les modules Yoctopuce accessibles au moment de l'exécution. Cela vous évite d'avoir à connaître le nom exact de votre fonction et celui de votre module.

Mais vous pouvez tout aussi bien utiliser des noms logiques que vous auriez préalablement configurés. Imaginons un module Yocto-Display avec le numéros de série YD128X32-123456 que vous auriez appelé "MonModule" et dont vous auriez nommé la fonction display "MaFonction", les cinq appels suivants seront strictement équivalents (pour autant que MaFonction ne soit définie qu'une fois, pour éviter toute ambiguïté).

```
C:\>YDisplay YD128X32-123456.display describe
C:\>YDisplay YD128X32-123456.MaFonction describe
C:\>YDisplay MonModule.display describe
C:\>YDisplay MonModule.MaFonction describe
C:\>YDisplay MaFonction describe
```

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

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

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

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

## 8.4. Contrôle de la partie module

Chaque module peut être contrôlé d'une manière similaire à l'aide de l'exécutable `YModule`. Par exemple, pour obtenir la liste de tous les modules connectés, utilisez:

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

Vous pouvez aussi utiliser la commande suivante pour obtenir une liste encore plus détaillée des modules connectés:

```
C:\>YModule all describe
```

Chaque propriété `xxx` du module peut être obtenue grâce à une commande du type `get_xxxx()`, et les propriétés qui ne sont pas en lecture seule peuvent être modifiées à l'aide de la commande `set xxx()`. Par exemple:

```
C:\>YModule YD128X32-12346 set_logicalName MonPremierModule
C:\>YModule YD128X32-12346 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 YD128X32-12346 set_logicalName MonPremierModule
C:\>YModule YD128X32-12346 saveToFlash
```

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

```
C:\>YModule -s YD128X32-12346 set_logicalName MonPremierModule
```

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

## 8.5. Limitations

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

Cette limitation peut aisément être contournée en utilisant `VirtualHub`: il suffit de faire tourner `VirtualHub`<sup>3</sup> sur la machine concernée et d'utiliser les exécutables de l'API en ligne de commande avec l'option `-r` par exemple, si vous utilisez:

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

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

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

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

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

## 9. Utilisation du Yocto-Display en Python

Python est un langage interprété orienté objet développé par Guido van Rossum. Il offre l'avantage d'être gratuit et d'être disponible pour la plupart de plate-formes tant Windows qu'Unix. C'est un langage idéal pour écrire des petits scripts sur un coin de table. La librairie Yoctopuce est compatible avec Python 2.7 et 3.x jusqu'aux toutes dernières versions officielles. Elle fonctionne sous Windows, macOS et Linux tant Intel qu'ARM. Les interpréteurs Python sont disponibles sur le site de Python<sup>1</sup>.

### 9.1. Fichiers sources

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

### 9.2. Librairie dynamique

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

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

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

### 9.3. Contrôle de la fonction Display

---

<sup>1</sup> <http://www.python.org/download/>

<sup>2</sup> [www.yoctopuce.com/FR/libraries.php](http://www.yoctopuce.com/FR/libraries.php)

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

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

# On récupère l'objet permettant d'interagir avec le module
display = YDisplay.FindDisplay("YD128X32-123456.display")

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

[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

## YAPI.RegisterHub

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

## YDisplay.FindDisplay

La fonction `YDisplay.FindDisplay` permet de retrouver un écran en fonction du numéro de série de son module hôte et de son nom de fonction. Mais vous pouvez tout aussi bien utiliser des noms logiques que vous auriez préalablement configurés. Imaginons un module Yocto-Display avec le numéro de série `YD128X32-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `display` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128X32-123456.display")
display = YDisplay.FindDisplay("YD128X32-123456.MaFonction")
display = YDisplay.FindDisplay("MonModule.display")
display = YDisplay.FindDisplay("MonModule.MaFonction")
display = YDisplay.FindDisplay("MaFonction")
```

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

## isOnline

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

## A propos des imports Python

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

```
from yocto_api import *
```

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

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



## get\_displayLayer

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

## Un exemple réel

Lancez votre interpréteur Python et ouvrez le script correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-Display** 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_display import *

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

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

errmsg = YRefParam()

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

target = sys.argv[1]

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

if target == 'any':
    # retrieve any RGB led
    disp = YDisplay.FirstDisplay()
    if disp is None:
        die('No module connected')
else:
    disp = YDisplay.FindDisplay(target + ".display")

if not disp.isOnline():
    die("Module not connected ")

# display clean up
disp.resetAll()

# retrieve the display size
w = disp.get_displayWidth()
h = disp.get_displayHeight()

# retrieve the first layer
l0 = disp.get_displayLayer(0)
l0.clear()

# display a text in the middle of the screen
l0.drawText(w / 2, h / 2, YDisplayLayer.ALIGN.CENTER, "Hello world!")

# visualize each corner
l0.moveTo(0, 5)
l0.lineTo(0, 0)
l0.lineTo(5, 0)
```

```

10.moveTo(0, h - 6)
10.lineTo(0, h - 1)
10.lineTo(5, h - 1)
10.moveTo(w - 1, h - 6)
10.lineTo(w - 1, h - 1)
10.lineTo(w - 6, h - 1)
10.moveTo(w - 1, 5)
10.lineTo(w - 1, 0)
10.lineTo(w - 6, 0)

# draw a circle in the top left corner of layer 1
l1 = disp.get_displayLayer(1)
l1.clear()
l1.drawCircle(h / 8, h / 8, h / 8)

# and animate the layer
print("Use Ctrl-C to stop")
x = 0
y = 0
vx = 1
vy = 1
while disp.isOnline():
    x += vx
    y += vy
    if x < 0 or x > w - (h / 4):
        vx = -vx
    if y < 0 or y > h - (h / 4):
        vy = -vy
    l1.setLayerPosition(x, y, 0)
    YAPI.Sleep(5, errmsg)
YAPI.FreeAPI()

```

## 9.4. Contrôle de la partie module

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

```

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

from yocto_api import *

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

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

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

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

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

    print("serial:          " + m.get_serialNumber())
    print("logical name:    " + m.get_logicalName())
    print("luminosity:      " + str(m.get_luminosity()))
    if m.get_beacon() == YModule.BEACON_ON:
        print("beacon:          ON")
    else:
        print("beacon:          OFF")
    print("upTime:          " + str(m.get_upTime() / 1000) + " sec")

```

```

print("USB current: " + str(m.get_usbCurrent()) + " mA")
print("logs:\n" + m.get_lastLogs())
else:
    print(sys.argv[1] + " not connected (check identification and USB cable)")
YAPI.FreeAPI()

```

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

## Modifications des réglages du module

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

```

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

from yocto_api import *

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

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

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

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

```

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

## Enumeration des modules

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

```

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

```

```

from yocto_api import *

errmsg = YRefParam()

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

print('Device list')

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

```

## 9.5. Gestion des erreurs

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

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

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

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

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

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

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



## 10. Utilisation du Yocto-Display en C++

Le C++ n'est pas le langage le plus simple à maîtriser. Pourtant, si on prend soin à se limiter aux fonctionnalités essentielles, c'est un langage tout à fait utilisable pour des petits programmes vite faits, et qui a l'avantage d'être très portable d'un système d'exploitation à l'autre. Sous Windows, C++ est supporté avec Microsoft Visual Studio 2017 et les versions plus récentes. Sous macOS, nous supportons les versions de XCode supportées par Apple. Sous Linux, nous supportons toutes les versions de gcc publiées depuis 2008. Par ailleurs, aussi bien sous macOS que sous Linux, vous pouvez compiler les exemples en ligne de commande avec GCC en utilisant le `GNUmakefile` fourni. De même, sous Windows, un `Makefile` vous permet de compiler les exemples en ligne de commande, en pleine connaissance des arguments de compilation et link.

Les bibliothèques Yoctopuce<sup>1</sup> pour C++ vous sont fournies au format source dans leur intégralité. Une partie de la bibliothèque de bas-niveau est écrite en C pur sucre, mais vous n'aurez à priori pas besoin d'interagir directement avec elle: tout a été fait pour que l'interaction soit le plus simple possible depuis le C++. La bibliothèque vous est fournie bien entendu aussi sous forme binaire, de sorte à pouvoir la linker directement si vous le préférez.

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

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

### 10.1. Contrôle de la fonction Display

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

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

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

---

<sup>1</sup> [www.yoctopuce.com/FR/libraries.php](http://www.yoctopuce.com/FR/libraries.php)

```
[...]
// On récupère l'objet permettant d'interagir avec le module
YDisplay *display;
display = YDisplay::FindDisplay("YD128X32-123456.display");

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

Voyons maintenant en détail ce que font ces quelques lignes.

## yocto\_api.h et yocto\_display.h

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

### YAPI::RegisterHub

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

### YDisplay::FindDisplay

La fonction `YDisplay::FindDisplay` permet de retrouver un écran en fonction du numéro de série de son module hôte et de son nom de fonction. Mais vous pouvez tout aussi bien utiliser des noms logiques que vous auriez préalablement configurés. Imaginons un module Yocto-Display avec le numéros de série `YD128X32-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `display` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YDisplay *display = YDisplay::FindDisplay("YD128X32-123456.display");
YDisplay *display = YDisplay::FindDisplay("YD128X32-123456.MaFonction");
YDisplay *display = YDisplay::FindDisplay("MonModule.display");
YDisplay *display = YDisplay::FindDisplay("MonModule.MaFonction");
YDisplay *display = YDisplay::FindDisplay("MaFonction");
```

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

### isOnline

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

### get\_displayLayer

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

### Un exemple réel

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

using namespace std;

static void usage(void)
{
    cout << "Wrong command line arguments" << endl;
    cout << "usage: demo <serial_number>" << endl;
    cout << "      demo <logical_name>" << endl;
    cout << "      demo any (use any discovered device)" << endl;
    u64 now = YAPI::GetTickCount();
    while (YAPI::GetTickCount() - now < 3000) {
        // wait 3 sec to show the message
    }
    exit(1);
}

int main(int argc, const char * argv[])
{
    string      errmsg;
    string      target;
    YDisplay    *disp;
    YDisplayLayer *l0, *l1;
    int w, h, x, y, vx, vy;

    if(argc < 2) {
        usage();
    }

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

    target      = (string) argv[1];
    if (target == "any") {
        disp = YDisplay::FirstDisplay();
        if (disp == NULL) {
            cout << "No module connected (check USB cable)" << endl;
            usage();
            return 1;
        }
    } else {
        disp = YDisplay::FindDisplay(target + ".display");
    }

    if (!disp->isOnline()) {
        cout << "Module is offline (check USB cable)" << endl;
        usage();
        return 1;
    }

    disp->resetAll();
    // retrieve the display size
    w = disp->get_displayWidth();
    h = disp->get_displayHeight();

    // retrieve the first layer
    l0 = disp->get_displayLayer(0);
    l0->clear();

    // display a text in the middle of the screen
    l0->drawText(w / 2, h / 2, YDisplayLayer::ALIGN_CENTER, "Hello world!" );
    // visualize each corner
    l0->moveTo(0, 5);
    l0->lineTo(0, 0);
    l0->lineTo(5, 0);
    l0->moveTo(0, h - 6);
    l0->lineTo(0, h - 1);
    l0->lineTo(5, h - 1);
    l0->moveTo(w - 1, h - 6);
    l0->lineTo(w - 1, h - 1);
}

```

```

l0->lineTo(w - 6, h - 1);
l0->moveTo(w - 1, 5);
l0->lineTo(w - 1, 0);
l0->lineTo(w - 6, 0);

// draw a circle in the top left corner of layer 1
l1 = disp->get_displayLayer(1);
l1->clear();
l1->drawCircle(h / 8, h / 8, h / 8);

// and animate the layer
cout << "Use Ctrl-C to stop";
x = 0;
y = 0;
vx = 1;
vy = 1;
while (disp->isOnline()) {
    x += vx;
    y += vy;
    if ((x < 0) || (x > w - (h / 4)))
        vx = -vx;
    if ((y < 0) || (y > h - (h / 4)))
        vy = -vy;
    l1->setLayerPosition(x, y, 0);
    YAPI::Sleep(5, errmsg);
}
YAPI::FreeAPI();
return 0;
}

```

## 10.2. Contrôle de la partie module

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

```

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

#include "yocto_api.h"

using namespace std;

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

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

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

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

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

    if (module->isOnline()) {
        if (argc > 2) {
            if (string(argv[2]) == "ON")
                module->set_beacon(Y_BEACON_ON);
            else
                module->set_beacon(Y_BEACON_OFF);
        }
        cout << "serial:      " << module->get_serialNumber() << endl;
    }
}

```

```

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

```

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

## Modifications des réglages du module

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

```

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

#include "yocto_api.h"

using namespace std;

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

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

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

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

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

    if (module->isOnline()) {
        if (argc >= 3) {
            string newname = argv[2];
            if (!YModule::CheckLogicalName(newname)) {
                cerr << "Invalid name (" << newname << ")" << endl;
                usage(argv[0]);
            }
            module->set_logicalName(newname);
            module->saveToFlash();
        }
        cout << "Current name: " << module->get_logicalName() << endl;
    } else {

```

```

    cout << argv[1] << " not connected (check identification and USB cable)"
         << endl;
}
YAPI::FreeAPI();
return 0;
}

```

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

## Enumeration des modules

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

```

#include <iostream>

#include "yocto_api.h"

using namespace std;

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

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

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

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

```

## 10.3. Gestion des erreurs

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

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

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

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

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

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

## 10.4. Intégration de la librairie Yoctopuce en C++

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

### Intégration au format source (recommandé)

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

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

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

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

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

## Intégration en librairie statique

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

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

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

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

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

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

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

## Intégration en librairie dynamique

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

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

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

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

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

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

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

## 11. Utilisation du Yocto-Display en C#

C# (prononcez C-Sharp) est un langage orienté objet promu par Microsoft qui n'est pas sans rappeler Java. Tout comme Visual Basic et Delphi, il permet de créer des applications Windows relativement facilement. C# est supporté sous Windows Visual Studio 2017 et ses versions plus récentes.

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

### 11.1. Installation

Téléchargez la librairie Yoctopuce pour Visual C# depuis le site web de Yoctopuce<sup>1</sup>. Il n'y a pas de programme d'installation, copiez simplement le contenu du fichier zip dans le répertoire de votre choix. Vous avez besoin essentiellement du contenu du répertoire *Sources*. Les autres répertoires contiennent la documentation et quelques programmes d'exemple. Les projets d'exemple sont des projets Visual C# 2010, si vous utilisez une version antérieure, il est possible que vous ayez à reconstruire la structure de ces projets.

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

La librairie Yoctopuce pour Visual C# .NET se présente sous la forme d'une DLL et de fichiers sources en Visual C#. La DLL n'est pas une DLL .NET mais une DLL classique, écrite en C, qui gère les communications à bas niveau avec les modules<sup>2</sup>. Les fichiers sources en Visual C# gèrent la partie haut niveau de l'API. Vous avez donc besoin de cette DLL et des fichiers .cs du répertoire *Sources* pour créer un projet gérant des modules Yoctopuce.

#### Configuration d'un projet Visual C#

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

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

---

<sup>1</sup> [www.yoctopuce.com/FR/libraries.php](http://www.yoctopuce.com/FR/libraries.php)

<sup>2</sup> Les sources de cette DLL sont disponibles dans l'API C++

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

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

Ensuite, ajoutez de la même manière la dll `yapi.dll`, qui se trouve dans le répertoire `Sources/dll`<sup>3</sup>. Puis depuis la fenêtre **Explorateur de solutions**, effectuez un clic droit sur la DLL, choisissez **Propriété** et dans le panneau **Propriétés**, mettez l'option **Copier dans le répertoire de sortie à toujours copier**. Vous êtes maintenant prêt à utiliser vos modules Yoctopuce depuis votre environnement Visual Studio.

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

### 11.3. Contrôle de la fonction Display

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

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

// On récupère l'objet permettant d'interagir avec le module
YDisplay display = YDisplay.FindDisplay("YD128X32-123456.display");

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

Voyons maintenant en détail ce que font ces quelques lignes.

#### YAPI.RegisterHub

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

#### YDisplay.FindDisplay

La fonction `YDisplay.FindDisplay` permet de retrouver un écran en fonction du numéro de série de son module hôte et de son nom de fonction. Mais vous pouvez tout aussi bien utiliser des noms logiques que vous auriez préalablement configurés. Imaginons un module Yocto-Display avec le numéros de série `YD128X32-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `display` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128X32-123456.display");
display = YDisplay.FindDisplay("YD128X32-123456.MaFonction");
display = YDisplay.FindDisplay("MonModule.display");
```

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



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

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

## isOnline

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

## get\_displayLayer

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

## Un exemple réel

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

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

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

namespace ConsoleApplication1
{
    class Program
    {
        static void usage()
        {
            string execname = System.AppDomain.CurrentDomain.FriendlyName;
            Console.WriteLine(execname + " <serial_number> ");
            Console.WriteLine(execname + " <logical_name>");
            Console.WriteLine(execname + " any ");
            System.Threading.Thread.Sleep(2500);
            Environment.Exit(0);
        }

        static void Main(string[] args)
        {
            string errmsg = "";
            string target;
            YDisplay disp;
            YDisplayLayer l0, l1;
            int h, w, y, x, vx, vy;
            if (args.Length < 1) usage();

            target = args[0].ToUpper();

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

            // find the display according to command line parameters
            if (target == "ANY") {
                disp = YDisplay.FirstDisplay();
                if (disp == null) {
                    Console.WriteLine("No module connected (check USB cable) ");
                    Environment.Exit(0);
                }
            } else disp = YDisplay.FindDisplay(target + ".display");

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

```

    }

    //clean up
    disp.resetAll();

    // retrieve the display size
    w = disp.get_displayWidth();
    h = disp.get_displayHeight();

    // reteive the first layer
    l0 = disp.get_displayLayer(0);

    // display a text in the middle of the screen
    l0.drawText(w / 2, h / 2, YDisplayLayer.ALIGN.CENTER, "Hello world!");

    // visualize each corner
    l0.moveTo(0, 5);
    l0.lineTo(0, 0);
    l0.lineTo(5, 0);
    l0.moveTo(0, h - 6);
    l0.lineTo(0, h - 1);
    l0.lineTo(5, h - 1);
    l0.moveTo(w - 1, h - 6);
    l0.lineTo(w - 1, h - 1);
    l0.lineTo(w - 6, h - 1);
    l0.moveTo(w - 1, 5);
    l0.lineTo(w - 1, 0);
    l0.lineTo(w - 6, 0);

    // draw a circle in the top left corner of layer 1
    l1 = disp.get_displayLayer(1);
    l1.clear();
    l1.drawCircle(h / 8, h / 8, h / 8);

    // and animate the layer
    Console.WriteLine("Use Ctrl-C to stop");
    x = 0;
    y = 0;
    vx = 1;
    vy = 1;
    while (disp.isOnline()) {
        x += vx;
        y += vy;
        if ((x < 0) || (x > w - (h / 4))) vx = -vx;
        if ((y < 0) || (y > h - (h / 4))) vy = -vy;
        l1.setLayerPosition(x, y, 0);
        YAPI.Sleep(5, ref errormsg);
    }
    YAPI.FreeAPI();
}
}
}

```

## 11.4. Contrôle de la partie module

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

```

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

namespace ConsoleApplication1
{
    class Program
    {
        static void usage()
        {
            string execname = System.AppDomain.CurrentDomain.FriendlyName;
            Console.WriteLine("Usage:");
        }
    }
}

```

```

        Console.WriteLine(execname + " <serial or logical name> [ON/OFF]");
        System.Threading.Thread.Sleep(2500);
        Environment.Exit(0);
    }

    static void Main(string[] args)
    {
        YModule m;
        string 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)");
        }
        YAPI.FreeAPI();
    }
}

```

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

## Modifications des réglages du module

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

```

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

namespace ConsoleApplication1
{

```

```

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

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

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

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

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

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

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

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

```

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

## Enumeration des modules

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

```

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

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

```

```

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

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

```

## 11.5. Gestion des erreurs

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

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

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

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

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

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

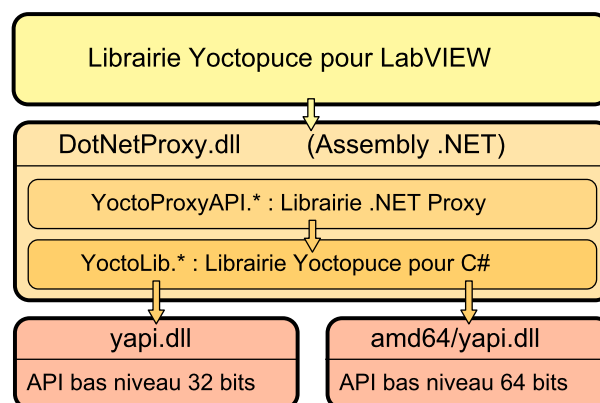
méthodes `errType()` et `errMessage()`. Ce sont les même informations qui auraient été associées à l'exception si elles avaient été actives.

## 12. Utilisation du Yocto-Display avec LabVIEW

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

### 12.1. Architecture

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



*Architecture de l'API Yoctopuce pour LabVIEW*

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

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

## 12.2. Compatibilité

### Firmwares

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

### LabVIEW pour Linux et MacOS

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

### LabVIEW NXG

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

### A propos de DotNetProxyLibrary.dll

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

## 12.3. Installation

Téléchargez la librairie pour LabVIEW depuis le site web de Yoctopuce<sup>1</sup>. Il s'agit d'un fichier ZIP dans lequel vous trouverez un répertoire par version de LabVIEW. Chacun de ses répertoires contient deux sous-répertoires. Le premier contient des exemples de programmation pour chaque produit Yoctopuce; le second, nommé *VIs*, contient tous les VI de l'API et les DLL nécessaires.

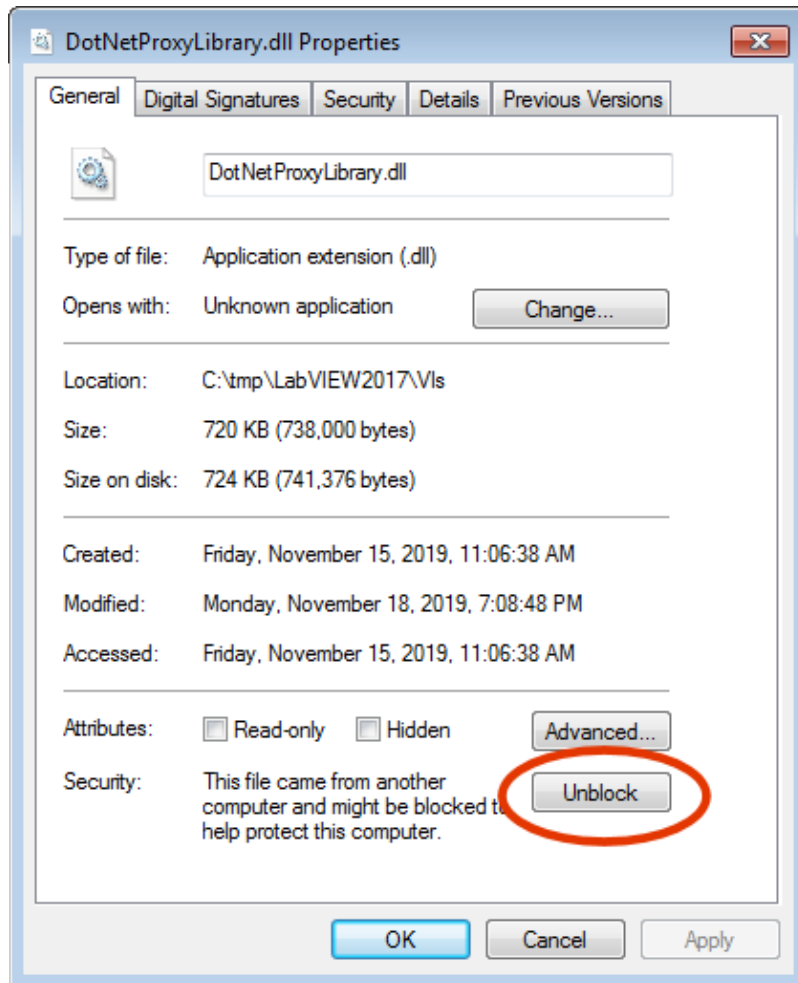
Suivant la configuration de Windows et la méthode utilisée pour la copier, la DLL *DotNetProxyLibrary.dll* peut se faire bloquer par Windows parce que ce dernier aura détecté qu'elle provient d'une autre machine. Un cas typique est la décompression de l'archive de la librairie avec l'explorateur de fichier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

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

---

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





Débloquer la DLL DotNetProxyLibrary.dll.

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

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

Veillez à choisir le bon répertoire en fonction de la version de LabVIEW que vous utilisez (32 bits vs 64 bits). Vous trouverez plus d'information à propos de ce fichier sur le site web de National Instrument<sup>2</sup>.

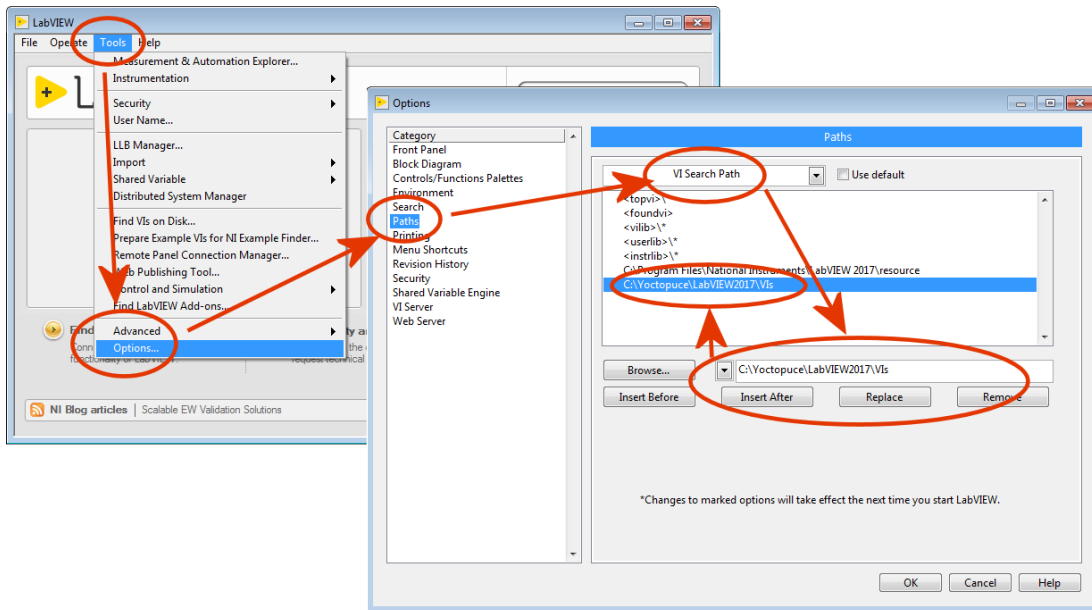
Pour installer l'API Yoctopuce pour LabVIEW vous avez plusieurs méthodes à votre disposition.

### Méthode 1 : Installation "à l'emporter"

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

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

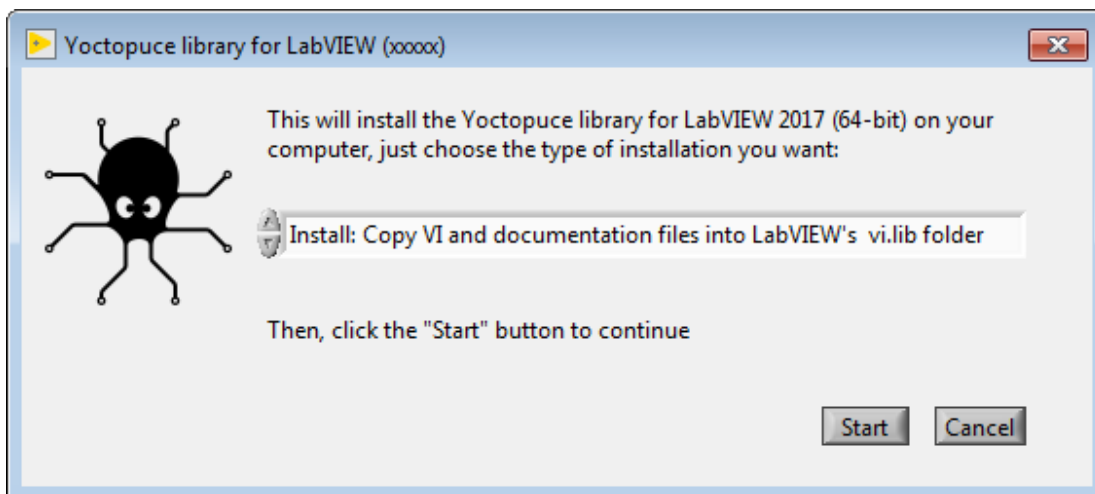
<sup>2</sup> <https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P8XnSAK>



Configuration du "VI Search Path"

## Méthode 2 : Installeur fourni avec la librairie

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



L'installeur fourni avec la librairie

Cet installeur offre trois options d'installation:

### Install: Keep VI and documentation files where they are.

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

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

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

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

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

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

### Uninstall Yoctopuce Library

Cette option supprime la Librairie Yoctopuce de votre installation LabVIEW:

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

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

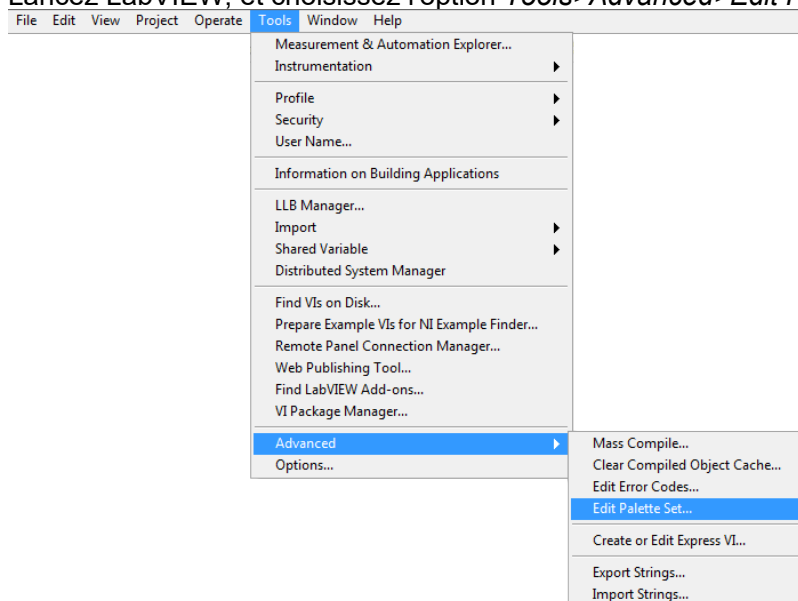
L'installateur reconnaît les répertoires contenant la librairie Yoctopuce en testant l'existence du fichier *YRegisterHub.vi*.

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

### Méthode 3 Installation manuelle dans la palette LabVIEW

Les étapes pour installer manuellement les VIs directement dans la Palette LabView sont un peu plus complexes, vous trouverez la procédure complète sur le site de National Instruments<sup>3</sup>, mais voici un résumé:

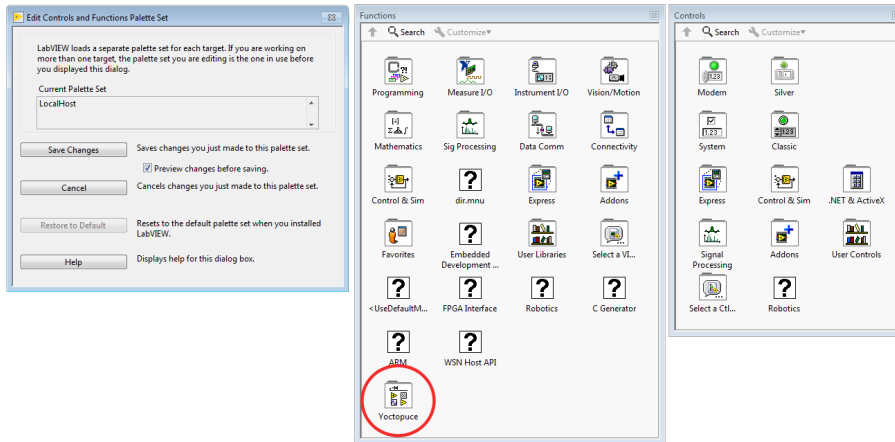
1. Créez un répertoire *Yoctopuce\API* dans le répertoire *C:\Program Files\National Instruments\LabVIEW xxxx\vi.lib*, et copiez tous les VIs et les DLL du répertoire *VIs* dedans.
2. Créez un répertoire *Yoctopuce* dans le répertoire *C:\Program Files\National Instruments\LabVIEW xxxx\menus\Categories*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



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

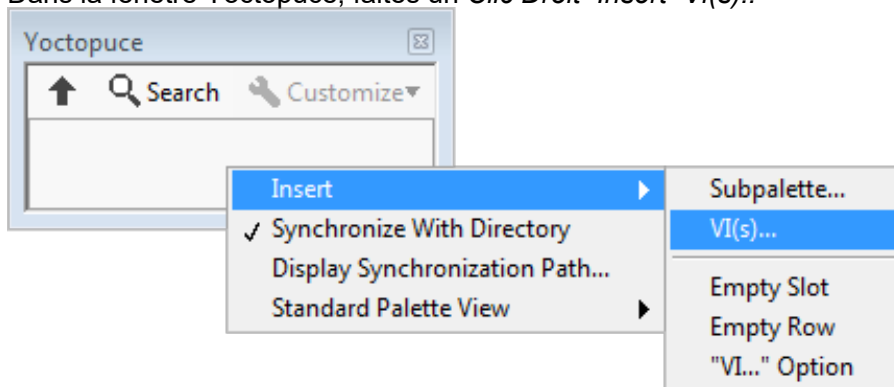
Trois fenêtres vont apparaître:

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

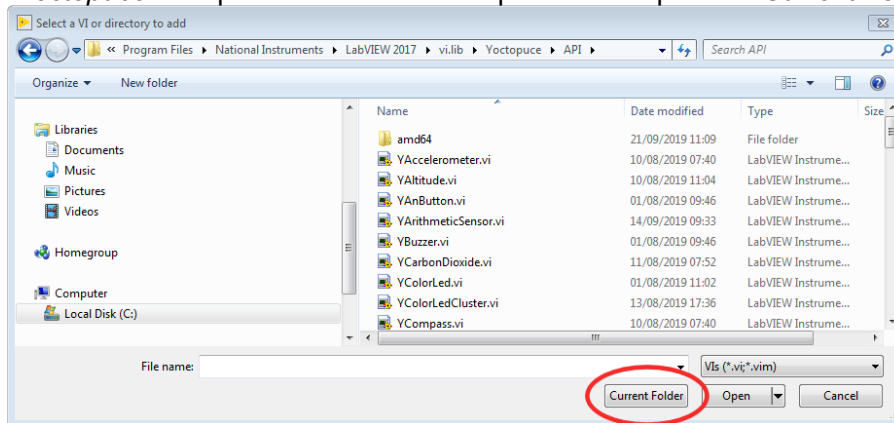


Dans la fenêtre *Function*, vous trouverez une icône *Yoctopuce*. Double-cliquez dessus, ce qui fera apparaître une fenêtre "Yoctopuce" vide.

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

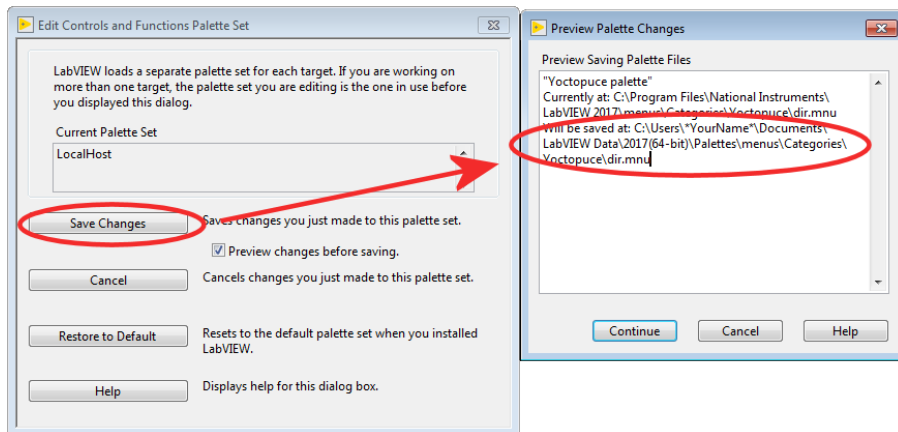


ce qui fera apparaître un sélecteur de fichier. Placer le sélecteur dans le répertoire *vi.lib \Yoctopuce\API* que vous avez créé au point 1 et cliquez sur *Current Folder*



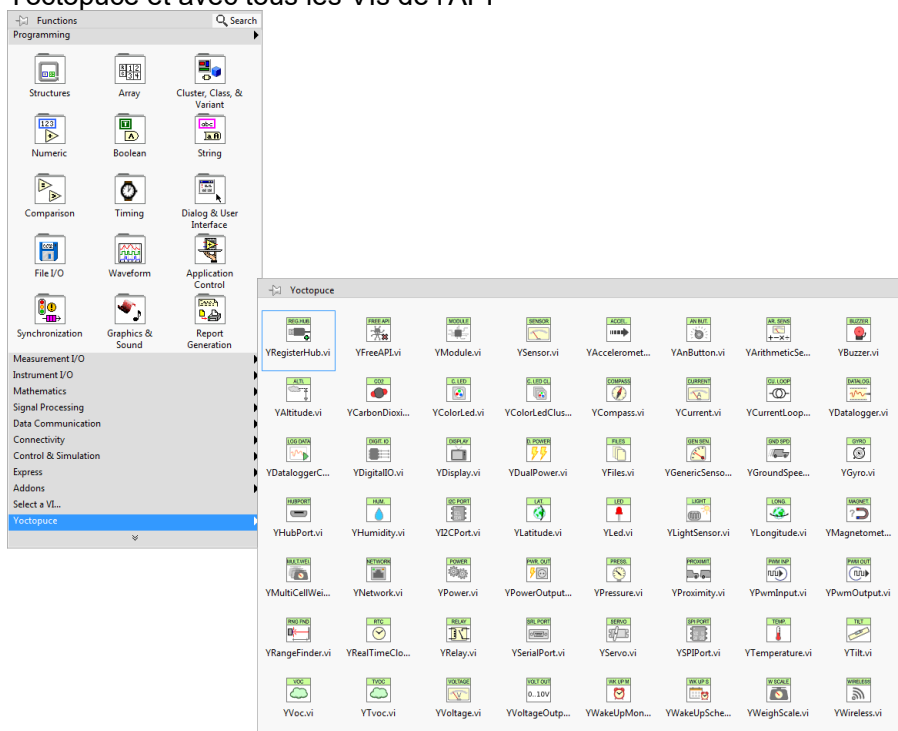
Tous les VIs Yoctopuce vont apparaître dans la fenêtre Yoctopuce. Par défaut, ils sont triés dans l'ordre alphabétique, mais vous pouvez les arranger comme bon vous semble en les glissant avec la souris. Pour que la palette soit bien utilisable, nous vous suggérons de réorganiser les icônes sur 8 colonnes.

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



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

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

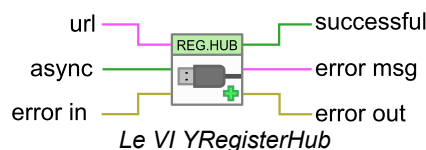


## 12.4. Présentation des VIs Yoctopuce

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

### YRegisterHub

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



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

- La chaîne de caractères "usb" pour indiquer que l'on souhaite travailler avec des modules locaux directement par USB
- Une adresse IP pour indiquer que l'on souhaite travailler avec des modules accessibles via une connexion réseau. Cette adresse IP peut être celle d'un YoctoHub<sup>4</sup> ou encore celle d'une machine sur laquelle tourne l'application VirtualHub<sup>5</sup>.

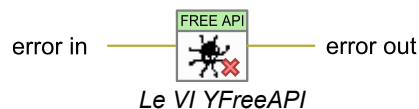
Dans le cas d'une adresse IP, le VI `YRegisterHub` va essayer de contacter cette adresse et générera une erreur s'il n'y arrive pas, à moins que le paramètre `async` ne soit mis à TRUE. Si `async` est mis à TRUE, aucune erreur ne sera générée, et les modules Yoctopuce correspondant à cette adresse IP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

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

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

## YFreeAPI

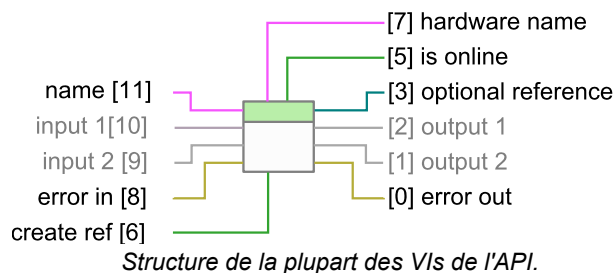
Le VI `YFreeAPI` permet de libérer les ressources allouées par l'API Yoctopuce.



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

## Structure des VI correspondant à une classe

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



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

<sup>4</sup> [www.yoctopuce.com/FR/products/category/extensions-and-networking](http://www.yoctopuce.com/FR/products/category/extensions-and-networking)

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

<sup>6</sup> voir section *Utilisation objets Proxy*

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

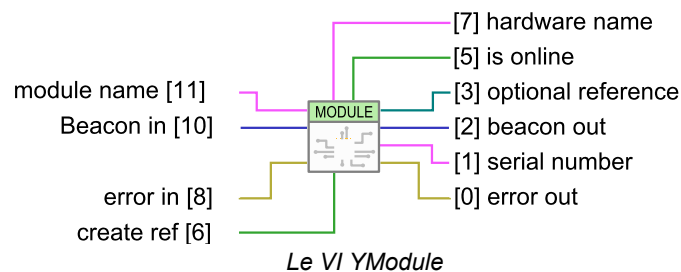
Vous trouverez la liste des fonctions disponibles sur votre Yocto-Display au chapitre *Programmation, concepts généraux*.

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

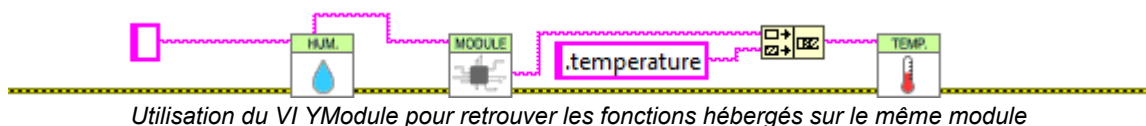
Si le paramètre *name* contient une chaîne vide, le VI ciblera la première fonction disponible du même type qu'il trouvera. Si aucune fonction n'est disponible, *is online* contiendra FALSE.

## Le VI YModule

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

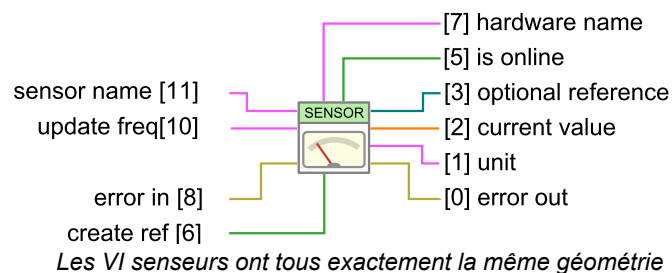


L'entrée *name* fonctionne de manière légèrement différente des autres VIs. S'il est appelé avec le paramètre *name* correspondant à un nom de fonction, le VI `YModule` trouvera la fonction *Module* du module hébergeant la fonction. Il est donc possible de trouver facilement le numéro de série du module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



## Les VI senseurs

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

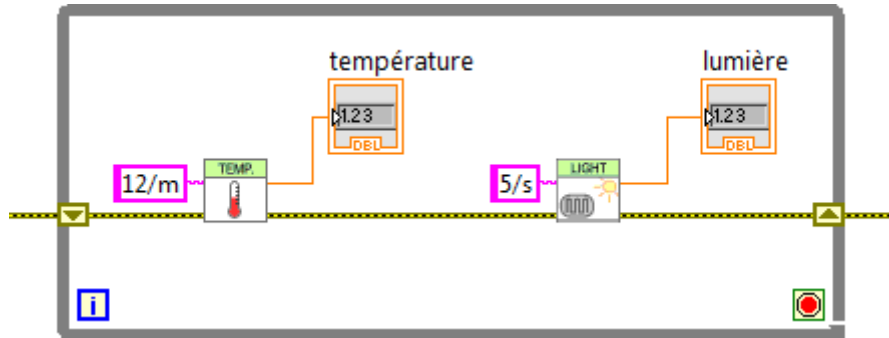


Le paramètre d'entrée *update freq* est une chaîne de caractères qui permet de configurer la façon dont la valeur de sortie est mis à jour:

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

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

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

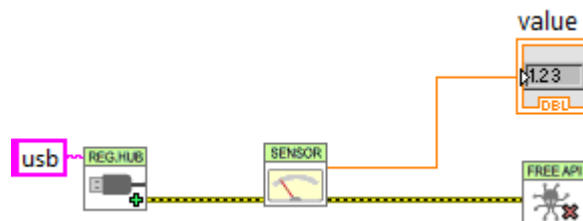


Changement de la fréquence de mise à jour du même module

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

## 12.5. Fonctionnement et utilisation des VIs

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



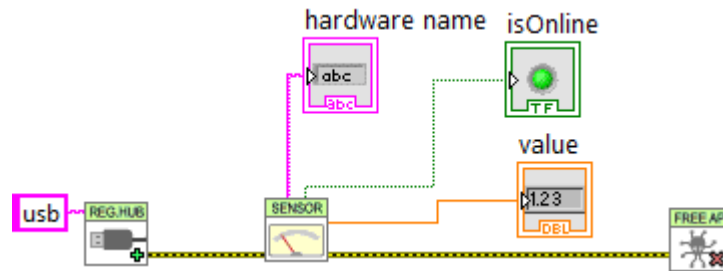
Exemple minimal d'utilisation de l'API Yoctopuce pour LabVIEW

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

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

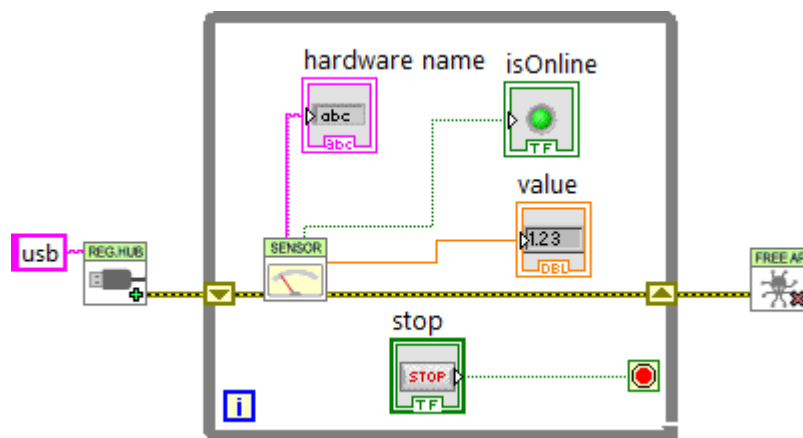
Cet exemple cherche automatiquement un capteur disponible, si un tel capteur est trouvé on pourra connaître son nom via la sortie `hardware name` et la sortie `isOnline` sera à `TRUE`. Si aucun capteur n'est disponible, le VI ne générera pas d'erreur mais émulerait un capteur fantôme qui sera "offline".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





Utilisation des sorties hardware name et isOnline

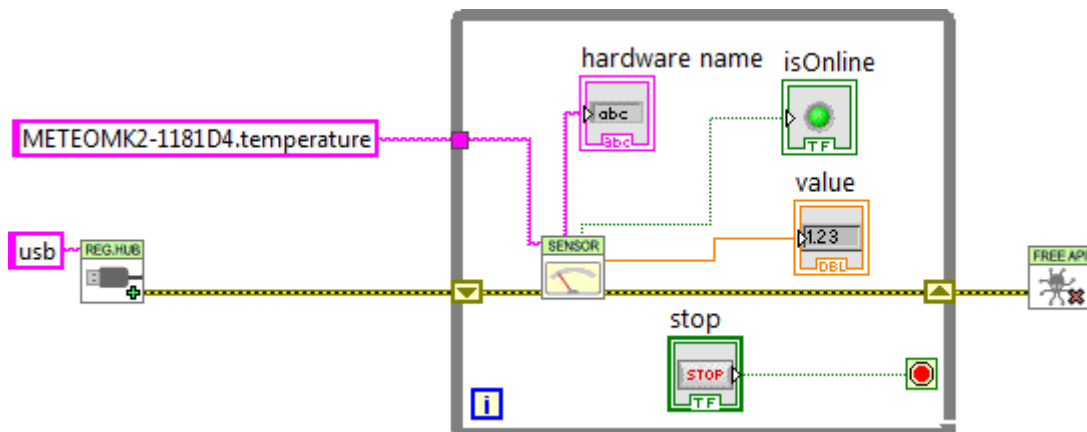
Les VIs de l'API Yoctopuce ne sont qu'une porte d'entrée sur la mécanique interne de la librairie Yoctopuce. Cette mécanique fonctionne indépendamment des VIs Yoctopuce. En effet, la plupart des communications avec les modules électroniques sont gérées automatiquement en arrière plan. C'est pourquoi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



Les VIs Yoctopuce peuvent être utilisés dans une boucle non temporisée

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

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

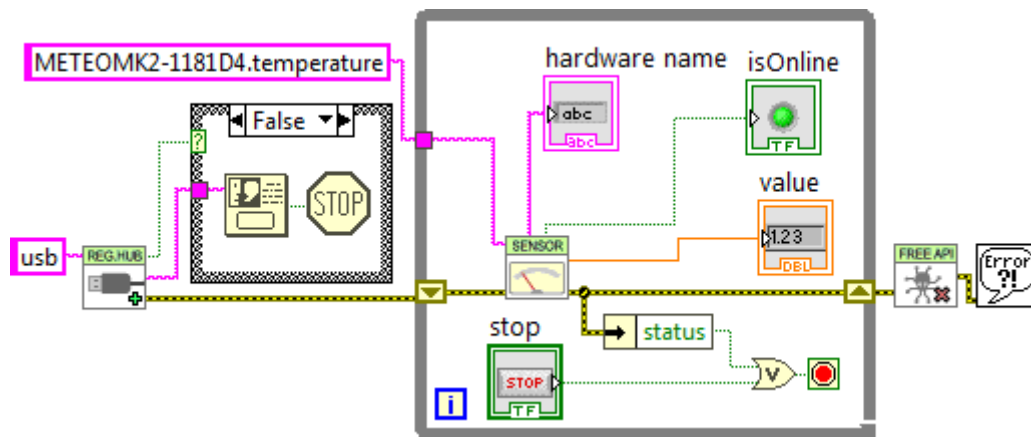


Utilisation de noms pour identifier les fonctions à utiliser

### Gestion des erreurs

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

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



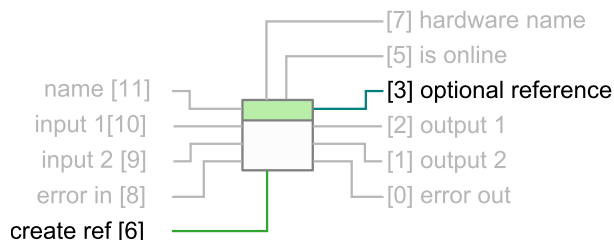
Gestion des erreurs

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

### 12.6. Utilisation des objets Proxy

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

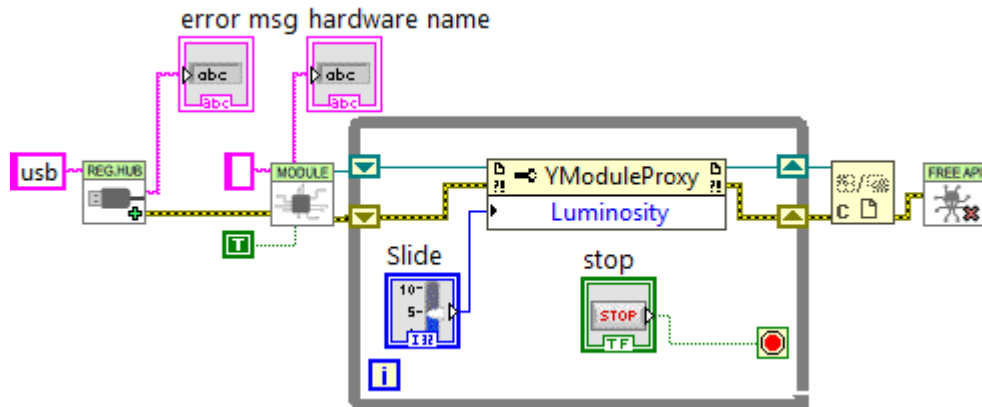
Chaque VI correspondant à une classe dispose de deux connecteurs *create ref* et *optional ref* qui permettent d'obtenir une référence sur l'objet Proxy de l'API .NET Proxy sur laquelle est construite la librairie LabVIEW.



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

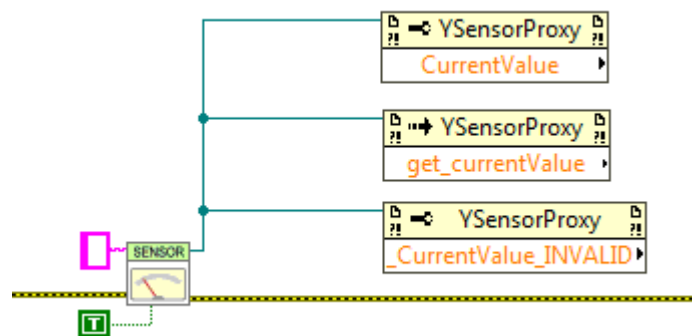
Pour obtenir cette référence, il suffit de mettre *optional ref* à TRUE. Attention, il est impératif de fermer toute référence créée de cette manière, sous peine de saturer rapidement la mémoire de l'ordinateur.

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



Contrôle de la luminosité des LEDs d'un module

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

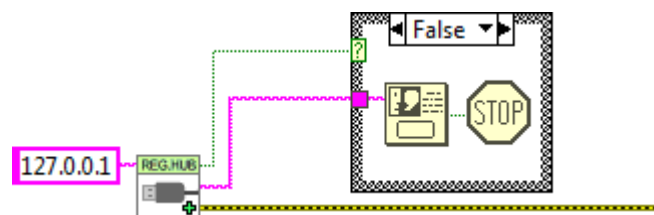


Noeuds Property et Invoke: Utilisation de propriétés, méthodes et constantes

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

## Utilisation en réseau

Sur une même machine, il ne peut y avoir qu'un seul processus qui accède aux modules Yoctopuce locaux directement par USB (url mis à "usb"). Par contre, plusieurs processus peuvent se connecter en parallèle à des YoctoHubs<sup>7</sup> ou à une machine sur laquelle tourne le logiciel *VirtualHub*<sup>8</sup>, y compris la machine locale. Si vous utilisez l'adresse réseau locale de votre machine (127.0.0.1) et qu'un VirtualHub tourne dessus, vous pourrez ainsi contourner la limitation qui empêche l'utilisation en parallèle de l'API native USB.

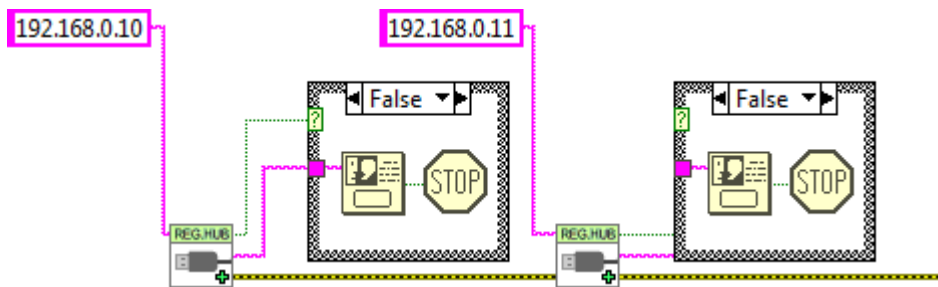


Utilisation en mode réseau

<sup>7</sup> [www.yoctopuce.com/FR/products/category/extensions-et-reseau](http://www.yoctopuce.com/FR/products/category/extensions-et-reseau)

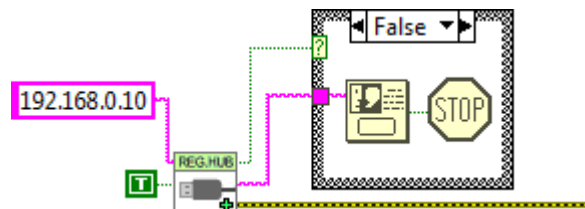
<sup>8</sup> [www.yoctopuce.com/FR/virtualhub.php](http://www.yoctopuce.com/FR/virtualhub.php)

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



Les connexions réseau multiples sont possibles

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



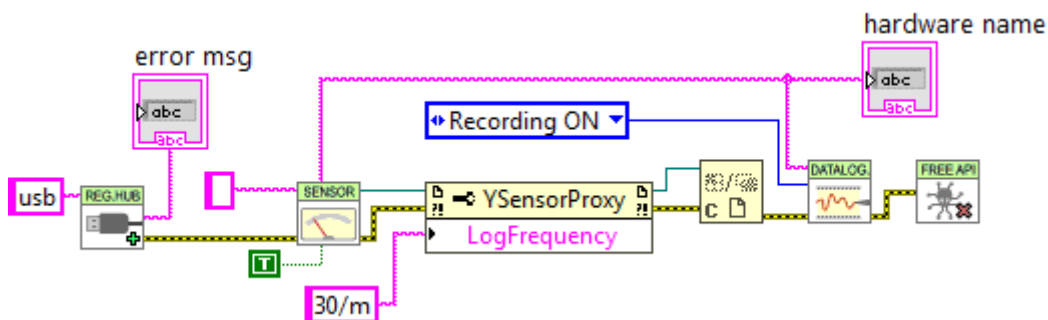
Connexion asynchrone

## 12.7. Gestion du datalogger

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

### Enregistrement

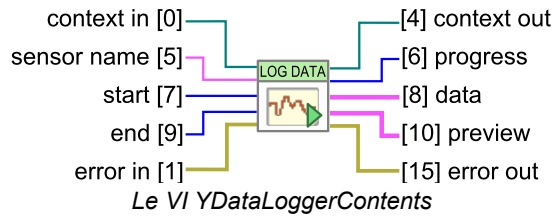
Pour ce faire, il faut configurer la fréquence d'enregistrement en utilisant la propriété "LogFrequency" que l'on atteint avec une référence sur l'objet Proxy du senseur utilisé, puis il faut mettre en marche l'enregistreur grâce au VI YDataLogger. Noter qu'à la manière du VI YModule, le VI YDataLogger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



Enclenchement de l'enregistrement de données dans le datalogger

### Lecture

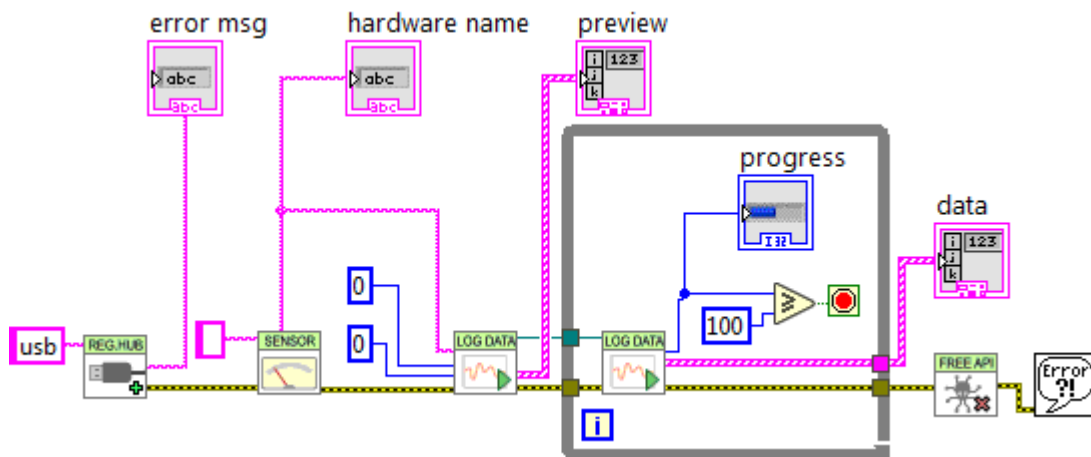
La récupération des données de l'enregistreur se fait à l'aide du VI YDataLoggerContents.



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

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

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



Récupération du contenu de l'enregistreur de données

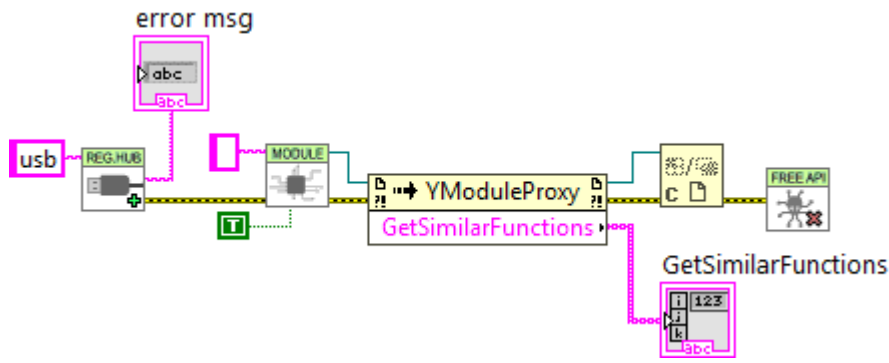
Les résultats et le résumé sont rendus sous la forme d'un tableau de structures qui contiennent les champs suivants:

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

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

## 12.8. Énumération de fonctions

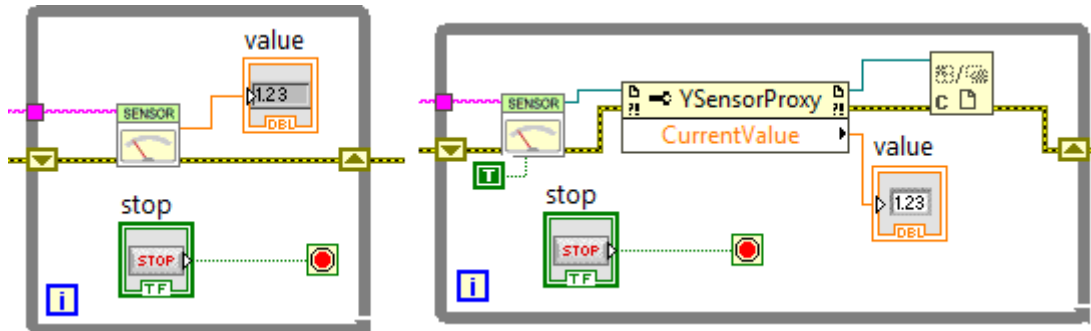
Chaque VI correspondant à un objet de l'API *.NET Proxy* permet de faire une énumération de toutes les fonctions de la même classe via la méthode *getSimilarfunctions()* de l'objet *Proxy* correspondant. Ainsi il est ainsi aisé de faire un inventaire de tous les modules connectés, de tous les capteurs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



Récupération de la liste de tous les modules connectés

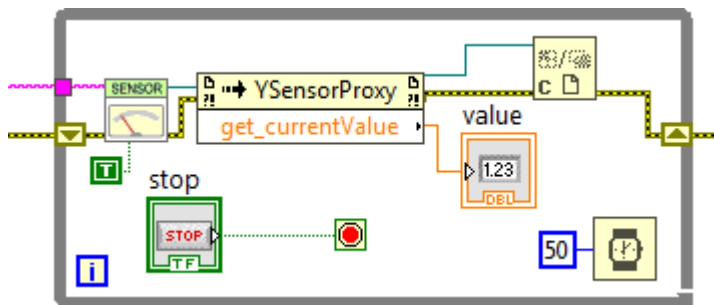
## 12.9. Un mot sur les performances

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



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

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



Cette boucle, qui utilise une méthode, doit être ralentie

## 12.10. Un exemple complet de programme LabVIEW

**UNABLE TO INCLUDE**  
<http://172.17.17.77/tu/projects/yoctodisplay-128x32/public/doc-labview-example-FR.html>

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

## 12.11. Différences avec les autres API Yoctopuce

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

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

### YFreeAPI

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

### Propriétés

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

### Callback vs Propriétés

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





## 13. Utilisation du Yocto-Display en Java

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

### 13.1. Préparation

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

- La librairie de programmation pour Java<sup>1</sup>
- VirtualHub<sup>2</sup> pour Windows, macOS ou Linux selon l'OS que vous utilisez

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

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

### 13.2. Contrôle de la fonction Display

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

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

// On récupère l'objet permettant d'interagir avec le module
display = YDisplay.FindDisplay("YD128X32-123456.display");

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

<sup>1</sup> [www.yoctopuce.com/FR/libraries.php](http://www.yoctopuce.com/FR/libraries.php)

<sup>2</sup> [www.yoctopuce.com/FR/virtualhub.php](http://www.yoctopuce.com/FR/virtualhub.php)

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

[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

## YAPI.RegisterHub

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

## YDisplay.FindDisplay

La fonction `YDisplay.FindDisplay` permet de retrouver un écran en fonction du numéro de série de son module hôte et de son nom de fonction. Mais vous pouvez tout aussi bien utiliser des noms logiques que vous auriez préalablement configurés. Imaginons un module Yocto-Display avec le numéros de série `YD128X32-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `display` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128X32-123456.display")
display = YDisplay.FindDisplay("YD128X32-123456.MaFonction")
display = YDisplay.FindDisplay("MonModule.display")
display = YDisplay.FindDisplay("MonModule.MaFonction")
display = YDisplay.FindDisplay("MaFonction")
```

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

## isOnline

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

## get\_displayLayer

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

## Un exemple réel

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

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

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

public class Demo {

    private static void disp(YDisplay display, String text, YDisplayLayer.ALIGN al) throws
YAPI_Exception
    {
        YDisplayLayer layer0 = display.get_displayLayer(0);
        int l = (int) display.get_displayWidth();
        int h = (int) display.get_displayHeight();
        int mx = l / 2;
        int my = h / 2;
        layer0.clear();
        layer0.moveTo(mx, 0);
        layer0.lineTo(mx, h);
    }
}
```

```

        layer0.moveTo(0, my);
        layer0.lineTo(1, my);
        layer0.drawText(mx, my, al, text);
    }

    public static void main(String[] args)
    {

        YDisplay disp;
        YDisplayLayer l0, l1;
        int h, w, y, x, vx, vy;

        // API init
        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 == 0) {
            disp = YDisplay.FirstDisplay();
            if (disp == null) {
                System.out.println("No module connected (check USB cable)");
                System.exit(1);
            }
        } else {
            disp = YDisplay.FindDisplay(args[0] + ".display");
        }

        try {
            //clean up
            disp.resetAll();

            // retrieve the display size
            w = disp.get_displayWidth();
            h = disp.get_displayHeight();

            // retrieve the first layer
            l0 = disp.get_displayLayer(0);

            // display a text in the middle of the screen
            l0.drawText(w / 2, h / 2, YDisplayLayer.ALIGN.CENTER, "Hello world!");

            // visualize each corner
            l0.moveTo(0, 5);
            l0.lineTo(0, 0);
            l0.lineTo(5, 0);
            l0.moveTo(0, h - 6);
            l0.lineTo(0, h - 1);
            l0.lineTo(5, h - 1);
            l0.moveTo(w - 1, h - 6);
            l0.lineTo(w - 1, h - 1);
            l0.lineTo(w - 6, h - 1);
            l0.moveTo(w - 1, 5);
            l0.lineTo(w - 1, 0);
            l0.lineTo(w - 6, 0);

            // draw a circle in the top left corner of layer 1
            l1 = disp.get_displayLayer(1);
            l1.clear();
            l1.drawCircle(h / 8, h / 8, h / 8);

            // and animate the layer
            System.out.println("Use Ctrl-C to stop");
            x = 0;
            y = 0;
            vx = 1;
            vy = 1;
            while (true) {
                x += vx;
                y += vy;
                if ((x < 0) || (x > w - (h / 4))) {
                    vx = -vx;
                }
            }
        }
    }

```

```

        if ((y < 0) || (y > h - (h / 4))) {
            vy = -vy;
        }
        ll.setLayerPosition(x, y, 0);
        YAPI.Sleep(5);
    }
} catch (YAPI_Exception ex) {
    System.out.println("Exception durring execution (" + ex.getLocalizedMessage() +
    ")");
    YAPI.FreeAPI();
    System.exit(1);
}
}
}

```

### 13.3. Contrôle de la partie module

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

```

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

public class Demo {

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

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

        try {
            if (args.length > 1) {
                if (args[1].equalsIgnoreCase("ON")) {
                    module.setBeacon(YModule.BEACON_ON);
                } else {
                    module.setBeacon(YModule.BEACON_OFF);
                }
            }
            System.out.println("serial:      " + module.get_serialNumber());
            System.out.println("logical name: " + module.get_logicalName());
            System.out.println("luminosity:  " + module.get_luminosity());
            if (module.get_beacon() == YModule.BEACON_ON) {
                System.out.println("beacon:      ON");
            } else {
                System.out.println("beacon:      OFF");
            }
            System.out.println("upTime:      " + module.get_upTime() / 1000 + " sec");
            System.out.println("USB current: " + module.get_usbCurrent() + " mA");
            System.out.println("logs:\n" + module.get_lastLogs());
        }
    }
}

```

```

    } catch (YAPI_Exception ex) {
        System.out.println(args[1] + " not connected (check identification and USB
cable)");
    }
    YAPI.FreeAPI();
}
}

```

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

## Modifications des réglages du module

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

```

import com.yoctopuce.YoctoAPI.*;

public class Demo {

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

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

        YModule m;
        String newname;

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

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

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

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

        YAPI.FreeAPI();
    }
}

```

```
}

```

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

## Enumeration des modules

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

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

public class Demo {

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

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

```

## 13.4. Gestion des erreurs

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

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

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







## 14. Utilisation du Yocto-Display avec Android

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

### 14.1. Accès Natif et VirtualHub

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

### 14.2. Préparation

Connectez-vous sur le site de Yoctopuce et téléchargez la librairie de programmation pour Java pour Android<sup>1</sup>. La librairie est disponible en fichiers sources, mais elle aussi disponible sous la forme d'un fichier jar. Branchez vos modules, décompressez les fichiers de la librairie dans le répertoire de votre choix. Et configurez votre environnement de programmation Android pour qu'il puisse les trouver.

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

### 14.3. Compatibilité

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

---

<sup>1</sup> [www.yoctopuce.com/FR/libraries.php](http://www.yoctopuce.com/FR/libraries.php)

## Version d'Android

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

## Support USB *host*

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

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

## 14.4. Activer le port USB sous Android

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

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

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

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

```
...
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    try {
        // Pass the application Context to the Yoctopuce Library
        YAPI.EnableUSBHost(this);
    }
}
```

```

    } catch (YAPI_Exception e) {
        Log.e("Yocto",e.getLocalizedMessage());
    }
}
...

```

## Lancement automatique

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

```

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

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

```

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

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <usb-device vendor-id="9440" product-id="12" />
    <usb-device vendor-id="9440" product-id="13" />
</resources>

```

## 14.5. Contrôle de la fonction Display

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

```

[... ]
// On active la détection des modules sur USB
YAPI.EnableUSBHost (this);
YAPI.RegisterHub("usb");
[... ]
// On récupère l'objet permettant de communiquer avec le module
display = YDisplay.FindDisplay("YD128X32-123456.display");

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

```

[...]

Voyons maintenant en détail ce que font ces quelques lignes.

### YAPI.EnableUSBHost

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

### YAPI.RegisterHub

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

### YDisplay.FindDisplay

La fonction `YDisplay.FindDisplay` permet de retrouver un écran en fonction du numéro de série de son module hôte et de son nom de fonction. Mais vous pouvez tout aussi bien utiliser des noms logiques que vous auriez préalablement configurés. Imaginons un module Yocto-Display avec le numéro de série `YD128X32-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `display` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128X32-123456.display")
display = YDisplay.FindDisplay("YD128X32-123456.MaFonction")
display = YDisplay.FindDisplay("MonModule.display")
display = YDisplay.FindDisplay("MonModule.MaFonction")
display = YDisplay.FindDisplay("MaFonction")
```

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

### isOnline

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

### get\_displayLayer

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

### Un exemple réel

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

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

```
package com.yoctopuce.doc_examples;

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

```

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

public class GettingStarted_Yocto_Display extends Activity implements
OnItemSelectedListener
{

    private YDisplay display = null;
    private ArrayAdapter<String> aa;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.gettingstarted_yocto_display);
        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();
        aa.clear();
        try {
            YAPI.EnableUSBHost(this);
            YAPI.RegisterHub("usb");
            YDisplay d = YDisplay.FirstDisplay();
            while (d != null) {
                String hwid = d.get_hardwareId();
                aa.add(hwid);
                d = d.nextDisplay();
            }
        } catch (YAPI_Exception e) {
            e.printStackTrace();
        }
        aa.notifyDataSetChanged();
    }

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

    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int pos, long id)
    {
        String hwid = parent.getItemAtPosition(pos).toString();
        display = YDisplay.FindDisplay(hwid);
        updateDisplay(null);
    }

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

    public void updateDisplay(View view)
    {
        if (display == null)
            return;

        EditText message = (EditText) findViewById(R.id.editText1);
        // clean up
        try {
            display.resetAll();

            // retrieve the display size
            int w = display.get_displayWidth();
            int h = display.get_displayHeight();

```

```

// retrieve the first layer
YDisplayLayer l0 = display.get_displayLayer(0);

// display a text in the middle of the screen
l0.drawText(w / 2, h / 2, YDisplayLayer.ALIGN.CENTER, message.getText(
).toString());

// visualize each corner
l0.moveTo(0, 5);
l0.lineTo(0, 0);
l0.lineTo(5, 0);
l0.moveTo(0, h - 6);
l0.lineTo(0, h - 1);
l0.lineTo(5, h - 1);
l0.moveTo(w - 1, h - 6);
l0.lineTo(w - 1, h - 1);
l0.lineTo(w - 6, h - 1);
l0.moveTo(w - 1, 5);
l0.lineTo(w - 1, 0);
l0.lineTo(w - 6, 0);
} catch (YAPI_Exception e) {
    e.printStackTrace();
}
}
}

```

## 14.6. Contrôle de la partie module

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

```

package com.yoctopuce.doc_examples;

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

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

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

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

    @Override
    protected void onStart()
    {

```

```

super.onStart();

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

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

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

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

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

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

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

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

    try {
        if (on) {

```

```

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

```

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

## Modifications des réglages du module

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

```

package com.yoctopuce.doc_examples;

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

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

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

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

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

        try {
            aa.clear();
            YAPI.EnableUSBHost(this);
            YAPI.RegisterHub("usb");
            YModule r = YModule.FirstModule();
            while (r != null) {
                String hwid = r.get_hardwareId();
            }
        }
    }
}

```



```

        aa.add(hwid);
        r = r.nextModule();
    }
} catch (YAPI_Exception e) {
    e.printStackTrace();
}
// refresh Spinner with detected relay
aa.notifyDataSetChanged();
}

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

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

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

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

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

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

```

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

## Enumeration des modules

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

```
package com.yoctopuce.doc_examples;

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

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

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

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

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

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

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

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

## 14.7. Gestion des erreurs

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

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

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



## 15. Utilisation du Yocto-Display en TypeScript

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

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

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

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

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

## 15.1. Utiliser la librairie Yoctopuce pour TypeScript

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

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

```
npm install -g typescript
```

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

- La librairie de programmation pour TypeScript<sup>1</sup>
- Le programme VirtualHub<sup>2</sup> pour Windows, macOS ou Linux selon l'OS que vous utilisez. En effet, TypeScript et JavaScript font partie de ces langages qui ne vous permettront pas d'accéder directement aux périphériques USB. C'est pourquoi si vous désirez travailler avec des modules branchés par USB, vous devrez faire tourner la passerelle de Yoctopuce appelée VirtualHub sur la machine à laquelle sont branchés les modules. Vous n'avez en revanche pas besoin d'installer de driver.

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

```
npm install
```

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

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

## 15.2. Petit rappel sur les fonctions asynchrones en JavaScript

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

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

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

<sup>1</sup> [www.yoctopuce.com/FR/libraries.php](http://www.yoctopuce.com/FR/libraries.php)

<sup>2</sup> [www.yoctopuce.com/FR/virtualhub.php](http://www.yoctopuce.com/FR/virtualhub.php)

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

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

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

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

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

### 15.3. Contrôle de la fonction Display

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

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

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

// On récupère l'objet permettant d'interagir avec le module
let display: YDisplay = YDisplay.FindDisplay("YD128X32-123456.display");

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

Voyons maintenant en détail ce que font ces quelques lignes.

#### Import de `yocto_api` et `yocto_display`

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

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

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

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

## YAPI.RegisterHub

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

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

## YDisplay.FindDisplay

La méthode `FindDisplay` permet de retrouver un écran en fonction du numéro de série de son module hôte et de son nom de fonction. Mais vous pouvez tout aussi bien utiliser des noms logiques que vous auriez préalablement configurés. Imaginons un module Yocto-Display avec le numéros de série `YD128X32-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `display` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128X32-123456.display")
display = YDisplay.FindDisplay("YD128X32-123456.MaFonction")
display = YDisplay.FindDisplay("MonModule.display")
display = YDisplay.FindDisplay("MonModule.MaFonction")
display = YDisplay.FindDisplay("MaFonction")
```

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

## isOnline

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

## get\_displayLayer

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

## Un exemple concret, en Node.js

Ouvrez une fenêtre de commande (un terminal, un shell...) et allez dans le répertoire **example\_nodejs/Doc-GettingStarted-Yocto-Display** de la librairie Yoctopuce pour TypeScript. Vous y trouverez un fichier nommé `demo.ts` avec le code d'exemple ci-dessous, qui reprend les fonctions expliquées précédemment, mais cette fois utilisées avec le décorum nécessaire à en faire un petit programme d'exemple concret.

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

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



```

let disp: YDisplay;
let l1: YDisplayLayer;
let h: number;
let w: number;
let y: number;
let x: number;
let vx: number;
let vy: number;

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

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

    // Select specified device, or use first available one
    let serial: string = process.argv[process.argv.length-1];
    if (serial[8] != '-') {
        let anydisplay = YDisplay.FirstDisplay();
        if (anydisplay) {
            let module: YModule = await anydisplay.get_module();
            serial = await module.get_serialNumber();
        } else {
            console.log('No matching device connected, check cable !');
            await YAPI.FreeAPI();
            return;
        }
    }
    console.log('Using device ' + serial);
    disp = YDisplay.FindDisplay(serial + ".display");

    //clean up
    await disp.resetAll();

    // retrieve the display size
    w = await disp.get_displayWidth();
    h = await disp.get_displayHeight();

    // retrieve the first layer
    let l0 = await disp.get_displayLayer(0);
    // display a text in the middle of the screen
    await l0.drawText(w / 2, h / 2, YDisplayLayer.ALIGN_CENTER, "Hello world!");

    // visualize each corner
    await l0.moveTo(0, 5);
    await l0.lineTo(0, 0);
    await l0.lineTo(5, 0);
    await l0.moveTo(0, h - 6);
    await l0.lineTo(0, h - 1);
    await l0.lineTo(5, h - 1);
    await l0.moveTo(w - 1, h - 6);
    await l0.lineTo(w - 1, h - 1);
    await l0.lineTo(w - 6, h - 1);
    await l0.moveTo(w - 1, 5);
    await l0.lineTo(w - 1, 0);
    await l0.lineTo(w - 6, 0);

    // draw a circle in the top left corner of layer 1
    l1 = await disp.get_displayLayer(1);
    await l1.clear();
    await l1.drawCircle(h / 8, h / 8, h / 8);

    // and animate the layer
    console.log("Use Ctrl-C to stop");
    x = 0;
    y = 0;
    vx = 1;
    vy = 1;
    refresh();
}

async function refresh(): Promise<void>

```

```

{
  if (await disp.isOnline()) {
    x += vx;
    y += vy;
    if ((x < 0) || (x > w - (h / 4))) vx = -vx;
    if ((y < 0) || (y > h - (h / 4))) vy = -vy;
    await ll.setLayerPosition(x, y, 0);
  } else {
    console.log('Module not connected');
  }
  setTimeout(refresh, 5);
}

startDemo();

```

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

```
npm install
```

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

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

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

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

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

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

```
npm install yoctolib-cjs
```

## Le même exemple, mais dans un navigateur

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

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

```
npm run app-server
```

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

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

## 15.4. Contrôle de la partie module

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

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

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

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

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

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

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

### Modifications des réglages du module

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

```

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

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

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

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

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

```

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

## Énumération des modules

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

```

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

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

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

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

        let module = YModule.FirstModule();
    }
}

```

```

while(module) {
    let line: string = await module.get_serialNumber();
    line += '(' + (await module.get_productName()) + ')';
    console.log(line);
    module = module.nextModule();
}
setTimeout(refresh, 500);
} catch(e) {
    console.log(e);
}
}

startDemo();

```

## 15.5. Gestion des erreurs

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

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

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

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

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

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



## 16. Utilisation du Yocto-Display en JavaScript / EcmaScript

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

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

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

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

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

Nous ne recommandons plus l'utilisation de `jspm` dès lors que `async / await` sont standardisés.

## 16.1. Fonctions bloquantes et fonctions asynchrones en JavaScript

JavaScript a été conçu pour éviter toute situation de *concurrency* durant l'exécution. Il n'y a jamais qu'un seul *thread* en JavaScript. Cela signifie que si un programme effectue une attente active durant une communication réseau, par exemple pour lire un capteur, le programme entier se trouve bloqué. Dans un navigateur, cela peut se traduire par un blocage complet de l'interface utilisateur. C'est pourquoi l'utilisation de fonctions d'entrée/sortie bloquantes en JavaScript est sévèrement découragée de nos jours, et les API bloquantes se font toutes déclarer *deprecated*.

Plutôt que d'utiliser des *threads* parallèles, JavaScript utilise les opérations asynchrones pour gérer les attentes dans les entrées/sorties: lorsqu'une fonction potentiellement bloquante doit être appelée, l'opération est uniquement déclenchée mais le flot d'exécution est immédiatement terminé. Le moteur JavaScript est alors libre pour exécuter d'autres tâches, comme la gestion de l'interface utilisateur par exemple. Lorsque l'opération bloquante se termine finalement, le système relance le code en appelant une fonction de callback, en passant en paramètre le résultat de l'opération, pour permettre de continuer la tâche originale.

Lorsqu'on les utilise avec des simples fonctions de callback, comme c'est fait quasi systématiquement dans les bibliothèques Node.js, les opérations asynchrones ont la fâcheuse tendance de rendre le code illisible puisqu'elles découpent systématiquement le flot du code en petites fonctions de callback déconnectées les unes des autres. Heureusement, de nouvelles idées sont apparues récemment pour améliorer la situation. En particulier, l'utilisation d'objets *Promise* pour travailler avec les opérations asynchrones aide beaucoup. N'importe quelle fonction qui effectue une opération potentiellement longue peut retourner une *promesse* de se terminer, et cet objet *Promise* peut être utilisé par l'appelant pour chaîner d'autres opérations en un flot d'exécution. La classe *Promise* fait partie du standard EcmaScript 2015.

Les objets *Promise* sont utiles, mais ce qui les rend vraiment pratique est la nouvelle syntaxe `async / await` pour la gestion des appels asynchrones:

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

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

Nous avons donc décidé d'écrire cette nouvelle bibliothèque EcmaScript en utilisant les objets *Promise* et des fonctions *async*, pour vous permettre d'utiliser la notation *await* si pratique. Et pour ne pas devoir vous poser la question pour chaque méthode de savoir si elle est asynchrone ou pas, la convention est la suivante: **toutes les méthodes publiques** de la bibliothèque EcmaScript **sont *async***, c'est-à-dire qu'elles retournent un objet *Promise*, **sauf**:

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



## 16.2. Utiliser la librairie Yoctopuce pour JavaScript / EcmaScript 2017

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

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

- La librairie de programmation pour Javascript / EcmaScript 2017<sup>1</sup>
- VirtualHub<sup>2</sup> pour Windows, macOS ou Linux selon l'OS que vous utilisez

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

### Utiliser la librairie Yoctopuce officielle pour node.js

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

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

```
npm install
```

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

```
node demo.js
```

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

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

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

### Utiliser la librairie Yoctopuce dans un navigateur (HTML)

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

### Utiliser la librairie Yoctopuce avec des anciennes version de JavaScript

Si vous avez besoin d'utiliser cette librairie avec des moteurs JavaScript plus anciens, vous pouvez utiliser Babel<sup>3</sup> pour transpiler votre code et la librairie dans une version antérieure du langage. Pour installer Babel avec les réglages usuels, tapez:

<sup>1</sup> [www.yoctopuce.com/FR/libraries.php](http://www.yoctopuce.com/FR/libraries.php)

<sup>2</sup> [www.yoctopuce.com/FR/virtualhub.php](http://www.yoctopuce.com/FR/virtualhub.php)

<sup>3</sup> <http://babeljs.io>

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

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

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

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

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

## Compatibilité avec l'ancienne librairie JavaScript

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

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

par

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

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

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

par

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

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

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

```

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

on peut utiliser:

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

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

Ce dernier appel asynchrone peut aussi être formulé comme:

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

## 16.3. Contrôle de la fonction Display

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

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

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

// On récupère l'objet permettant d'interagir avec le module
let display = YDisplay.FindDisplay("YD128X32-123456.display");

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

Voyons maintenant en détail ce que font ces quelques lignes.

### Require de yocto\_api et yocto\_display

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

### YAPI.RegisterHub

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

### YDisplay.FindDisplay

La méthode `FindDisplay` permet de retrouver un écran en fonction du numéro de série de son module hôte et de son nom de fonction. Mais vous pouvez tout aussi bien utiliser des noms logiques

que vous auriez préalablement configurés. Imaginons un module Yocto-Display avec le numéros de série `YD128X32-123456` que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction `display` "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128X32-123456.display")
display = YDisplay.FindDisplay("YD128X32-123456.MaFonction")
display = YDisplay.FindDisplay("MonModule.display")
display = YDisplay.FindDisplay("MonModule.MaFonction")
display = YDisplay.FindDisplay("MaFonction")
```

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

## isOnline

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

## get\_displayLayer

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

## Un exemple concret, en Node.js

Ouvrez une fenêtre de commande (un terminal, un shell...) et allez dans le répertoire **example\_nodejs/Doc-GettingStarted-Yocto-Display** de la librairie Yoctopuce pour JavaScript / EcmaScript 2017. Vous y trouverez un fichier nommé `demo.js` avec le code d'exemple ci-dessous, qui reprend les fonctions expliquées précédemment, mais cette fois utilisées avec le décorum nécessaire à en faire un petit programme d'exemple concret.

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

```
"use strict";

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

let disp, ll;
let h, w, y, x, vx, vy;

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

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

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

```

disp = YDisplay.FindDisplay(serial + ".display");

//clean up
await disp.resetAll();

// retrieve the display size
w = await disp.get_displayWidth();
h = await disp.get_displayHeight();

// retrieve the first layer
let l0 = await disp.get_displayLayer(0);
// display a text in the middle of the screen
await l0.drawText(w / 2, h / 2, YDisplayLayer.ALIGN_CENTER, "Hello world!");

// visualize each corner
await l0.moveTo(0, 5);
await l0.lineTo(0, 0);
await l0.lineTo(5, 0);
await l0.moveTo(0, h - 6);
await l0.lineTo(0, h - 1);
await l0.lineTo(5, h - 1);
await l0.moveTo(w - 1, h - 6);
await l0.lineTo(w - 1, h - 1);
await l0.lineTo(w - 6, h - 1);
await l0.moveTo(w - 1, 5);
await l0.lineTo(w - 1, 0);
await l0.lineTo(w - 6, 0);

// draw a circle in the top left corner of layer 1
l1 = await disp.get_displayLayer(1);
await l1.clear();
await l1.drawCircle(h / 8, h / 8, h / 8);

// and animate the layer
console.log("Use Ctrl-C to stop");
x = 0;
y = 0;
vx = 1;
vy = 1;
refresh();
}

async function refresh() {
  if (await disp.isOnline()) {
    x += vx;
    y += vy;
    if ((x < 0) || (x > w - (h / 4))) vx = -vx;
    if ((y < 0) || (y > h - (h / 4))) vy = -vy;
    await l1.setLayerPosition(x, y, 0);
  } else {
    console.log('Module not connected');
  }
  setTimeout(refresh, 5);
}

startDemo();

```

Comme décrit au début de ce chapitre, vous devez avoir installé Node.js v7.6 ou suivant pour essayer ces exemples. Si vous l'avez fait, vous pouvez maintenant taper les deux commandes suivantes pour télécharger automatiquement les bibliothèques dont cet exemple dépend :

```
npm install
```

Une fois terminé, vous pouvez lancer votre code d'exemple dans Node.js avec la commande suivante, en remplaçant les [...] par les arguments que vous voulez passer au programme :

```
node demo.js [...]
```

### Le même exemple, mais dans un navigateur

Si vous voulez voir comment utiliser la bibliothèque dans un navigateur plutôt que dans Node.js, changez de répertoire et allez dans **example\_html/Doc-GettingStarted-Yocto-Display**. Vous y trouverez un

fichier html, avec une section JavaScript similaire au code précédent, mais avec quelques variantes pour permettre une interaction à travers la page HTML plutôt que sur la console JavaScript

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Hello World</title>
  <script src="../../lib/yocto_api.js"></script>
  <script src="../../lib/yocto_display.js"></script>
  <script>
    var disp, l1;
    var h, w, y, x, vx, vy;

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

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

      // Select specified device, or use first available one
      let serial = document.getElementById('serial').value;
      if (serial == '') {
        // by default use any connected module suitable for the demo
        let anydisplay = YDisplay.FirstDisplay();
        if (anydisplay) {
          let module = await anydisplay.module();
          serial = await module.get_serialNumber();
          document.getElementById('serial').value = serial;
        }
      }
      disp = YDisplay.FindDisplay(serial + ".display");

      //clean up
      await disp.resetAll();

      // retrieve the display size
      w = await disp.get_displayWidth();
      h = await disp.get_displayHeight();

      // retrieve the first layer
      let l0 = await disp.get_displayLayer(0);
      // display a text in the middle of the screen
      await l0.drawText(w / 2, h / 2, l0.ALIGN_CENTER, "Hello world!");

      // visualize each corner
      await l0.moveTo(0, 5);
      await l0.lineTo(0, 0);
      await l0.lineTo(5, 0);
      await l0.moveTo(0, h - 6);
      await l0.lineTo(0, h - 1);
      await l0.lineTo(5, h - 1);
      await l0.moveTo(w - 1, h - 6);
      await l0.lineTo(w - 1, h - 1);
      await l0.lineTo(w - 6, h - 1);
      await l0.moveTo(w - 1, 5);
      await l0.lineTo(w - 1, 0);
      await l0.lineTo(w - 6, 0);

      // draw a circle in the top left corner of layer 1
      l1 = await disp.get_displayLayer(1);
      await l1.clear();
      await l1.drawCircle(h / 8, h / 8, h / 8);

      // and animate the layer
      x = 0;
      y = 0;
      vx = 1;
      vy = 1;
      refresh();
    }

    async function refresh() {

```

```

    if (await disp.isOnline()) {
        x += vx;
        y += vy;
        if ((x < 0) || (x > w - (h / 4))) vx = -vx;
        if ((y < 0) || (y > h - (h / 4))) vy = -vy;
        await ll.setLayerPosition(x, y, 0);
    } else {
        document.getElementById('msg').value = 'Module not connected';
    }
    setTimeout(refresh, 5);
}

startDemo();
</script>
</head>
<body>
Module to use:      <input id='serial'><span style='color:red;border:none;' id='msg'></span><br>
</body>
</html>

```

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

## 16.4. Contrôle de la partie module

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

```

"use strict";

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

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

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

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

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

```

}

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

## Modifications des réglages du module

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

```
"use strict";

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

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

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

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

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

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

## Énumération des modules

Obtenir la liste des modules connectés se fait à l'aide de la fonction `YModule.FirstModule()` qui renvoie le premier module trouvé, il suffit ensuite d'appeler la fonction `nextModule()` de cet



objet pour trouver les modules suivants, et ce tant que la réponse n'est pas un `null`. Ci-dessous un petit exemple listant les module connectés

```
"use strict";

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

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

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

async function refresh()
{
    try {
        let errmsg = new YErrorMsg();
        await YAPI.UpdateDeviceList(errmsg);

        let module = YModule.FirstModule();
        while(module) {
            let line = await module.get_serialNumber();
            line += '(' + (await module.get_productName()) + ')';
            console.log(line);
            module = module.nextModule();
        }
        setTimeout(refresh, 500);
    } catch(e) {
        console.log(e);
    }
}

try {
    startDemo();
} catch(e) {
    console.log(e);
}
```

## 16.5. Gestion des erreurs

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

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

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

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

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

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

## 17. Utilisation du Yocto-Display en PHP

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

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

Pour démarrer vos essais en PHP, vous allez avoir besoin d'un serveur PHP 7.1 ou plus récent <sup>1</sup> de préférence en local sur votre machine. Si vous souhaitez utiliser celui qui se trouve chez votre provider internet, c'est possible, mais vous devrez probablement configurer votre routeur ADSL pour qu'il accepte et forward les requêtes TCP sur le port 4444.

### 17.1. Préparation

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

- La librairie de programmation pour PHP<sup>2</sup>
- VirtualHub<sup>3</sup> pour Windows, macOS ou Linux selon l'OS que vous utilisez

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

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

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

---

<sup>1</sup> Quelques serveurs PHP gratuits: easyPHP pour Windows, MAMP pour macOS

<sup>2</sup> [www.yoctopuce.com/FR/libraries.php](http://www.yoctopuce.com/FR/libraries.php)

<sup>3</sup> [www.yoctopuce.com/FR/virtualhub.php](http://www.yoctopuce.com/FR/virtualhub.php)

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

- php7
- php8
- phpPSR

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

## 17.2. Contrôle de la fonction Display

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

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

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

// On récupère l'objet permettant d'interagir avec le module
$display = YDisplay::FindDisplay("YD128X32-123456.display");

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

Voyons maintenant en détail ce que font ces quelques lignes.

### yocto\_api.php et yocto\_display.php

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

### YAPI::RegisterHub

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

### YDisplay::FindDisplay

La fonction `YDisplay::FindDisplay` permet de retrouver un écran en fonction du numéro de série de son module hôte et de son nom de fonction. Mais vous pouvez tout aussi bien utiliser des noms logiques que vous auriez préalablement configurés. Imaginons un module Yocto-Display avec le numéro de série `YD128X32-123456` que vous auriez appelé `"MonModule"` et dont vous auriez nommé la fonction `display` `"MaFonction"`, les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
$display = YDisplay::FindDisplay("YD128X32-123456.display");
$display = YDisplay::FindDisplay("YD128X32-123456.MaFonction");
$display = YDisplay::FindDisplay("MonModule.display");
$display = YDisplay::FindDisplay("MonModule.MaFonction");
$display = YDisplay::FindDisplay("MaFonction");
```

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

## isOnline

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

## get\_displayLayer

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

## Un exemple réel

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

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

```
<HTML>
<HEAD>
<TITLE>Hello World</TITLE>
</HEAD>
<BODY>
<FORM method='get'>
<?php
include('../..//php8/yocto_api.php');
include('../..//php8/yocto_display.php');

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

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

@$serial = $_GET['serial'];
if ($serial != '') {
    // Check if a specified module is available online
    $disp = YDisplay::FindDisplay("$serial.display");
    if (!$disp->isOnline()) {
        die("Module not connected (check serial and USB cable)");
    }
} else {
    // or use any connected module suitable for the demo
    $disp = YDisplay::FirstDisplay();
    if(is_null($disp)) {
        die("No module connected (check USB cable)");
    }
}
$serial = $disp->get_module()->get_serialNumber();
Print("Module to use: <input name='serial' value='$serial'><br>");

$disp->resetAll();
// retrieve the display size
$w=$disp->get_displayWidth();
$h=$disp->get_displayHeight();

// retrieve the first layer
$l0=$disp->get_displayLayer(0);
$l0->clear();

// display a text in the middle of the screen
$l0->drawText($w / 2, $h / 2, YDisplayLayer::ALIGN_CENTER, "Hello world!" );
```

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

```
// visualize each corner
$10->moveTo(0,5);      $10->lineTo(0,0);      $10->lineTo(5,0);
$10->moveTo(0,$h-6);  $10->lineTo(0,$h-1);  $10->lineTo(5,$h-1);
$10->moveTo($w-1,$h-6);$10->lineTo($w-1,$h-1);$10->lineTo($w-6,$h-1);
$10->moveTo($w-1,5);  $10->lineTo($w-1,0);  $10->lineTo($w-6,0);
YAPI::FreeAPI();

?>
<br><input type='submit' value="Refresh">
</FORM>
</BODY>
</HTML>
```

## 17.3. Contrôle de la partie module

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

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

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

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

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

    if (isset($_GET['beacon'])) {
      if ($_GET['beacon']=='ON')
        $module->set_beacon(Y_BEACON_ON);
      else
        $module->set_beacon(Y_BEACON_OFF);
    }
    printf('serial: %s<br>', $module->get_serialNumber());
    printf('logical name: %s<br>', $module->get_logicalName());
    printf('luminosity: %s<br>', $module->get_luminosity());
    print('beacon: ');
    if($module->get_beacon() == Y_BEACON_ON) {
      printf("<input type='radio' name='beacon' value='ON' checked>ON ");
      printf("<input type='radio' name='beacon' value='OFF'>OFF<br>");
    } else {
      printf("<input type='radio' name='beacon' value='ON'>ON ");
    }
  }
  </?php>
  </FORM>
</BODY>
</HTML>
```

```

        printf("<input type='radio' name='beacon' value='OFF' checked>OFF<br>");
    }
    printf('upTime: %s sec<br>',intVal($module->get_upTime()/1000));
    printf('USB current: %sA<br>', $module->get_usbCurrent());
    printf('logs:<br><pre>%s</pre>', $module->get_lastLogs());
    YAPI::FreeAPI();
?>
<input type='submit' value='refresh'>
</FORM>
</BODY>
</HTML>

```

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

## Modifications des réglages du module

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

```

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

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

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

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

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

```

```

YAPI::FreeAPI();
?>
<input type='submit'>
</FORM>
</BODY>
</HTML>

```

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

## Enumération des modules

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

```

<HTML>
<HEAD>
  <TITLE>inventory</TITLE>
</HEAD>
<BODY>
<H1>Device list</H1>
<TT>
  <?php
  include('../..../php8/yocto_api.php');
  YAPI::RegisterHub("http://127.0.0.1:4444/");
  $module = YModule::FirstModule();
  while (!is_null($module)) {
    printf("%s (%s)<br>\n", $module->get_serialNumber(),
      $module->get_productName());
    $module=$module->nextModule();
  }
  YAPI::FreeAPI();
  ?>
</TT>
</BODY>
</HTML>

```

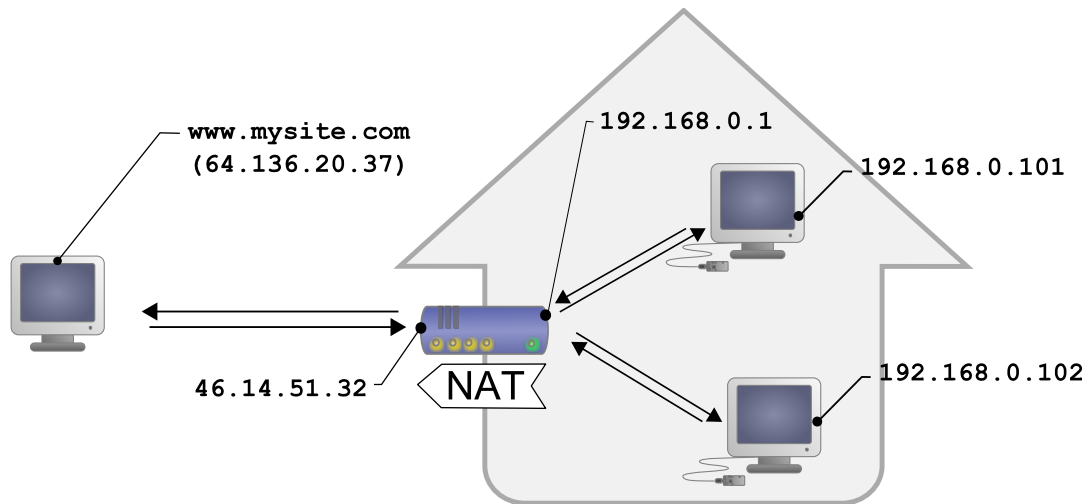
## 17.4. API par callback HTTP et filtres NAT

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

### Le filtre NAT, avantages et inconvénients

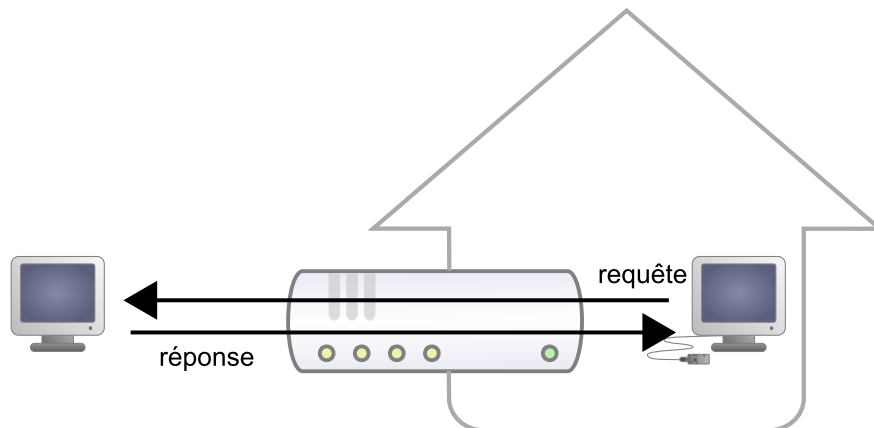
Un routeur DSL qui effectue de la traduction d'adresse réseau (NAT) fonctionne un peu comme un petit central téléphonique privé: les postes internes peuvent s'appeler l'un l'autre ainsi que faire des appels vers l'extérieur, mais vu de l'extérieur, il n'existe qu'un numéro de téléphone officiel, attribué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



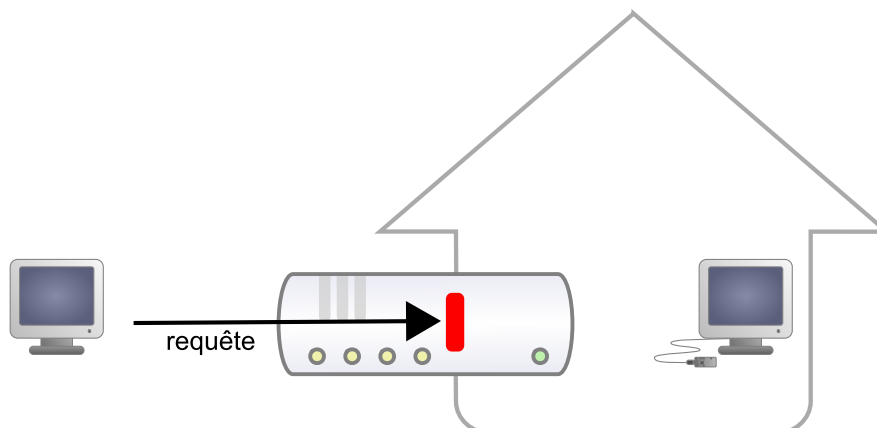


Configuration DSL typique, les machines du LAN sont isolées de l'extérieur par le router DSL

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



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

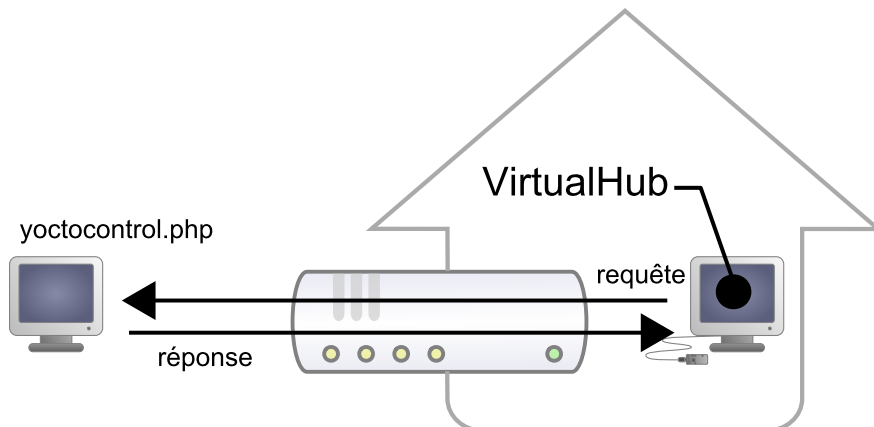


Mais les requêtes venant de l'extérieur sont bloquées.

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

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

L'API par callback HTTP 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.



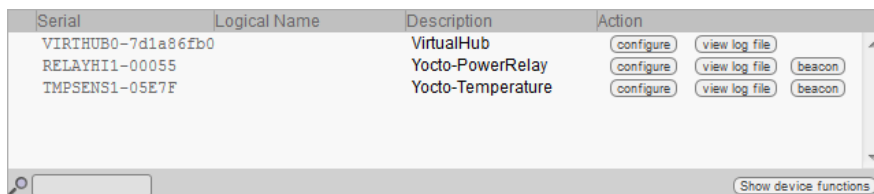
L'API par callback HTTP utilise le VirtualHub, et c'est lui qui initie les requêtes.

### Configuration

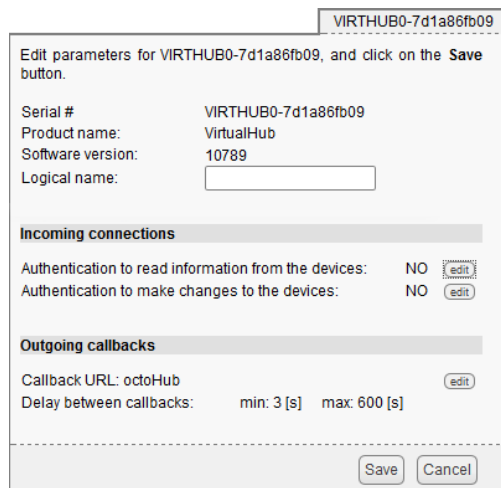
L'API callback se sert donc du *Virtual Hub* comme passerelle. Toutes les communications sont initiées par le *Virtual Hub*, ce sont donc des communication sortantes, et par conséquent parfaitement autorisée par le routeur DSL.

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, EcmaScript (Node.JS) et Java.

## 17.5. Gestion des erreurs

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

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

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

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

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

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

## 18. Utilisation du Yocto-Display en VisualBasic .NET

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

### 18.1. Installation

Téléchargez la librairie Yoctopuce pour Visual Basic depuis le site web de Yoctopuce<sup>1</sup>. Il n'y a pas de programme d'installation, copiez simplement le contenu du fichier zip dans le répertoire de votre choix. Vous avez besoin essentiellement du contenu du répertoire `Sources`. Les autres répertoires contiennent la documentation et quelques programmes d'exemple. Les projets d'exemple sont des projets Visual Basic 2010, si vous utilisez une version antérieure, il est possible que vous ayez à reconstruire la structure de ces projets.

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

La librairie Yoctopuce pour Visual Basic .NET se présente sous la forme d'une DLL et de fichiers sources en Visual Basic. La DLL n'est pas une DLL .NET mais une DLL classique, écrite en C, qui gère les communications à bas niveau avec les modules<sup>2</sup>. Les fichiers sources en Visual Basic gèrent la partie haut niveau de l'API. Vous avez donc besoin de cette DLL et des fichiers .vb du répertoire `Sources` pour créer un projet gérant des modules Yoctopuce.

#### Configuration d'un projet Visual Basic

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

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

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

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

---

<sup>1</sup> [www.yoctopuce.com/FR/libraries.php](http://www.yoctopuce.com/FR/libraries.php)

<sup>2</sup> Les sources de cette DLL sont disponibles dans l'API C++

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

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

### 18.3. Contrôle de la fonction Display

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

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

' On récupère l'objet permettant d'interagir avec le module
Dim display As YDisplay
display = YDisplay.FindDisplay("YD128X32-123456.display")

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

[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

#### YAPI.RegisterHub

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

#### YDisplay.FindDisplay

La fonction `YDisplay.FindDisplay` permet de retrouver un écran en fonction du numéro de série de son module hôte et de son nom de fonction. Mais vous pouvez tout aussi bien utiliser des noms logiques que vous auriez préalablement configurés. Imaginons un module Yocto-Display avec le numéros de série `YD128X32-123456` que vous auriez appelé `"MonModule"` et dont vous auriez nommé la fonction `display` `"MaFonction"`, les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128X32-123456.display")
display = YDisplay.FindDisplay("YD128X32-123456.MaFonction")
display = YDisplay.FindDisplay("MonModule.display")
display = YDisplay.FindDisplay("MonModule.MaFonction")
display = YDisplay.FindDisplay("MaFonction")
```

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

<sup>3</sup> 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 `YDisplay.FindDisplay` permet de savoir si le module correspondant est présent et en état de marche.

## get\_displayLayer

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

## Un exemple réel

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

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

```
Module Module1

    Private Sub Usage()
        Dim execname = System.AppDomain.CurrentDomain.FriendlyName
        Console.WriteLine("Usage:")
        Console.WriteLine(execname + " <serial_number>")
        Console.WriteLine(execname + " <logical_name> ")
        Console.WriteLine(execname + " any")
        System.Threading.Thread.Sleep(2500)
    End Sub

    Sub Main()
        Dim argv() As String = System.Environment.GetCommandLineArgs()
        Dim errmsg As String = ""
        Dim target As String
        Dim disp As YDisplay
        Dim l0, l1 As YDisplayLayer
        Dim h, w, y, x, vx, vy As Integer

        If argv.Length <= 1 Then Usage()

        target = argv(1)

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

        If target = "any" Then
            disp = YDisplay.FirstDisplay()
            If disp Is Nothing Then
                Console.WriteLine("No module connected (check USB cable) ")
            End If
        Else
            disp = YDisplay.FindDisplay(target + ".display")

        End If

        If Not (disp.isOnline()) Then
            Console.WriteLine("Module not connected (check identification and USB cable)")
        End If

        REM Display clean up
        disp.resetAll()
        REM retrieve the display size
        w = disp.get_displayWidth()
        h = disp.get_displayHeight()

        REM retrieve the first layer
        l0 = disp.get_displayLayer(0)
    End Sub
End Module
```

```

REM display a text in the middle of the screen
l0.drawText(CInt(w / 2), CInt(h / 2), Y_ALIGN.CENTER, "Hello world!")

REM visualize each corner
l0.moveTo(0, 5)
l0.lineTo(0, 0)
l0.lineTo(5, 0)
l0.moveTo(0, h - 6)
l0.lineTo(0, h - 1)
l0.lineTo(5, h - 1)
l0.moveTo(w - 1, h - 6)
l0.lineTo(w - 1, h - 1)
l0.lineTo(w - 6, h - 1)
l0.moveTo(w - 1, 5)
l0.lineTo(w - 1, 0)
l0.lineTo(w - 6, 0)

REM draw a circle in the top left corner of layer 1
l1 = disp.get_displayLayer(1)
l1.clear()
l1.drawCircle(CInt(h / 8), CInt(h / 8), CInt(h / 8))

REM and animate the layer
Console.WriteLine("Use Ctrl-C to stop")
x = 0
y = 0
vx = 1
vy = 1

While (disp.isOnline())
    x += vx
    y += vy
    If ((x < 0) Or (x > w - (h / 4))) Then vx = -vx
    If ((y < 0) Or (y > h - (h / 4))) Then vy = -vy
    l1.setLayerPosition(x, y, 0)
    YAPI.Sleep(5, errmsg)
End While
YAPI.FreeAPI()
End Sub

End Module

```

## 18.4. Contrôle de la partie module

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

```

Imports System.IO
Imports System.Environment

Module Module1

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

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

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

        If argv.Length < 2 Then usage()

        m = YModule.FindModule(argv(1)) REM use serial or logical name
    End Sub
End Module

```



```

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

End Module

```

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

  m = YModule.FindModule(argv(1)) REM use serial or logical name
  If m.isOnline() Then
    newname = argv(2)
    If (Not YAPI.CheckLogicalName(newname)) Then
      Console.WriteLine("Invalid name (" + newname + ")")
    End
  End If
  m.set_logicalName(newname)
  m.saveToFlash() REM do not forget this

```

```

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

End Sub

End Module

```

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

## Enumeration des modules

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

```

Module Module1

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

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

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

End Module

```

## 18.5. Gestion des erreurs

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

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

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

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

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

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



## 19. Utilisation du Yocto-Display en Delphi / Lazarus

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

Lazarus<sup>2</sup> est un IDE gratuit basé sur Free-Pascal qui n'a pas grand chose à envier à Delphi. Il a aussi l'avantage d'exister pour Windows et Linux. La librairie Yoctopuce pour Delphi est compatible avec Lazarus tant sous Windows que Linux.

Les librairies pour Delphi / Lazarus sont fournies non pas sous forme de composants VCL, mais directement sous forme de fichiers source. Ces fichiers sont compatibles avec la plupart des versions de Delphi / Lazarus<sup>3</sup>.

### 19.1. Préparation

Connectez-vous sur le site de Yoctopuce et téléchargez la la librairie Yoctopuce pour Delphi<sup>4</sup>. Décompressez le tout dans le répertoire de votre choix.

- Avec Delphi ajoutez le sous-répertoire *sources* de l'archive dans la liste des répertoires des librairies de Delphi<sup>5</sup>.
- Avec Lazarus, ouvrez les options de votre projet et ajoutez le répertoire *sources* dans le champs "other unit files"<sup>6</sup>.

#### Windows

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

---

<sup>1</sup> 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.

<sup>2</sup> [www.lazarus-ide.org](http://www.lazarus-ide.org)

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

<sup>4</sup> [www.yoctopuce.com/FR/libraries.php](http://www.yoctopuce.com/FR/libraries.php)

<sup>5</sup> Utilisez le menu **outils / options d'environnement**

<sup>6</sup> Utilisez le menu **Project / Project options/ Compiler options / Paths**

## Linux

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

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

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

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

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

## A propos des exemples

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

Notez que la plupart de ces exemples utilisent des paramètres passés sur la ligne de commande<sup>7</sup>.

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

## 19.2. Contrôle de la fonction Display

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

```
uses yocto_api, yocto_display;

var errmsg: string;
    display: TYDisplay;

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

// On récupère l'objet permettant d'interagir avec le module
display = yFindDisplay("YD128X32-123456.display")

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

Voyons maintenant en détail ce que font ces quelques lignes.

### yocto\_api et yocto\_display

Ces deux unités permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. *yocto\_api* doit toujours être utilisé, *yocto\_display* est nécessaire pour gérer les modules contenant un écran, comme le Yocto-Display.

<sup>7</sup> voir <http://www.yoctopuce.com/FR/article/a-propos-des-programmes-d-exemples>

## yRegisterHub

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

## yFindDisplay

La fonction `yFindDisplay` permet de retrouver un écran en fonction du numéro de série de son module hôte et de son nom de fonction. Mais vous pouvez tout aussi bien utiliser des noms logiques que vous auriez préalablement configurés. Imaginons un module Yocto-Display avec le numéros de série `YD128X32-123456` que vous auriez appelé `"MonModule"` et dont vous auriez nommé la fonction `display` `"MaFonction"`, les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display := yFindDisplay("YD128X32-123456.display");
display := yFindDisplay("YD128X32-123456.MaFonction");
display := yFindDisplay("MonModule.display");
display := yFindDisplay("MonModule.MaFonction");
display := yFindDisplay("MaFonction");
```

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

## isOnline

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

## get\_displayLayer

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

## Un exemple réel

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

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

```
program helloworld;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  {$IFDEF UNIX}
  windows,
  {$ENDIF UNIX}
  yocto_api,
  yocto_display;

Procedure Usage();
var
  exe : string;

begin
  exe:= ExtractFileName(paramstr(0));
  WriteLn(exe+' <serial_number>');
  WriteLn(exe+' <logical_name>');
  WriteLn(exe+' any');
  sleep(3000);
  halt;
End;

var
  disp      : TYDisplay;
  l0,l1     : TYDisplayLayer;
```

```

errmsg      : string;
w,h         : integer;
x,y,vx,vy  :integer;

begin

  if (paramcount<1) then usage();

  // Setup the API to use local USB devices
  if yRegisterHub('usb', errmsg)<>YAPI_SUCCESS then
  begin
    Write('RegisterHub error: '+errmsg);
    sleep(3000);
    exit;
  end;

  // first one of the two RGB leds
  if paramstr(1)='any' then
  begin
    disp := yFirstDisplay();
    if disp=nil then
      begin
        writeln('No module connected (check USB cable)');
        sleep(3000);
        halt;
      end
    end
  else
  disp:= YFindDisplay(paramstr(1)+'.display');

  // make sure it is online
  if not(disp.isOnline()) then
  begin
    writeln('No module connected (check USB cable)');
    sleep(3000);
    halt;
  end;

  // display clean up
  disp.resetAll();

  // retrieve the display size
  w:=disp.get_displayWidth();
  h:=disp.get_displayHeight();

  // retrieve the first layer
  L0:=Disp.get_displaylayer(0);

  // display a text in the middle of the screen
  L0.drawText(w div 2, h div 2, Y_ALIGN_CENTER, 'Hello world!');
  // visualize eah corner
  L0.moveto(0,5);L0.lineto(0,0);L0.lineto(5,0);
  L0.moveto(0,h-6);L0.lineto(0,H-1);L0.lineto(5,H-1);
  L0.moveto(w-1,h-6);L0.lineto(w-1,H-1);L0.lineto(w-6,H-1);
  L0.moveto(w-1,5);L0.lineto(w-1,0);L0.lineto(w-6,0);

  // draw a circle in the top left corner of layer 1
  L1:=Disp.get_displaylayer(1);
  L1.clear();
  L1.drawCircle(H div 8, H div 8, h div 8);

  // and animate the layer
  Writeln('Use Ctrl-C to stop');
  x:=0; y:=0; vx:=1; vy:=1;
  while (disp.isOnline()) do
  begin
    x:=x+vx;y:=y+vy;
    if (x<0) or (x>w-(h div 4)) then vx:=-vx;
    if (y<0) or (y>h-(h div 4)) then vy:=-vy;
    l1.setLayerPosition(x,y,0);
    ysleep(5,errmsg);
  end;
  yFreeAPI();

end.

```



## 19.3. Contrôle de la partie module

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

```

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

const
  serial = 'YD128X32-123456'; // use serial number or logical name

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

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

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

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

  module := yFindModule(serial);
  refresh(module);

  repeat
    read(c);
    case c of
      'r': refresh(module);
      'b': beacon(module,Y_BEACON_ON);
      ' ': beacon(module,Y_BEACON_OFF);
    end;
  until c = 'x';
  yFreeAPI();
end.

```

Chaque propriété xxx du module peut être lue grâce à une méthode du type `get_xxxx()`, et les propriétés qui se sont pas en lecture seule peuvent être modifiées à l'aide de la méthode `set_xxx()` Pour plus de détails concernant ces fonctions utilisées, reportez-vous aux chapitre API

## Modifications des réglages du module

Lorsque que vous souhaitez modifier les réglages d'un module, il suffit d'appeler la fonction `set_xxx()` correspondante, cependant cette modification n'a lieu que dans la mémoire vive du module: si le module redémarre, les modifications seront perdues. Pour qu'elle soient mémorisées de manière persistante, il est nécessaire de demander au module de sauvegarder sa configuration courante dans sa mémoire non volatile. Pour cela il faut utiliser la méthode `saveToFlash()`. Inversement il est possible de forcer le module à oublier ses réglages courants en utilisant la méthode `revertFromFlash()`. Ce petit exemple ci-dessous vous permet changer le nom logique d'un module.

```

program savesettings;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  yocto_api;

const
  serial = 'YD128X32-123456'; // use serial number or logical name

var
  module : TYModule;
  errmsg : string;
  newname : string;

begin
  // Setup the API to use local USB devices
  if yRegisterHub('usb', errmsg) <> YAPI_SUCCESS then
  begin
    Write('RegisterHub error: '+errmsg);
    exit;
  end;

  module := yFindModule(serial);
  if (not(module.isOnline)) then
  begin
    writeln('Module not connected (check identification and USB cable)');
    exit;
  end;

  Writeln('Current logical name : '+module.get_logicalName());
  Write('Enter new name : ');
  Readln(newname);
  if (not(yCheckLogicalName(newname))) then
  begin
    Writeln('invalid logical name');
    exit;
  end;
  module.set_logicalName(newname);
  module.saveToFlash();
  yFreeAPI();
  Writeln('logical name is now : '+module.get_logicalName());
end.

```

Attention, le nombre de cycles d'écriture de la mémoire non volatile du module est limité. Passé cette limite plus rien ne garantit que la sauvegarde des réglages se passera correctement. Cette limite, liée à la technologie employée par le micro-processeur du module se situe aux alentours de 100000 cycles. Pour résumer vous ne pouvez employer la fonction `saveToFlash()` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

## Énumération des modules

Obtenir la liste des modules connectés se fait à l'aide de la fonction `yFirstModule()` qui renvoie le premier module trouvé, il suffit ensuite d'appeler la fonction `nextModule()` de cet objet pour trouver les modules suivants, et ce tant que la réponse n'est pas un `nil`. Ci-dessous un petit exemple listant les modules connectés

```

program inventory;
{$APPTYPE CONSOLE}
uses

```

```

SysUtils,
yocto_api;

var
  module : TYModule;
  errmsg : string;

begin
  // Setup the API to use local USB devices
  if yRegisterHub('usb', errmsg)<>YAPI_SUCCESS then
  begin
    Write('RegisterHub error: '+errmsg);
    exit;
  end;

  Writeln('Device list');

  module := yFirstModule();
  while module<>nil do
  begin
    Writeln( module.get_serialNumber()+ ' ('+module.get_productName()+')');
    module := module.nextModule();
  end;
  yFreeAPI();
end.

```

## 19.4. Gestion des erreurs

Lorsque vous implémentez un programme qui doit interagir avec des modules USB, vous ne pouvez pas faire abstraction de la gestion des erreurs. Il y aura forcément une occasion où un utilisateur aura débranché le périphérique, soit avant de lancer le programme, soit même en pleine opération. La librairie Yoctopuce est prévue pour vous aider à supporter ce genre de comportements, mais votre code doit néanmoins être fait pour se comporter au mieux pour interpréter les erreurs signalées par la librairie.

La manière la plus simple de contourner le problème est celle que nous avons employé pour les petits exemples précédents de ce chapitre: avant d'accéder à un module, on vérifie qu'il est en ligne avec la méthode `isOnline()` et on suppose ensuite qu'il va y rester pendant la fraction de seconde nécessaire à exécuter les lignes de code suivantes. Ce n'est pas parfait, mais ça peut suffire dans certains cas. Il faut toutefois être conscient qu'on ne peut pas totalement exclure une erreur se produisant après le `isOnline()`, qui pourrait faire planter le programme. La seule manière de l'éviter est d'implémenter une des deux techniques de gestion des erreurs décrites ci-dessous.

La méthode recommandée par la plupart des langages de programmation pour la gestion des erreurs imprévisibles est l'utilisation d'exceptions. C'est le comportement par défaut de la librairie Yoctopuce. Si une erreur se produit alors qu'on essaie d'accéder à un module, la librairie va lancer une exception. Dans ce cas, de trois choses l'une:

- Si votre code attrape l'exception au vol et la gère, et tout se passe bien.
- Si votre programme tourne dans le debugger, vous pourrez relativement facilement déterminer où le problème s'est produit, et voir le message explicatif lié à l'exception.
- Sinon... l'exception va crasher votre programme, boum!

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `YAPI.DisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur `NomDeClasse.STATE_INVALID`, une méthode `get_currentValue` retournera une valeur `NomDeClasse.CURRENTVALUE_INVALID`, etc. Dans tous les cas, la valeur retournée sera du

type attendu, et ne sera pas un pointeur nul qui risquerait de faire crasher votre programme. Au pire, si vous affichez la valeur sans la tester, elle sera hors du cadre attendu pour la valeur retournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI.SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

Quand vous travaillez sans les exceptions, il est possible d'obtenir un code d'erreur et un message expliquant l'origine de l'erreur en le demandant à l'objet qui a retourné une erreur à l'aide des méthodes `errType()` et `errMessage()`. Ce sont les même informations qui auraient été associées à l'exception si elles avaient été actives.

## 20. Utilisation du Yocto-Display avec Universal Windows Platform

Universal Windows Platform, abrégé UWP, n'est pas un langage à proprement parler mais une plate-forme logicielle créée par Microsoft. Cette plateforme permet d'exécuter un nouveau type d'applications : les applications universelles Windows. Ces applications peuvent fonctionner sur toutes les machines qui fonctionnent sous Windows 10. Cela comprend les PC, les tablettes, les smartphones, la Xbox One, mais aussi Windows IoT Core.

La bibliothèque Yoctopuce UWP permet d'utiliser les modules Yoctopuce dans une application universelle Windows et est entièrement écrite C#. Elle peut être ajoutée à un projet Visual Studio 2017<sup>1</sup>.

### 20.1. Fonctions bloquantes et fonctions asynchrones

La bibliothèque Universal Windows Platform n'utilise pas l'API win32 mais uniquement l'API Windows Runtime qui est disponible sur toutes les versions de Windows 10 et pour n'importe quelle architecture. Grâce à cela la bibliothèque UWP peut être utilisée sur toutes les versions de Windows 10, y compris Windows 10 IoT Core.

Cependant, l'utilisation des nouvelles API UWP n'est pas sans conséquence : l'API Windows Runtime pour accéder aux ports USB est asynchrone, et par conséquent la bibliothèque Yoctopuce doit aussi être asynchrone. Concrètement les méthodes asynchrones ne retournent pas directement le résultat mais un objet `Task` ou `Task<>` et le résultat peut être obtenu plus tard. Fort heureusement, le langage C# version 6 supporte les mots-clés `async` et `await` qui simplifient beaucoup l'utilisation de ces fonctions. Il est ainsi possible d'utiliser les fonctions asynchrones de la même manière que les fonctions traditionnelles pour autant que les deux règles suivantes soient respectées :

- La méthode est déclarée comme asynchrone à l'aide du mot-clé `async`
- le mot-clé `await` est ajouté lors de l'utilisation d'une fonction asynchrone

Exemple :

```
async Task<int> MyFunction(int val)
{
    // do some long computation
    ...
    return result;
}
```

---

<sup>1</sup> <https://www.visualstudio.com/fr/vs/>

```
int res = await MyFunction(1234);
```

Notre librairie suit ces deux règles et peut donc utiliser la notation `await`.

Pour ne pas devoir vous poser la question pour chaque méthode de savoir si elle est asynchrone ou pas, la convention est la suivante: **toutes les méthodes publiques** de la librairie UWP **sont asynchrones**, c'est-à-dire qui faut les appeler en ajoutant le mot clef `await`, **sauf**:

- `GetTickCount()`, parce que mesurer le temps de manière asynchrone n'a pas beaucoup de sens...
- `FindModule()`, `FirstModule()`, `nextModule()`,... parce que la détection et l'énumération des modules est faite en tâche de fond sur des structures internes qui sont gérées de manière transparente, et qu'il n'est donc pas nécessaire de faire des opérations bloquantes durant le simple parcours de ces listes de modules.

## 20.2. Installation

Téléchargez la librairie Yoctopuce pour Universal Windows Platform depuis le site web de Yoctopuce <sup>2</sup>. Il n'y a pas de programme d'installation, copiez simplement le contenu du fichier zip dans le répertoire de votre choix. Vous avez besoin essentiellement du contenu du répertoire `Sources`. Les autres répertoires contiennent la documentation et quelques programmes d'exemple. Les projets d'exemple sont des projets Visual Studio 2017 qui est disponible sur le site de Microsoft <sup>3</sup>.

## 20.3. Utilisation l'API Yoctopuce dans un projet Visual Studio

Commencez par créer votre projet, puis depuis le panneau **Explorateur de solutions** effectuez un clic droit sur votre projet, et choisissez **Ajouter** puis **Élément existant**.

Une fenêtre de sélection de fichiers apparaît: sélectionnez tous les fichiers du répertoire `Sources` de la librairie.

Vous avez alors le choix entre simplement ajouter ces fichiers à votre projet, ou les ajouter en tant que lien (le bouton **Ajouter** est en fait un menu déroulant). Dans le premier cas, Visual Studio va copier les fichiers choisis dans votre projet, dans le second Visual Studio va simplement garder un lien sur les fichiers originaux. Il est recommandé d'utiliser des liens, une éventuelle mise à jour de la librairie sera ainsi beaucoup plus facile.

### Le fichier `Package.appxmanifest`

Par défaut, une application Universal Windows n'a pas le droit d'accéder aux ports USB. Si l'on désire accéder à un périphérique USB, il faut impérativement le déclarer dans le fichier `Package.appxmanifest`.

Malheureusement, la fenêtre d'édition de ce fichier ne permet pas cette opération et il faut modifier le fichier `Package.appxmanifest` à la main. Dans le panneau "Solutions Explorer", faites un clic droit sur le fichier `Package.appxmanifest` et sélectionnez "View Code".

Dans ce fichier XML, il faut rajouter un `nod DeviceCapability` dans le `nod Capabilities`. Ce `nod` doit avoir un attribut "Name" qui vaut "humaninterfacedevice".

A l'intérieur de ce `nod`, il faut déclarer tous les modules qui peuvent être utilisés. Concrètement, pour chaque module, il faut ajouter un `nod Device` avec un attribut "Id" dont la valeur est une chaîne de caractères "vidpid:USB\_VENDORID USB\_DEVICE\_ID". Le `USB_VENDORID` de Yoctopuce est 24e0 et le `USB_DEVICE_ID` de chaque module Yoctopuce peut être trouvé dans la

<sup>2</sup> [www.yoctopuce.com/FR/libraries.php](http://www.yoctopuce.com/FR/libraries.php)

<sup>3</sup> <https://www.visualstudio.com/downloads/>

documentation dans la section "Caractéristiques". Pour finir, le n u d "Device" doit contenir un n u d "Function" avec l'attribut "Type" dont la valeur est "usage:ff00 0001".

Pour le Yocto-Display voici ce qu'il faut ajouter dans le n u d "Capabilities":

```
<DeviceCapability Name="humaninterfacedevice">
  <!-- Yocto-Display -->
  <Device Id="vidpid:24e0 002D">
    <Function Type="usage:ff00 0001" />
  </Device>
</DeviceCapability>
```

Malheureusement, il n'est pas possible d'écrire un règle qui autorise tous les modules Yoctopuce, par conséquent il faut impérativement ajouter chaque module que l'on désire utiliser.

## 20.4. Contrôle de la fonction Display

Il suffit de quelques lignes de code pour piloter un Yocto-Display. Voici le squelette d'un fragment de code *c#* qui utilise la fonction Display.

```
[...]
// On active la détection des modules sur USB
await YAPI.RegisterHub("usb");
[...]

// On récupère l'objet permettant d'interagir avec le module
YDisplay display = YDisplay.FindDisplay("YD128X32-123456.display");

// Pour gérer le hot-plug, on vérifie que le module est là
if (await display.IsOnline())
{
    // Use display.get_displayLayer()
    ...
}
[...]
```

Voyons maintenant en détail ce que font ces quelques lignes.

### YAPI.RegisterHub

La fonction `YAPI.RegisterHub` initialise l'API de Yoctopuce en indiquant où les modules doivent être recherchés. Le paramètre est l'adresse du virtual hub capable de voir les modules. Si l'on passe la chaîne de caractère "usb", l'API va travailler avec les modules connectés localement à la machine. Si l'initialisation se passe mal, une exception sera générée.

### YDisplay.FindDisplay

La fonction `YDisplay.FindDisplay` permet de retrouver un écran en fonction du numéro de série de son module hôte et de son nom de fonction. Mais vous pouvez tout aussi bien utiliser des noms logiques que vous auriez préalablement configurés. Imaginons un module Yocto-Display avec le numéro de série `YD128X32-123456` que vous auriez appelé "MonModule" et dont vous auriez nommé la fonction `display` "MaFonction", les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
display = YDisplay.FindDisplay("YD128X32-123456.display");
display = YDisplay.FindDisplay("YD128X32-123456.MaFonction");
display = YDisplay.FindDisplay("MonModule.display");
display = YDisplay.FindDisplay("MonModule.MaFonction");
display = YDisplay.FindDisplay("MaFonction");
```

`YDisplay.FindDisplay` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'écran.

## isOnline

La méthode `isOnline()` de l'objet renvoyé par `YDisplay.FindDisplay` permet de savoir si le module correspondant est présent et en état de marche.

## get\_displayLayer

La méthode `get_displayLayer()` de l'objet renvoyé par `YDisplay.FindDisplay` permet récupérer un objet correspondant à une des couches de l'écran. Cet objet fourni toutes les routines graphiques.

## 20.5. Un exemple concret

Lancez Visual Studio et ouvrez le projet correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-Display** de la librairie Yoctopuce.

Le projets Visual Studio contient de nombreux fichiers dont la plupart ne sont pas liés à l'utilisation de la librairie Yoctopuce. Pour simplifier la lecture du code nous avons regroupé tout le code qui utilise la librairie dans la classe `Demo` qui se trouve dans le fichier `demo.cs`. Les propriétés de cette classe correspondent aux différents champs qui sont affichés à l'écran, et la méthode `Run()` contient le code qui est exécuté quand le bouton "Start" est pressé.

Vous reconnaîtrez dans cet exemple l'utilisation des fonctions expliquées ci-dessus, cette fois utilisées avec le décorum nécessaire à en faire un petit programme d'exemple concret.

```
using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.UI.Xaml.Controls;
using com.yoctopuce.YoctoAPI;

namespace Demo
{
    public class Demo : DemoBase
    {
        public string HubURL { get; set; }
        public string Target { get; set; }
        public string Message { get; set; }

        public override async Task<int> Run()
        {
            try {
                await YAPI.RegisterHub(HubURL);

                YDisplay disp;
                YDisplayLayer l0, l1;
                int h, w, y, x, vx, vy;

                // find the display according to command line parameters
                if (Target.ToLower() == "any") {
                    disp = YDisplay.FirstDisplay();
                    if (disp == null) {
                        WriteLine("No module connected (check USB cable) ");
                        return -1;
                    }
                } else {
                    disp = YDisplay.FindDisplay(Target + ".display");
                }

                if (!await disp.isOnline()) {
                    WriteLine("Module not connected (check identification and USB cable) ");
                    return -1;
                }

                //clean up
                await disp.resetAll();

                // retrieve the display size
                w = await disp.get_displayWidth();
                h = await disp.get_displayHeight();
            }
        }
    }
}
```



```

// retrieve the first layer
l0 = await disp.get_displayLayer(0);

// display a text in the middle of the screen
await l0.drawText(w / 2, h / 2, YDisplayLayer.ALIGN.CENTER, Message);

// visualize each corner
await l0.moveTo(0, 5);
await l0.lineTo(0, 0);
await l0.lineTo(5, 0);
await l0.moveTo(0, h - 6);
await l0.lineTo(0, h - 1);
await l0.lineTo(5, h - 1);
await l0.moveTo(w - 1, h - 6);
await l0.lineTo(w - 1, h - 1);
await l0.lineTo(w - 6, h - 1);
await l0.moveTo(w - 1, 5);
await l0.lineTo(w - 1, 0);
await l0.lineTo(w - 6, 0);

// draw a circle in the top left corner of layer 1
l1 = await disp.get_displayLayer(1);
await l1.clear();
await l1.drawCircle(h / 8, h / 8, h / 8);

// and animate the layer
x = 0;
y = 0;
vx = 1;
vy = 1;
while (await disp.isOnline()) {
    x += vx;
    y += vy;
    if ((x < 0) || (x > w - (h / 4))) vx = -vx;
    if ((y < 0) || (y > h - (h / 4))) vy = -vy;
    await l1.setLayerPosition(x, y, 0);
    await YAPI.Sleep(5);
}

WriteLine("Module not connected (check identification and USB cable) ");
} catch (YAPI_Exception ex) {
    WriteLine("error: " + ex.Message);
}

await YAPI.FreeAPI();
return 0;
}
}
}

```

## 20.6. Contrôle de la partie module

Chaque module peut-être contrôlé d'une manière similaire, vous trouverez ci-dessous un simple programme d'exemple affichant les principaux paramètres d'un module et permettant d'activer la balise de localisation.

```

using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.UI.Xaml.Controls;
using com.yoctopuce.YoctoAPI;

namespace Demo
{
    public class Demo : DemoBase
    {
        public string HubURL { get; set; }
        public string Target { get; set; }
        public bool Beacon { get; set; }

        public override async Task<int> Run()
        {
            YModule m;

```

```

string errmsg = "";

if (await YAPI.RegisterHub(HubURL) != YAPI.SUCCESS) {
    WriteLine("RegisterHub error: " + errmsg);
    return -1;
}
m = YModule.FindModule(Target + ".module"); // use serial or logical name
if (await m.isOnline()) {
    if (Beacon) {
        await m.set_beacon(YModule.BEACON_ON);
    } else {
        await m.set_beacon(YModule.BEACON_OFF);
    }

    WriteLine("serial: " + await m.get_serialNumber());
    WriteLine("logical name: " + await m.get_logicalName());
    WriteLine("luminosity: " + await m.get_luminosity());
    Write("beacon: ");
    if (await m.get_beacon() == YModule.BEACON_ON)
        WriteLine("ON");
    else
        WriteLine("OFF");
    WriteLine("upTime: " + (await m.get_upTime() / 1000) + " sec");
    WriteLine("USB current: " + await m.get_usbCurrent() + " mA");
    WriteLine("Logs:\r\n" + await m.get_lastLogs());
} else {
    WriteLine(Target + " not connected on" + HubURL +
        "(check identification and USB cable)");
}
await YAPI.FreeAPI();
return 0;
}
}
}

```

Chaque propriété xxx du module peut être lue grâce à une méthode du type `YModule.get_xxxx()`, et les propriétés qui se sont pas en lecture seule peuvent être modifiées à l'aide de la méthode `YModule.set_xxx()`. Pour plus de détails concernant ces fonctions utilisées, reportez-vous aux chapitre API

## Modifications des réglages du module

Lorsque que vous souhaitez modifier les réglages d'un module, il suffit d'appeler la fonction `YModule.set_xxx()` correspondante, cependant cette modification n'a lieu que dans la mémoire vive du module: si le module redémarre, les modifications seront perdues. Pour qu'elle soient mémorisées de manière persistante, il est nécessaire de demander au module de sauvegarder sa configuration courante dans sa mémoire non volatile. Pour cela il faut utiliser la méthode `YModule.saveToFlash()`. Inversement il est possible de forcer le module à oublier ses réglages courants en utilisant la méthode `YModule.revertFromFlash()`. Ce petit exemple ci-dessous vous permet changer le nom logique d'un module.

```

using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.UI.Xaml.Controls;
using com.yoctopuce.YoctoAPI;

namespace Demo
{
    public class Demo : DemoBase
    {
        public string HubURL { get; set; }
        public string Target { get; set; }
        public string LogicalName { get; set; }

        public override async Task<int> Run()
        {
            try {
                YModule m;

                await YAPI.RegisterHub(HubURL);

                m = YModule.FindModule(Target); // use serial or logical name
            }
        }
    }
}

```

```

    if (await m.isOnline()) {
        if (!YAPI.CheckLogicalName(LogicalName)) {
            WriteLine("Invalid name (" + LogicalName + ")");
            return -1;
        }

        await m.set_logicalName(LogicalName);
        await m.saveToFlash(); // do not forget this
        Write("Module: serial= " + await m.get_serialNumber());
        WriteLine(" / name= " + await m.get_logicalName());
    } else {
        Write("not connected (check identification and USB cable");
    }
} catch (YAPI_Exception ex) {
    WriteLine("RegisterHub error: " + ex.Message);
}
await YAPI.FreeAPI();
return 0;
}
}
}

```

Attention, le nombre de cycles d'écriture de la mémoire non volatile du module est limité. Passé cette limite plus rien ne garantit que la sauvegarde des réglages se passera correctement. Cette limite, liée à la technologie employée par le micro-processeur du module se situe aux alentours de 100000 cycles. Pour résumer vous ne pouvez employer la fonction `YModule.saveToFlash()` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

## Enumeration des modules

Obtenir la liste des modules connectés se fait à l'aide de la fonction `YModule.yFirstModule()` qui renvoie le premier module trouvé, il suffit ensuite d'appeler la méthode `nextModule()` de cet objet pour trouver les modules suivants, et ce tant que la réponse n'est pas un `null`. Ci-dessous un petit exemple listant les modules connectés

```

using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.UI.Xaml.Controls;
using com.yoctopuce.YoctoAPI;

namespace Demo
{
    public class Demo : DemoBase
    {
        public string HubURL { get; set; }

        public override async Task<int> Run()
        {
            YModule m;
            try {
                await YAPI.RegisterHub(HubURL);

                WriteLine("Device list");
                m = YModule.FirstModule();
                while (m != null) {
                    WriteLine(await m.get_serialNumber()
                        + " (" + await m.get_productName() + ")");
                    m = m.nextModule();
                }
            } catch (YAPI_Exception ex) {
                WriteLine("Error:" + ex.Message);
            }
            await YAPI.FreeAPI();
            return 0;
        }
    }
}

```

## 20.7. Gestion des erreurs

Lorsque vous implémentez un programme qui doit interagir avec des modules USB, vous ne pouvez pas faire abstraction de la gestion des erreurs. Il y aura forcément une occasion où un utilisateur aura débranché le périphérique, soit avant de lancer le programme, soit même en pleine opération. La librairie Yoctopuce est prévue pour vous aider à supporter ce genre de comportements, mais votre code doit néanmoins être fait pour se comporter au mieux pour interpréter les erreurs signalées par la librairie.

La manière la plus simple de contourner le problème est celle que nous avons employé pour les petits exemples précédents de ce chapitre: avant d'accéder à un module, on vérifie qu'il est en ligne avec la méthode `isOnline()` et on suppose ensuite qu'il va y rester pendant la fraction de seconde nécessaire à exécuter les lignes de code suivantes. Ce n'est pas parfait, mais ça peut suffire dans certains cas. Il faut toutefois être conscient qu'on ne peut pas totalement exclure une erreur se produisant après le `isOnline()`, qui pourrait faire planter le programme.

Dans la librairie Universal Windows Platform, le traitement d'erreur est implémenté au moyen d'exceptions. Vous devrez donc intercepter et traiter correctement ces exceptions si vous souhaitez avoir un projet fiable qui ne crashera pas des que vous débrancherez un module.

Les exceptions lancées de la librairie sont toujours de type `YAPI_Exception`, ce qui permet facilement de les séparer des autres exceptions dans un bloc `try{...} catch{...}`.

Exemple:

```
try {
    ....
} catch (YAPI_Exception ex) {
    Debug.WriteLine("Exception from Yoctopuce lib:" + ex.Message);
} catch (Exception ex) {
    Debug.WriteLine("Other exceptions :" + ex.Message);
}
```

## 21. Utilisation du Yocto-Display en Objective-C

Objective-C est le langage de prédilection pour programmer sous macOS, en raison de son intégration avec le générateur d'interfaces Cocoa. Yoctopuce supporte les versions de XCode supportées par Apple. La librairie Yoctopuce est compatible ARC. Il vous sera donc possible de coder vos projet soit en utilisant la traditionnelle méthode de *retain / release*, soit en activant l'*Automatic Reference Counting*.

Les librairies Yoctopuce<sup>1</sup> 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<sup>2</sup> avec des séquences vidéo montrant comment intégrer les fichiers de la librairie à vos projets.

### 21.1. Contrôle de la fonction Display

Il suffit de quelques lignes de code pour piloter un Yocto-Display. Voici le squelette d'un fragment de code Objective-C qui utilise la fonction Display.

```
#import "yocto_api.h"
#import "yocto_display.h"

...
NSError *error;
[YAPI RegisterHub:@"usb": &error]
...
// On récupère l'objet représentant le module (ici connecté en local sur USB)
display = [YDisplay FindDisplay:@"YD128X32-123456.display"];

// Pour gérer le hot-plug, on vérifie que le module est là
if([display isOnline])
```

<sup>1</sup> [www.yoctopuce.com/FR/libraries.php](http://www.yoctopuce.com/FR/libraries.php)

<sup>2</sup> [www.yoctopuce.com/FR/article/nouvelle-librairie-objective-c-pour-mac-os-x](http://www.yoctopuce.com/FR/article/nouvelle-librairie-objective-c-pour-mac-os-x)

```
{
    // Utiliser [display get_displayLayer]
    ...
}
```

Voyons maintenant en détail ce que font ces quelques lignes.

## yocto\_api.h et yocto\_display.h

Ces deux fichiers importés permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api.h` doit toujours être utilisé, `yocto_display.h` est nécessaire pour gérer les modules contenant un écran, comme le Yocto-Display.

### [YAPI RegisterHub]

La fonction `[YAPI RegisterHub]` initialise l'API de Yoctopuce en indiquant où les modules doivent être recherchés. Utilisée avec le paramètre `@"usb"`, elle permet de travailler avec les modules connectés localement à la machine. Si l'initialisation se passe mal, cette fonction renverra une valeur différente de `YAPI_SUCCESS`, et retournera via le paramètre `errmsg` une explication du problème.

### [Display FindDisplay]

La fonction `[Display FindDisplay]`, permet de retrouver un écran en fonction du numéro de série de son module hôte et de son nom de fonction. Mais vous pouvez tout aussi bien utiliser des noms logiques que vous auriez préalablement configurés. Imaginons un module Yocto-Display avec le numéro de série `YD128X32-123456` que vous auriez appelé `"MonModule"` et dont vous auriez nommé la fonction `display` `"MaFonction"`, les cinq appels suivants seront strictement équivalents (pour autant que `MaFonction` ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YDisplay *display = [YDisplay FindDisplay:@"YD128X32-123456.display"];
YDisplay *display = [YDisplay FindDisplay:@"YD128X32-123456.MaFonction"];
YDisplay *display = [YDisplay FindDisplay:@"MonModule.display"];
YDisplay *display = [YDisplay FindDisplay:@"MonModule.MaFonction"];
YDisplay *display = [YDisplay FindDisplay:@"MaFonction"];
```

`[YDisplay FindDisplay]` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler l'écran.

### isOnline

La méthode `isOnline` de l'objet renvoyé par `[YDisplay FindDisplay]` permet de savoir si le module correspondant est présent et en état de marche.

### get\_displayLayer

La méthode `get_displayLayer()` de l'objet renvoyé par `YDisplay.FindDisplay` permet récupérer un objet correspondant à une des couches de l'écran. Cet objet fournit toutes les routines graphiques.

### Un exemple réel

Lancez Xcode 4.2 et ouvrez le projet exemple correspondant, fourni dans le répertoire **Exemples/Doc-GettingStarted-Yocto-Display** de la librairie Yoctopuce.

Vous reconnaîtrez dans cet exemple l'utilisation des fonctions expliquées ci-dessus, cette fois utilisées avec le décorum nécessaire à en faire un petit programme d'exemple concret.

```
#import <Foundation/Foundation.h>
#import "yocto_api.h"
#import "yocto_display.h"

static void usage(void)
{
    NSLog(@"usage: demo <serial_number> ");
    NSLog(@"      demo <logical_name>");
    NSLog(@"      demo any          (use any discovered device)");
}
```

```

    exit(1);
}

int main(int argc, const char * argv[])
{
    NSError *error;
    YDisplay *disp;
    YDisplayLayer *l0, *l1;
    int h, w, y, x, vx, vy;
    @autoreleasepool {

        // Setup the API to use local USB devices
        if([YAPI RegisterHub:@"usb": &error] != YAPI_SUCCESS) {
            NSLog(@"RegisterHub error: %@", [error localizedDescription]);
            usage();
            return 1;
        }
        if(argc < 2) {
            disp = [YDisplay FirstDisplay];
            if(disp == nil) {
                NSLog(@"No module connected (check USB cable)");
                usage();
                return 1;
            }
        } else {
            NSString *target = [NSString stringWithUTF8String:argv[1]];

            disp = [YDisplay FindDisplay:target];
            if(![disp isOnline]) {
                NSLog(@"Module not connected (check identification and USB cable)");
                usage();
                return 1;
            }
        }
        // clear screen (and all layers)
        [disp resetAll];
        // retrieve the display size
        w = [disp get_displayWidth];
        h = [disp get_displayHeight];

        // retrieve the first layer
        l0 = [disp get_displayLayer:0];

        // display a text in the middle of the screen
        [l0 drawText:w / 2 :h / 2 :Y_ALIGN_CENTER :@"Hello world!"];

        // visualize each corner
        [l0 moveTo:0 :5];
        [l0 lineTo:0 :0];
        [l0 lineTo:5 :0];
        [l0 moveTo:0 :h - 6];
        [l0 lineTo:0 :h - 1];
        [l0 lineTo:5 :h - 1];
        [l0 moveTo:w - 1 :h - 6];
        [l0 lineTo:w - 1 :h - 1];
        [l0 lineTo:w - 6 :h - 1];
        [l0 moveTo:w - 1 :5];
        [l0 lineTo:w - 1 :0];
        [l0 lineTo:w - 6 :0];

        // draw a circle in the top left corner of layer 1
        l1 = [disp get_displayLayer:1];
        [l1 clear];
        [l1 drawCircle:h / 8 :h / 8 :h / 8];

        // and animate the layer
        NSLog(@"Use Ctrl-C to stop");
        x = 0;
        y = 0;
        vx = 1;
        vy = 1;
        while ([disp isOnline]) {
            x += vx;
            y += vy;
            if ((x < 0) || (x > w - (h / 4))) vx = -vx;
            if ((y < 0) || (y > h - (h / 4))) vy = -vy;
            [l1 setLayerPosition:x :y :0];
            [YAPI Sleep:5 :&error];
        }
    }
}

```

```

    }
    [YAPI FreeAPI];
}

return 0;
}

```

## 21.2. Contrôle de la partie module

Chaque module peut-être contrôlé d'une manière similaire, vous trouverez ci dessous un simple programme d'exemple affichant les principaux paramètres d'un module et permettant d'activer la balise de localisation.

```

#import <Foundation/Foundation.h>
#import "yocto_api.h"

static void usage(const char *exe)
{
    NSLog(@"usage: %s <serial or logical name> [ON/OFF]\n", exe);
    exit(1);
}

int main (int argc, const char * argv[])
{
    NSError *error;

    @autoreleasepool {
        // Setup the API to use local USB devices
        if([YAPI RegisterHub:@"usb": &error] != YAPI_SUCCESS) {
            NSLog(@"RegisterHub error: %@", [error localizedDescription]);
            return 1;
        }
        if(argc < 2)
            usage(argv[0]);
        NSString *serial_or_name = [NSString stringWithUTF8String:argv[1]];
        // use serial or logical name
        YModule *module = [YModule FindModule:serial_or_name];
        if ([module isOnline]) {
            if (argc > 2) {
                if (strcmp(argv[2], "ON") == 0)
                    [module setBeacon:Y_BEACON_ON];
                else
                    [module setBeacon:Y_BEACON_OFF];
            }
            NSLog(@"serial:      %@\n", [module serialNumber]);
            NSLog(@"logical name: %@\n", [module logicalName]);
            NSLog(@"luminosity:  %d\n", [module luminosity]);
            NSLog(@"beacon:      ");
            if ([module beacon] == Y_BEACON_ON)
                NSLog(@"ON\n");
            else
                NSLog(@"OFF\n");
            NSLog(@"upTime:      %ld sec\n", [module upTime] / 1000);
            NSLog(@"USB current: %d mA\n", [module usbCurrent]);
            NSLog(@"logs:       %@\n", [module get_lastLogs]);
        } else {
            NSLog(@"%@ not connected (check identification and USB cable)\n",
                serial_or_name);
        }
        [YAPI FreeAPI];
    }
    return 0;
}

```

Chaque propriété xxx du module peut être lue grâce à une méthode du type `get_xxxx`, et les propriétés qui se sont pas en lecture seule peuvent être modifiées à l'aide de la méthode `set_xxx`: Pour plus de détails concernant ces fonctions utilisées, reportez-vous aux chapitre API



## Modifications des réglages du module

Lorsque que vous souhaitez modifier les réglages d'un module, il suffit d'appeler la fonction `set_xxx`: correspondante, cependant cette modification n'a lieu que dans la mémoire vive du module: si le module redémarre, les modifications seront perdues. Pour qu'elle soient mémorisées de manière persistante, il est nécessaire de demander au module de sauvegarder sa configuration courante dans sa mémoire non volatile. Pour cela il faut utiliser la méthode `saveToFlash`. Inversement il est possible de forcer le module à oublier ses réglages courants en utilisant la méthode `revertFromFlash`. Ce petit exemple ci-dessous vous permet changer le nom logique d'un module.

```
#import <Foundation/Foundation.h>
#import "yocto_api.h"

static void usage(const char *exe)
{
    NSLog(@"usage: %s <serial> <newLogicalName>\n", exe);
    exit(1);
}

int main (int argc, const char * argv[])
{
    NSError *error;

    @autoreleasepool {
        // Setup the API to use local USB devices
        if([YAPI RegisterHub:@"usb" :&error] != YAPI_SUCCESS) {
            NSLog(@"RegisterHub error: %@", [error localizedDescription]);
            return 1;
        }

        if(argc < 2)
            usage(argv[0]);

        NSString *serial_or_name = [NSString stringWithUTF8String:argv[1]];
        // use serial or logical name
        YModule *module = [YModule FindModule:serial_or_name];

        if (module.isOnline) {
            if (argc >= 3) {
                NSString *newname = [NSString stringWithUTF8String:argv[2]];
                if (![YAPI CheckLogicalName:newname]) {
                    NSLog(@"Invalid name (%@)\n", newname);
                    usage(argv[0]);
                }
                module.logicalName = newname;
                [module saveToFlash];
            }
            NSLog(@"Current name: %@\n", module.logicalName);
        } else {
            NSLog(@"%% not connected (check identification and USB cable)\n",
                serial_or_name);
        }
        [YAPI FreeAPI];
    }
    return 0;
}
```

Attention, le nombre de cycles d'écriture de la mémoire non volatile du module est limité. Passé cette limite plus rien ne garantit que la sauvegarde des réglages se passera correctement. Cette limite, liée à la technologie employée par le micro-processeur du module se situe aux alentours de 100000 cycles. Pour résumer vous ne pouvez employer la fonction `saveToFlash` que 100000 fois au cours de la vie du module. Veillez donc à ne pas appeler cette fonction depuis l'intérieur d'une boucle.

## Enumeration des modules

Obtenir la liste des modules connectés se fait à l'aide de la fonction `yFirstModule()` qui renvoie le premier module trouvé, il suffit ensuite d'appeler la fonction `nextModule()` de cet objet pour

trouver les modules suivants, et ce tant que la réponse n'est pas un NULL. Ci-dessous un petit exemple listant les module connectés

```
#import <Foundation/Foundation.h>
#import "yocto_api.h"

int main (int argc, const char * argv[])
{
    NSError *error;

    @autoreleasepool {
        // Setup the API to use local USB devices
        if([YAPI RegisterHub:@"usb" :&error] != YAPI_SUCCESS) {
            NSLog(@"RegisterHub error: %@\n", [error localizedDescription]);
            return 1;
        }

        NSLog(@"Device list:\n");

        YModule *module = [YModule FirstModule];
        while (module != nil) {
            NSLog(@"%@ %@", module.serialNumber, module.productName);
            module = [module nextModule];
        }
        [YAPI FreeAPI];
    }
    return 0;
}
```

## 21.3. Gestion des erreurs

Lorsque vous implémentez un programme qui doit interagir avec des modules USB, vous ne pouvez pas faire abstraction de la gestion des erreurs. Il y aura forcément une occasion où un utilisateur aura débranché le périphérique, soit avant de lancer le programme, soit même en pleine opération. La librairie Yoctopuce est prévue pour vous aider à supporter ce genre de comportements, mais votre code doit néanmoins être fait pour se comporter au mieux pour interpréter les erreurs signalées par la librairie.

La manière la plus simple de contourner le problème est celle que nous avons employé pour les petits exemples précédents de ce chapitre: avant d'accéder à un module, on vérifie qu'il est en ligne avec la méthode `isOnline()` et on suppose ensuite qu'il va y rester pendant la fraction de seconde nécessaire à exécuter les lignes de code suivantes. Ce n'est pas parfait, mais ça peut suffire dans certains cas. Il faut toutefois être conscient qu'on ne peut pas totalement exclure une erreur se produisant après le `isOnline()`, qui pourrait faire planter le programme. La seule manière de l'éviter est d'implémenter une des deux techniques de gestion des erreurs décrites ci-dessous.

La méthode recommandée par la plupart des langages de programmation pour la gestion des erreurs imprévisibles est l'utilisation d'exceptions. C'est le comportement par défaut de la librairie Yoctopuce. Si une erreur se produit alors qu'on essaie d'accéder à un module, la librairie va lancer une exception. Dans ce cas, de trois choses l'une:

- Si votre code attrape l'exception au vol et la gère, et tout se passe bien.
- Si votre programme tourne dans le debugger, vous pourrez relativement facilement déterminer où le problème s'est produit, et voir le message explicatif lié à l'exception.
- Sinon... l'exception va crasher votre programme, boum!

Comme cette dernière situation n'est pas la plus souhaitable, la librairie Yoctopuce offre une autre alternative pour la gestion des erreurs, permettant de faire un programme robuste sans devoir attraper les exceptions à chaque ligne de code. Il suffit d'appeler la fonction `YAPI.DisableExceptions()` pour commuter la librairie dans un mode où les exceptions de chaque fonction sont systématiquement remplacées par des valeurs de retour particulières, qui peuvent être testées par l'appelant lorsque c'est pertinent. Le nom de la valeur de retour en cas d'erreur pour chaque fonction est systématiquement documenté dans la référence de la librairie. Il suit toujours la même logique: une méthode `get_state()` retournera une valeur

NomDeClasse.STATE\_INVALID, une méthode `get_currentValue` retournera une valeur `NomDeClasse.CURRENTVALUE_INVALID`, etc. Dans tous les cas, la valeur retournée sera du type attendu, et ne sera pas un pointeur nul qui risquerait de faire crasher votre programme. Au pire, si vous affichez la valeur sans la tester, elle sera hors du cadre attendu pour la valeur retournée. Dans le cas de fonctions qui ne retournent à priori pas d'information, la valeur de retour sera `YAPI.SUCCESS` si tout va bien, et un code d'erreur différent en cas d'échec.

Quand vous travaillez sans les exceptions, il est possible d'obtenir un code d'erreur et un message expliquant l'origine de l'erreur en le demandant à l'objet qui a retourné une erreur à l'aide des méthodes `errType()` et `errMessage()`. Ce sont les mêmes informations qui auraient été associées à l'exception si elles avaient été actives.



## 22. Utilisation avec des langages non supportés

Les modules Yoctopuce peuvent être contrôlés depuis la plupart des langages de programmation courants. De nouveaux langages sont ajoutés régulièrement en fonction de l'intérêt exprimé par les utilisateurs de produits Yoctopuce. Cependant, certains langages ne sont pas et ne seront jamais supportés par Yoctopuce, les raisons peuvent être diverses: compilateurs plus disponibles, environnements inadaptés, etc...

Il existe cependant des méthodes alternatives pour accéder à des modules Yoctopuce depuis un langage de programmation non supporté.

### 22.1. Utilisation en ligne de commande

Le moyen le plus simple pour contrôler des modules Yoctopuce depuis un langage non supporté consiste à utiliser l'API en ligne de commande à travers des appels système. L'API en ligne de commande se présente en effet sous la forme d'un ensemble de petits exécutables qu'il est facile d'appeler et dont la sortie est facile à analyser. La plupart des langages de programmation permettant d'effectuer des appels système, cela permet de résoudre le problème en quelques lignes.

Cependant, si l'API en ligne de commande est la solution la plus facile, ce n'est pas la plus rapide ni la plus efficace. A chaque appel, l'exécutable devra initialiser sa propre API et faire l'inventaire des modules USB connectés. Il faut compter environ une seconde par appel.

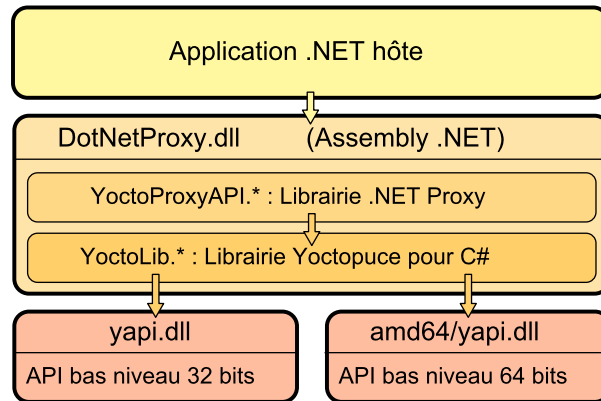
### 22.2. Assembly .NET

Un Assembly .NET permet de partager un ensemble de classes précompilées pour offrir un service, en annonçant des points d'entrées qui peuvent être utilisés par des applications tierces. Dans notre cas, c'est toute la librairie Yoctopuce qui est disponible dans l'Assembly .NET, de sorte à pouvoir être utilisée dans n'importe quel environnement qui supporte le chargement dynamique d'Assembly .NET.

La librairie Yoctopuce sous forme d'Assembly .NET ne contient pas uniquement la librairie Yoctopuce standard pour C#, car cela n'aurait pas permis une utilisation optimale dans tous les environnements. En effet, on ne peut pas attendre forcément des applications hôtes d'offrir un système de threads ou de callbacks, pourtant très utiles pour la gestion du plug-and-play et des capteurs à taux de rafraîchissements élevé. De même, on ne peut pas attendre des applications externes un comportement transparent dans le cas où un appel de fonction dans l'Assembly cause un délai en raison de communication réseau.

Nous y avons donc ajouté une surcouche, appelée librairie *.NET Proxy*. Cette surcouche offre une interface très similaire à la librairie standard mais un peu simplifiée, car elle gère en interne tous les

mécanismes de callbacks. A la place, cette librairie offre des objets miroirs, appelés *Proxys*, qui publient par le biais de *Propriétés* les principaux attributs des fonctions Yoctopuce tels que la mesure courante, les paramètres de configuration, l'état, etc.



Architecture de l'Assembly .NET

Les propriétés des objets *Proxys* sont automatiquement mises à jour en tâche de fond par le mécanisme de callbacks, sans que l'application hôte n'ait à s'en soucier. Celle-ci peut donc à tout moment et sans aucun risque de latence afficher la valeur de toutes les propriétés des objets *Proxys* Yoctopuce.

Notez bien que la librairie de communication de bas niveau `yapi.dll` n'est **pas** incluse dans l'Assembly .NET. Il faut donc bien penser à la garder toujours avec `DotNetProxyLibrary.dll`. La version 32 bits doit être dans le même répertoire que `DotNetProxyLibrary.dll`, tandis que la version 64 bits doit être dans un sous-répertoire nommé `amd64`.

## Exemple d'utilisation avec MATLAB

Voici comment charger notre Assembly .NET Proxy dans MATLAB et lire la valeur du premier capteur branché par USB trouvé sur la machine :

```

NET.addAssembly("C:/Yoctopuce/DotNetProxyLibrary.dll");
import YoctoProxyAPI.*

errmsg = YAPIProxy.RegisterHub("usb");
sensor = YSensorProxy.FindSensor("");
measure = sprintf('%0.3f %s', sensor.CurrentValue, sensor.Unit);
  
```

## Exemple d'utilisation en PowerShell

Les commandes en PowerShell sont un peu plus étranges, mais on reconnaît le même schéma :

```

Add-Type -Path "C:/Yoctopuce/DotNetProxyLibrary.dll"

$errmsg = [YoctoProxyAPI.YAPIProxy]::RegisterHub("usb")
$sensor = [YoctoProxyAPI.YSensorProxy]::FindSensor("")
$measure = "{0:n3} {1}" -f $sensor.CurrentValue, $sensor.Unit
  
```

## Particularités de la librairie .NET Proxy

Par rapport aux librairies Yoctopuce classiques, on notera en particulier les différences suivantes.

### Pas de méthode `FirstModule/nextModule`

Pour obtenir un objet se référant au premier module trouvé, on appelle un `YModuleProxy.FindModule("")`. Si aucun module n'est connecté, cette méthode retournera un objet avec la propriété `module.IsOnline` à `False`. Dès le branchement d'un module, la propriété passera à `True` et l'identifiant matériel du module sera mis à jour.

Pour énumérer les modules, on peut appeler la méthode `module.GetSimilarFunctions()` qui retourne un tableau de chaînes de caractères contenant les identifiants de tous les modules trouvés.

### Pas de fonctions de callback

Les fonctions de callback sont implémentées en interne et mettent à jour les propriétés des objets. Vous pouvez donc simplement faire du polling sur les propriétés, sans pénalité significative de performance. Prenez garde au fait que si vous utilisez l'une des méthodes qui désactive les callbacks, le rafraîchissement automatique des propriétés des objets en sera altéré.

Une nouvelle méthode `YAPIProxy.GetLog` permet de récupérer les logs de diagnostics de bas niveau sans recourir à l'utilisation de callbacks.

### Types énumérés

Pour maximiser la compatibilité avec les applications hôte, la librairie .NET Proxy n'utilise pas de véritables types énumérés .NET, mais des simples entiers. Pour chaque type énuméré, la librairie publie des constantes entières nommées correspondant aux valeurs possibles. Contrairement aux librairies Yoctopuce classiques, les valeurs utiles commencent toujours à 1, la valeur 0 étant réservée pour signifier une valeur invalide, par exemple lorsque le module est débranché.

### Valeurs numériques invalides

Pour toutes les grandeurs numériques, plutôt qu'une constante arbitraire, la valeur invalide retournée en cas d'erreur est `NaN`. Il faut donc utiliser la fonction `isNaN()` pour détecter cette valeur.

## Utilisation de l'Assembly .NET sans la librairie Proxy

Si pour une raison ou une autre vous ne désirez pas utiliser la librairie Proxy, et que votre environnement le permet, vous pouvez utiliser l'API C# standard puisqu'elle se trouve dans l'Assembly, sous le namespace `YoctoLib`. Attention toutefois à ne pas mélanger les deux utilisations: soit vous passez par la librairie Proxy, soit vous utilisez directement la version `YoctoLib`, mais pas les deux !

### Compatibilité

Pour que la librairie *.NET Proxy* fonctionne correctement avec vos modules Yoctopuce, ces derniers doivent avoir au moins le firmware 37120.

Afin d'être compatible avec un maximum de version de Windows, y compris Windows XP, la librairie *DotNetProxyLibrary.dll* est compilée en .NET 3.5, qui est disponible par défaut sur toutes les versions de Windows depuis XP. A ce jour nous n'avons pas trouvé d'environnement hormis Windows qui supporte le chargement d'Assemblies, donc seules les dll de bas niveau pour Windows sont distribuées avec l'Assembly.

## 22.3. Virtual Hub et HTTP GET

Le *Virtual Hub* est disponible pour presque toutes les plateformes actuelles, il sert généralement de passerelle pour permettre l'accès aux modules Yoctopuce depuis des langages qui interdisent l'accès direct aux couches matérielles d'un ordinateur (Javascript, PHP, Java...).

Il se trouve que le *Virtual Hub* est en fait un petit serveur Web qui est capable de router des requêtes HTTP vers les modules Yoctopuce. Ce qui signifie que si vous pouvez faire une requête HTTP depuis votre langage de programmation, vous pouvez contrôler des modules Yoctopuce, même si ce langage n'est pas officiellement supporté.

### Interface REST

A bas niveau, les modules sont pilotés à l'aide d'une API REST. Ainsi pour contrôler un module, il suffit de faire les requêtes HTTP appropriées sur le *Virtual Hub*. Par défaut le port HTTP du *Virtual Hub* est 4444.

Un des gros avantages de cette technique est que les tests préliminaires sont très faciles à mettre en œuvre, il suffit d'un *Virtual Hub* et d'un simple browser Web. Ainsi, si vous copiez l'URL suivante dans votre browser favori, alors que le *Virtual Hub* est en train de tourner, vous obtiendrez la liste des modules présents.

```
http://127.0.0.1:4444/api/services/whitePages.txt
```

Remarquez que le résultat est présenté sous forme texte, mais en demandant *whitePages.xml* vous auriez obtenu le résultat en XML. De même, *whitePages.json* aurait permis d'obtenir le résultat en JSON. L'extension *html* vous permet même d'afficher une interface sommaire vous permettant de changer les valeurs en direct. Toute l'API REST est disponible dans ces différents formats.

## Contrôle d'un module par l'interface REST

Chaque module Yoctopuce a sa propre interface REST disponible sous différentes formes. Imaginons un Yocto-Display avec le numéro de de série *YD128X32-12345* et le nom logique *monModule*. L'URL suivante permettra de connaître l'état du module.

```
http://127.0.0.1:4444/bySerial/YD128X32-12345/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/YD128X32-12345/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/YD128X32-12345/api/module?luminosity=100
```

## Contrôle des différentes fonctions du module par l'interface REST

Les fonctionnalités des modules se manipulent de la même manière. Pour connaître l'état de la fonction *display*, il suffit de construire l'URL suivante.

```
http://127.0.0.1:4444/bySerial/YD128X32-12345/api/display.txt
```

En revanche, si vous pouvez utiliser le nom logique du module en lieu et place de son numéro de série, vous ne pouvez pas utiliser les noms logiques des fonctions, seuls les noms hardware sont autorisés pour les fonctions.

Vous pouvez retrouver un attribut d'une fonction d'un module d'une manière assez similaire à celle utilisée avec les modules, par exemple:

```
http://127.0.0.1:4444/bySerial/YD128X32-12345/api/display/logicalName
```

Assez logiquement, les attributs peuvent être modifiés de la même manière.

```
http://127.0.0.1:4444/bySerial/YD128X32-12345/api/display?logicalName=maFonction
```

Vous trouverez la liste des attributs disponibles pour votre Yocto-Display 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/YD128X32-12345/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/YD128X32-12345/dataLogger.json?id=display&utc=1389801080
```

## 22.4. Utilisation des bibliothèques dynamiques

L'API Yoctopuce bas niveau est disponible sous différents formats de bibliothèque dynamiques écrites en C, dont les sources sont disponibles avec l'API C++. Utiliser une de ces bibliothèques bas niveau vous permettra de vous passer du *Virtual Hub*.

Filename	Plateforme
libyapi.dylib	Max OS X
libyapi-amd64.so	Linux Intel (64 bits)
libyapi-armel.so	Linux ARM EL (32 bits)
libyapi-armhf.so	Linux ARM HL (32 bits)
libyapi-aarch64.so	Linux ARM (64 bits)
libyapi-i386.so	Linux Intel (32 bits)
yapi64.dll	Windows (64 bits)
yapi.dll	Windows (32 bits)

Ces bibliothèques dynamiques contiennent toutes les fonctionnalités nécessaires pour reconstruire entièrement toute l'API haut niveau dans n'importe quel langage capable d'intégrer ces bibliothèques. Ce chapitre se limite cependant à décrire une utilisation de base des modules.

### Contrôle d'un module

Les trois fonctions essentielles de l'API bas niveau sont les suivantes:

```
int yapiInitAPI(int connection_type, char *errmsg);
int yapiUpdateDeviceList(int forceupdate, char *errmsg);
int yapiHTTPRequest(char *device, char *request, char* buffer, int bufsize, int *fullsize,
char *errmsg);
```

La fonction *yapiInitAPI* permet d'initialiser l'API et doit être appelée une fois en début du programme. Pour une connexion de type USB, le paramètre *connection\_type* doit prendre la valeur 1. *errmsg* est un pointeur sur un buffer de 255 caractères destiné à récupérer un éventuel message d'erreur. Ce pointeur peut être aussi mis à *NULL*. La fonction retourne un entier négatif en cas d'erreur, ou zéro dans le cas contraire.

La fonction *yapiUpdateDeviceList* gère l'inventaire des modules Yoctopuce connectés, elle doit être appelée au moins une fois. Pour pouvoir gérer le hot plug, et détecter d'éventuels nouveaux modules connectés, cette fonction devra être appelée à intervalles réguliers. Le paramètre *forceupdate* devra être à la valeur 1 pour forcer un scan matériel. Le paramètre *errmsg* devra pointer sur un buffer de 255 caractères pour récupérer un éventuel message d'erreur. Ce pointeur peut aussi être à *null*. Cette fonction retourne un entier négatif en cas d'erreur, ou zéro dans le cas contraire.

Enfin, la fonction *yapiHTTPRequest* permet d'envoyer des requêtes HTTP à l'API REST du module. Le paramètre *device* devra contenir le numéro de série ou le nom logique du module que vous cherchez à atteindre. Le paramètre *request* doit contenir la requête HTTP complète (y compris les sauts de ligne terminaux). *buffer* doit pointer sur un buffer de caractères suffisamment grand pour

contenir la réponse. *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 = 'YD128X32-12345';
    getValue = 'GET /api/module/luminosity HTTP/1.1'#13#10#13#10;
    setValue = 'GET /api/module?luminosity=100 HTTP/1.1'#13#10#13#10;

begin
    errmsg := @errmsgBuffer;
    data := @dataBuffer;
    // API initialization
    if(yapiInitAPI(1,errmsg)<0) then
        begin
            writeln(errmsg);
            halt;
        end;

    // forces a device inventory
    if( yapiUpdateDeviceList(1,errmsg)<0) then
        begin
            writeln(errmsg);
            halt;
        end;

    // requests the module luminosity
    if (yapiHTTPRequest(serial,getValue,data,sizeof(dataBuffer),fullsize,errmsg)<0) then
        begin
            writeln(errmsg);
            halt;
        end;

    // searches for the HTTP header end
    p := pos(#13#10#13#10,data);

    // displays the response minus the HTTP header
    writeln(copy(data,p+4,length(data)-p-3));

    // change the luminosity
    if (yapiHTTPRequest(serial,setValue,data,sizeof(dataBuffer),fullsize,errmsg)<0) then
        begin
            writeln(errmsg);
            halt;
        end;
end;
```

```
end;

end.
```

## Inventaire des modules

Pour procéder à l'inventaire des modules Yoctopuce, deux fonctions de la librairie dynamique sont nécessaires

```
int yapiGetAllDevices(int *buffer, int maxsize, int *neededsized, char *errmsg);
int yapiGetDeviceInfo(int devdesc, yDeviceSt *infos, char *errmsg);
```

La fonction `yapiGetAllDevices` permet d'obtenir la liste des modules connectés sous la forme d'une liste de handles. `buffer` pointe sur un tableau d'entiers 32 bits qui contiendra les handles retournés. `Maxsize` est la taille en bytes du buffer. `neededsized` contiendra au retour la taille nécessaire pour stocker tous les handles. Cela permet d'en déduire le nombre de module connectés, ou si le buffer passé en entrée est trop petit. Le paramètre `errmsg` devra pointer sur un buffer de 255 caractères pour récupérer un éventuel message d'erreur. Ce pointeur peut aussi être à `null`. Cette fonction retourne un entier négatif en cas d'erreur, ou zéro dans le cas contraire.

La fonction `yapiGetDeviceInfo` permet de récupérer les informations relatives à un module à partir de son handle. `devdesc` est un entier 32bit qui représente le module, et qui a été obtenu grâce à `yapiGetAllDevices`. `infos` pointe sur une structure de données dans laquelle sera stocké le résultat. Le format de cette structure est le suivant:

Nom	Type	Taille (bytes)	Description
vendorid	int	4	ID USB de Yoctopuce
deviceid	int	4	ID USB du module
devrelease	int	4	Version du module
nbinbterfaces	int	4	Nombre d'interfaces USB utilisée par le module
manufacturer	char[]	20	Yoctopuce (null terminé)
productname	char[]	28	Modèle (null terminé)
serial	char[]	20	Numéro de série (null terminé)
logicalname	char[]	20	Nom logique (null terminé)
firmware	char[]	22	Version du firmware (null terminé)
beacon	byte	1	Etat de la balise de localisation (0/1)

Le paramètre `errmsg` devra pointer sur un buffer de 255 caractères pour récupérer un éventuel message d'erreur.

Voici un programme d'exemple écrit en pascal qui utilise la DLL `yapi.dll` pour lister les modules connectés.

```
// device description structure
type yDeviceSt = packed record
  vendorid      : word;
  deviceid      : word;
  devrelease    : word;
  nbinbterfaces : word;
  manufacturer  : array [0..19] of ansichar;
  productname   : array [0..27] of ansichar;
  serial        : array [0..19] of ansichar;
  logicalname   : array [0..19] of ansichar;
  firmware      : array [0..21] of ansichar;
  beacon        : byte;
end;

// Dll function import
function yapiInitAPI(mode:integer;
  errmsg : pansichar):integer;cdecl;
  external 'yapi.dll' name 'yapiInitAPI';

function yapiUpdateDeviceList(force:integer;errmsg : pansichar):integer;cdecl;
  external 'yapi.dll' name 'yapiUpdateDeviceList';
```

```

function yapiGetAllDevices( buffer:pointer;
                           maxsize:integer;
                           var neededsize:integer;
                           errmsg : pansichar):integer; cdecl;
external 'yapi.dll' name 'yapiGetAllDevices';

function apiGetDeviceInfo(d:integer; var infos:yDeviceSt;
                          errmsg : pansichar):integer; cdecl;
external 'yapi.dll' name 'yapiGetDeviceInfo';

var
  errmsgBuffer : array [0..256] of ansichar;
  dataBuffer   : array [0..127] of integer; // max of 128 USB devices
  errmsg,data  : pansichar;
  neededsize,i : integer;
  devinfos    : yDeviceSt;

begin
  errmsg := @errmsgBuffer;

  // API initialisation
  if(yapiInitAPI(1,errmsg)<0) then
  begin
    writeln(errmsg);
    halt;
  end;

  // forces a device inventory
  if( yapiUpdateDeviceList(1,errmsg)<0) then
  begin
    writeln(errmsg);
    halt;
  end;

  // loads all device handles into dataBuffer
  if yapiGetAllDevices(@dataBuffer,sizeof(dataBuffer),neededsize,errmsg)<0 then
  begin
    writeln(errmsg);
    halt;
  end;

  // gets device info from each handle
  for i:=0 to neededsize div sizeof(integer)-1 do
  begin
    if (apiGetDeviceInfo(dataBuffer[i], devinfos, errmsg)<0) then
    begin
      writeln(errmsg);
      halt;
    end;
    writeln(pansichar(@devinfos.serial)+' ('+pansichar(@devinfos.productname)+'));
  end;

end.

```

## VB6 et yapi.dll

Chaque point d'entrée de la DLL yapi.dll est disponible en deux versions, une classique C-decl, et une seconde compatible avec Visual Basic 6 préfixée avec *vb6\_*.

## 22.5. Port de la librairie haut niveau

Toutes les sources de l'API Yoctopuce étant fournies dans leur intégralité, vous pouvez parfaitement entreprendre le port complet de l'API dans le langage de votre choix. Sachez cependant qu'une grande partie du code source de l'API est généré automatiquement.

Ainsi, il n'est pas nécessaire de porter la totalité de l'API, il suffit de porter le fichier *yocto\_api* et un de ceux correspondant à une fonctionnalité, par exemple *yocto\_relay*. Moyennant un peu de travail supplémentaire, Yoctopuce sera alors en mesure de générer tous les autres fichiers. C'est pourquoi il est fortement recommandé de contacter le support Yoctopuce avant d'entreprendre le port de la librairie Yoctopuce dans un autre langage. Un travail collaboratif sera profitable aux deux parties.

## 23. Programmation avancée

Les chapitres précédents vous ont présenté dans chaque langage disponible les fonctions de programmation de base utilisables avec votre module Yocto-Display. Ce chapitre présente de façon plus générale une utilisation plus avancée de votre module. Les exemples sont donnés dans le langage le plus populaire auprès des clients de Yoctopuce, à savoir C#. Néanmoins, vous trouverez dans les bibliothèques de programmation pour chaque langage des exemples complets illustrant les concepts présentés ici.

Afin de rester le plus concis possible, les exemples donnés dans ce chapitre ne font aucune gestion d'erreur. Ne les copiez pas tels-quels dans une application de production.

### 23.1. Programmation par événements

Les méthodes de gestion des modules Yoctopuce qui vous ont été présentées dans les chapitres précédents sont des fonctions de polling, qui consistent à demander en permanence à l'API si quelque chose a changé. Facile à appréhender, cette technique de programmation n'est pas la plus efficace ni la plus réactive. C'est pourquoi l'API de programmation Yoctopuce propose aussi un modèle de programmation par événements. Cette technique consiste à demander à l'API de signaler elle-même les changements importants dès qu'ils sont détectés. A chaque fois qu'un paramètre clé change, l'API appelle une fonction de callback que vous avez prédéfinie.

#### Détecter l'arrivée et le départ des modules

La gestion du *hot-plug* est importante lorsque l'on travaille avec des modules USB, car tôt ou tard vous serez amené à brancher et débrancher un module après le lancement de votre programme. L'API a été conçue pour gérer l'arrivée et le départ inopinés des modules de manière transparente, mais votre application doit en général en tenir compte si elle veut éviter de prétendre utiliser un module qui a été débranché.

La programmation par événements est particulièrement utile pour détecter les branchements/débranchements de modules. Il est en effet plus simple de se faire signaler les branchements, que de devoir lister en permanence les modules branchés pour en déduire ceux qui sont arrivés et ceux qui sont partis. Pour pouvoir être prévenu dès qu'un module arrive, vous avez besoin de trois morceaux de code.

#### Le callback

Le callback est la fonction qui sera appelée à chaque fois qu'un nouveau module Yoctopuce sera branché. Elle prend en paramètre le module concerné.

```
static void deviceArrival(YModule m)
```

```
{  
    Console.WriteLine("Nouveau module : " + m.get_serialNumber());  
}
```

### L'initialisation

Vous devez ensuite signaler à l'API qu'il faut appeler votre callback quand un nouveau module est branché.

```
YAPI.RegisterDeviceArrivalCallback(deviceArrival);
```

Notez que si des modules sont déjà branchés lorsque le callback est enregistré, le callback sera appelé pour chacun de ces modules déjà branchés.

### Déclenchement des callbacks

Un problème classique de la programmation par callbacks est que ces callbacks peuvent être appelés n'importe quand, y compris à des moments où le programme principal n'est pas prêt à les recevoir, ce qui peut avoir des effets de bords indésirables comme des dead-locks et autres conditions de course. C'est pourquoi dans l'API Yoctopuce, les callbacks d'arrivée/départs de modules ne sont appelés que pendant l'exécution de la fonction `UpdateDeviceList()`. Il vous suffit d'appeler `UpdateDeviceList()` à intervalle régulier depuis un timer ou un thread spécifique pour contrôler précisément quand les appels à ces callbacks auront lieu:

```
// boucle d'attente gérant les callback  
while (true)  
{  
    // callback d'arrivée / départ de modules  
    YAPI.UpdateDeviceList(ref errmsg);  
    // attente non active gérant les autres callbacks  
    YAPI.Sleep(500, ref errmsg);  
}
```

De manière similaire, il est possible d'avoir un callback quand un module est débranché. Vous trouverez un exemple concret démontrant toutes ces techniques dans la librairie de programmation Yoctopuce de chaque langage. L'exemple se trouve dans le répertoire *Examples/Prog-EventBased*.

Attention: dans la plupart des langages, les callbacks doivent être des procédures globales, et non pas des méthodes. Si vous souhaitez que le callback appelle une méthode d'un objet, définissez votre callback sous la forme d'une procédure globale qui ensuite appellera votre méthode.

## 24. Référence de l'API de haut niveau

Ce chapitre résume les fonctions de l'API de haut niveau pour commander votre Yocto-Display. La syntaxe et les types précis peuvent varier d'un langage à l'autre mais, sauf avis contraire toutes sont disponibles dans chaque langage. Pour une information plus précise sur les types des arguments et des valeurs de retour dans un langage donné, veuillez vous référer au fichier de définition pour ce langage (`yocto_api.*` ainsi que les autres fichiers `yocto_*` définissant les interfaces des fonctions).

Dans les langages qui supportent les exceptions, toutes ces fonctions vont par défaut générer des exceptions en cas d'erreur plutôt que de retourner la valeur d'erreur documentée pour chaque fonction, afin de faciliter le débogage. Il est toutefois possible de désactiver l'utilisation d'exceptions à l'aide de la fonction `yDisableExceptions()`, si l'on préfère travailler avec des valeurs de retour d'erreur.

Ce chapitre ne reprend pas en détail les concepts de programmation décrits plus tôt, afin d'offrir une référence plus concise. En cas de doute, n'hésitez pas à retourner au chapitre décrivant en détail de chaque attribut configurable.

## 24.1. La classe YAPI

### Fonctions générales

Ces quelques fonctions générales permettent l'initialisation et la configuration de la librairie Yoctopuce. Dans la plupart des cas, un appel à `yRegisterHub()` suffira en tout et pour tout. Ensuite, vous pourrez appeler la fonction globale `yFind...()` ou `yFirst...()` correspondant à votre module pour pouvoir interagir avec lui.

Pour utiliser les fonctions décrites ici, vous devez inclure:

java	<code>import com.yoctopuce.YoctoAPI.YAPI;</code>
dnp	<code>import YoctoProxyAPI.YAPIProxy</code>
cp	<code>#include "yocto_api_proxy.h"</code>
ml	<code>import YoctoProxyAPI.YAPIProxy"</code>
js	<code>&lt;script type='text/javascript' src='yocto_api.js'&gt;&lt;/script&gt;</code>
cpp	<code>#include "yocto_api.h"</code>
m	<code>#import "yocto_api.h"</code>
pas	<code>uses yocto_api;</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>
php	<code>require_once('yocto_api.php');</code>
ts	<code>in HTML: import { YAPI, YErrorMsg, YModule, YSensor } from '../dist/esm/yocto_api_browser.js';</code> <code>in Node.js: import { YAPI, YErrorMsg, YModule, YSensor } from 'yoctolib-cjs/yocto_api_nodejs.js';</code>
es	<code>in HTML: &lt;script src=" ../lib/yocto_api.js"&gt;&lt;/script&gt;</code> <code>in node.js: require('yoctolib-es2017/yocto_api.js');</code>
vi	<code>YModule.vi</code>

### Fonction globales

#### YAPI.AddUdevRule(force)

Ajoute une règle UDEV qui autorise tous les utilisateurs à accéder aux modules Yoctopuce connectés sur les ports USB.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es`

#### YAPI.CheckLogicalName(name)

Vérifie si un nom donné est valide comme nom logique pour un module ou une fonction.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es`

#### YAPI.ClearHTTPCallbackCacheDir(removeFiles)

Désactive le cache de callback HTTP.

`java` `php`

#### YAPI.DisableExceptions()

Désactive l'utilisation d'exceptions pour la gestion des erreurs.

`cpp` `m` `pas` `vb` `cs` `uwp` `py` `php` `ts` `es`

#### YAPI.EnableExceptions()

Réactive l'utilisation d'exceptions pour la gestion des erreurs.

`cpp` `m` `pas` `vb` `cs` `uwp` `py` `php` `ts` `es`

#### YAPI.EnableUSBHost(osContext)



Cette fonction est utilisée uniquement sous Android.

java

#### YAPI.FreeAPI()

Attends que toutes les communications en cours avec les modules Yoctopuce soient terminées puis libère les ressources utilisées par la librairie Yoctopuce.

cpp m pas vb cs java uwp py php ts es dnp

#### YAPI.GetAPIVersion()

Retourne la version de la librairie Yoctopuce utilisée.

cpp m pas vb cs java uwp py php ts es dnp

#### YAPI.GetCacheValidity()

Retourne la durée de validité des données chargée par la librairie.

cpp m pas vb cs java uwp py php ts es

#### YAPI.GetDeviceListValidity()

Retourne le délai entre chaque énumération forcée des YoctoHubs utilisés.

cpp m pas vb cs java uwp py php ts es

#### YAPI.GetDIArchitecture()

Retourne l'architecture du binaire de la librairie de communication Yoctopuce utilisée.

dnp

#### YAPI.GetDIPath()

Retourne l'emplacement sur le disque des librairies Yoctopuce utilisées.

dnp

#### YAPI.GetLog(lastLogLine)

Récupère les messages de logs de la librairie de communication Yoctopuce.

dnp

#### YAPI.GetNetworkTimeout()

Retourne le délai de connexion réseau pour `yRegisterHub()` et `yUpdateDeviceList()`.

cpp m pas vb cs java uwp py php ts es dnp

#### YAPI.GetTickCount()

Retourne la valeur du compteur monotone de temps (en millisecondes).

cpp m pas vb cs java uwp py php ts es

#### YAPI.HandleEvents(errmsg)

Maintient la communication de la librairie avec les modules Yoctopuce.

cpp m pas vb cs java uwp py php ts es

#### YAPI.InitAPI(mode, errmsg)

Initialise la librairie de programmation de Yoctopuce explicitement.

cpp m pas vb cs java uwp py php ts es

#### YAPI.PreregisterHub(url, errmsg)

Alternative plus tolérante à `yRegisterHub()`.

cpp m pas vb cs java uwp py php ts es dnp

#### YAPI.RegisterDeviceArrivalCallback(arrivalCallback)

Enregistre une fonction de callback qui sera appelée à chaque fois qu'un module est branché.

cpp m pas vb cs java uwp py php ts es

#### YAPI.RegisterDeviceRemovalCallback(removalCallback)

Enregistre une fonction de callback qui sera appelée à chaque fois qu'un module est débranché.

cpp m pas vb cs java uwp py php ts es

#### YAPI.RegisterHub(url, errmsg)

Configure la librairie Yoctopuce pour utiliser les modules connectés sur une machine donnée.

cpp m pas vb cs java uwp py php ts es dnp

#### YAPI.RegisterHubDiscoveryCallback(hubDiscoveryCallback)

Enregistre une fonction de callback qui est appelée chaque fois qu'un hub réseau s'annonce avec un message SSDP.

cpp m pas vb cs java uwp py ts es

#### YAPI.RegisterHubWebsocketCallback(ws, errmsg, authpwd)

Variante de la fonction `yRegisterHub()` destinée à initialiser l'API Yoctopuce sur une session Websocket existante, dans le cadre d'un callback WebSocket entrant.

#### YAPI.RegisterLogFunction(logfun)

Enregistre une fonction de callback qui sera appelée à chaque fois que l'API a quelque chose à dire.

cpp m pas vb cs java uwp py ts es

#### YAPI.SelectArchitecture(arch)

Sélectionne manuellement l'architecture de la librairie dynamique à utiliser pour accéder à USB.

py

#### YAPI.SetCacheValidity(cacheValidityMs)

Change la durée de validité des données chargées par la librairie.

cpp m pas vb cs java uwp py php ts es

#### YAPI.SetDelegate(object)

(Objective-C uniquement) Enregistre un objet délégué qui doit se conformer au protocole `YDeviceHotPlug`.

m

#### YAPI.SetDeviceListValidity(deviceListValidity)

Change le délai entre chaque énumération forcée des YoctoHub utilisés.

cpp m pas vb cs java uwp py php ts es

#### YAPI.SetHTTPCallbackCacheDir(directory)

Active le cache du callback HTTP.

java php

#### YAPI.SetNetworkTimeout(networkMsTimeout)

Change le délai de connexion réseau pour `yRegisterHub()` et `yUpdateDeviceList()`.

cpp m pas vb cs java uwp py php ts es dnp

#### YAPI.SetTimeout(callback, ms\_timeout, args)

Appelle le callback spécifié après un temps d'attente spécifié.

ts es

#### YAPI.SetUSBPacketAckMs(pktAckDelay)

Active la quittance des paquets USB reçus par la librairie Yoctopuce.

java

**YAPI.Sleep(ms\_duration, errmsg)**

Effectue une pause dans l'exécution du programme pour une durée spécifiée.

cpp m pas vb cs java uwp py php ts es

**YAPI.TestHub(url, mtimeout, errmsg)**

Test si un hub est joignable.

cpp m pas vb cs java uwp py php ts es dnp

**YAPI.TriggerHubDiscovery(errmsg)**

Relance une détection des hubs réseau.

cpp m pas vb cs java uwp py ts es

**YAPI.UnregisterHub(url)**

Configure la librairie Yoctopuce pour ne plus utiliser les modules connectés sur une machine préalablement enregistré avec RegisterHub.

cpp m pas vb cs java uwp py php ts es

**YAPI.UpdateDeviceList(errmsg)**

Force une mise-à-jour de la liste des modules Yoctopuce connectés.

cpp m pas vb cs java uwp py php ts es

**YAPI.UpdateDeviceList\_async(callback, context)**

Force une mise-à-jour de la liste des modules Yoctopuce connectés.

## 24.2. La classe YModule

Interface de contrôle des paramètres généraux des modules Yoctopuce

La classe `YModule` est utilisable avec tous les modules USB de Yoctopuce. Elle permet de contrôler les paramètres généraux du module, et d'énumérer les fonctions fournies par chaque module.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code>&lt;script type='text/javascript' src='yocto_api.js'&gt;&lt;/script&gt;</code>
cpp	<code>#include "yocto_api.h"</code>
m	<code>#import "yocto_api.h"</code>
pas	<code>uses yocto_api;</code>
vb	<code>yocto_api.vb</code>
cs	<code>yocto_api.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YModule;</code>
py	<code>from yocto_api import *</code>
php	<code>require_once('yocto_api.php');</code>
ts	<code>in HTML: import { YAPI, YErrorMsg, YModule, YSensor } from '../dist/esm/yocto_api_browser.js'; in Node.js: import { YAPI, YErrorMsg, YModule, YSensor } from 'yoctolib-cjs/yocto_api_nodejs.js';</code>
es	<code>in HTML: &lt;script src="../lib/yocto_api.js"&gt;&lt;/script&gt; in node.js: require('yoctolib-es2017/yocto_api.js');</code>
dnsp	<code>import YoctoProxyAPI.YModuleProxy</code>
cp	<code>#include "yocto_module_proxy.h"</code>
vi	<code>YModule.vi</code>
ml	<code>import YoctoProxyAPI.YModuleProxy"</code>

### Fonction globales

#### `YModule.FindModule(func)`

Permet de retrouver un module d'après son numéro de série ou son nom logique.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnsp`

#### `YModule.FindModuleInContext(yctx, func)`

Permet de retrouver un module d'après un identifiant donné dans un Context YAPI.

`java` `uwp` `ts` `es`

#### `YModule.FirstModule()`

Commence l'énumération des modules accessibles par la librairie.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es`

### Propriétés des objets YModuleProxy

#### `module`→`Beacon` [*modifiable*]

état de la balise de localisation.

`dnsp`

#### `module`→`FirmwareRelease` [*lecture seule*]

Version du logiciel embarqué du module.

`dnsp`

#### `module`→`FunctionId` [*lecture seule*]

Identifiant matériel de la *nième* fonction du module.

	<input type="button" value="dnp"/>
<b>module→HardwareId [lecture seule]</b> Identifiant unique du module.	<input type="button" value="dnp"/>
<b>module→IsOnline [lecture seule]</b> Vérifie si le module est joignable.	<input type="button" value="dnp"/>
<b>module→LogicalName [modifiable]</b> Nom logique du module.	<input type="button" value="dnp"/>
<b>module→Luminosity [modifiable]</b> Luminosité des leds informatives du module (valeur entre 0 et 100).	<input type="button" value="dnp"/>
<b>module→ProductId [lecture seule]</b> Identifiant USB du module, préprogrammé en usine.	<input type="button" value="dnp"/>
<b>module→ProductName [lecture seule]</b> Nom commercial du module, préprogrammé en usine.	<input type="button" value="dnp"/>
<b>module→ProductRelease [lecture seule]</b> Numéro uméro de révision du module hardware, préprogrammé en usine.	<input type="button" value="dnp"/>
<b>module→SerialNumber [lecture seule]</b> Numéro de série du module, préprogrammé en usine.	<input type="button" value="dnp"/>
<b>Méthodes des objets YModule</b>	
<b>module→addFileToHTTPCallback(filename)</b> Ajoute un fichier aux données uploadées lors du prochain callback HTTP.	<input type="button" value="cpp"/> <input type="button" value="m"/> <input type="button" value="pas"/> <input type="button" value="vb"/> <input type="button" value="cs"/> <input type="button" value="java"/> <input type="button" value="uwp"/> <input type="button" value="py"/> <input type="button" value="php"/> <input type="button" value="ts"/> <input type="button" value="es"/> <input type="button" value="dnp"/> <input type="button" value="cmd"/>
<b>module→checkFirmware(path, onlynew)</b> Teste si le fichier byn est valide pour le module.	<input type="button" value="cpp"/> <input type="button" value="m"/> <input type="button" value="pas"/> <input type="button" value="vb"/> <input type="button" value="cs"/> <input type="button" value="java"/> <input type="button" value="uwp"/> <input type="button" value="py"/> <input type="button" value="php"/> <input type="button" value="ts"/> <input type="button" value="es"/> <input type="button" value="dnp"/> <input type="button" value="cmd"/>
<b>module→clearCache()</b> Invalide le cache.	<input type="button" value="cpp"/> <input type="button" value="m"/> <input type="button" value="pas"/> <input type="button" value="vb"/> <input type="button" value="cs"/> <input type="button" value="java"/> <input type="button" value="py"/> <input type="button" value="php"/> <input type="button" value="ts"/> <input type="button" value="es"/>
<b>module→describe()</b> Retourne un court texte décrivant le module.	<input type="button" value="cpp"/> <input type="button" value="m"/> <input type="button" value="pas"/> <input type="button" value="vb"/> <input type="button" value="cs"/> <input type="button" value="java"/> <input type="button" value="py"/> <input type="button" value="php"/> <input type="button" value="ts"/> <input type="button" value="es"/>
<b>module→download(pathname)</b> Télécharge le fichier choisi du module et retourne son contenu.	

cpp m pas vb cs java uwp py php ts es dnp cmd

**module**→**functionBaseType(functionIndex)**

Retourne le type de base de la *n*ème fonction du module.

cpp pas vb cs java py php ts es

**module**→**functionCount()**

Retourne le nombre de fonctions (sans compter l'interface "module") existant sur le module.

cpp m pas vb cs java py php ts es

**module**→**functionId(functionIndex)**

Retourne l'identifiant matériel de la *n*ème fonction du module.

cpp m pas vb cs java py php ts es

**module**→**functionName(functionIndex)**

Retourne le nom logique de la *n*ème fonction du module.

cpp m pas vb cs java py php ts es

**module**→**functionType(functionIndex)**

Retourne le type de la *n*ème fonction du module.

cpp pas vb cs java py php ts es

**module**→**functionValue(functionIndex)**

Retourne la valeur publiée par la *n*ème fonction du module.

cpp m pas vb cs java py php ts es

**module**→**get\_allSettings()**

Retourne tous les paramètres de configuration du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module**→**get\_beacon()**

Retourne l'état de la balise de localisation.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module**→**get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

cpp m pas vb cs java py php ts es

**module**→**get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

cpp m pas vb cs java py php ts es

**module**→**get\_firmwareRelease()**

Retourne la version du logiciel embarqué du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module**→**get\_functionIds(funType)**

Retourne les identifiants matériels des fonctions correspondant au type passé en argument.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module**→**get\_hardwareId()**

Retourne l'identifiant unique du module.

cpp m vb cs java py php ts es dnp pas uwp cmd

**module**→**get\_icon2d()**

Retourne l'icône du module.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **module→get\_lastLogs()**

Retourne une chaîne de caractère contenant les derniers logs du module.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **module→get\_logicalName()**

Retourne le nom logique du module.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **module→get\_luminosity()**

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **module→get\_parentHub()**

Retourne le numéro de série du YoctoHub sur lequel est connecté le module.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **module→get\_persistentSettings()**

Retourne l'état courant des réglages persistents du module.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **module→get\_productId()**

Retourne l'identifiant USB du module, préprogrammé en usine.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **module→get\_productName()**

Retourne le nom commercial du module, préprogrammé en usine.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **module→get\_productRelease()**

Retourne le numéro uméro de révision du module hardware, préprogrammé en usine.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **module→get\_rebootCountdown()**

Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **module→get\_serialNumber()**

Retourne le numéro de série du module, préprogrammé en usine.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **module→get\_subDevices()**

Retourne la liste des modules branchés au module courant.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **module→get\_upTime()**

Retourne le nombre de millisecondes écoulées depuis la mise sous tension du module

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **module→get\_url()**

Retourne l'URL utilisée pour accéder au module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get\_usbCurrent()**

Retourne le courant consommé par le module sur le bus USB, en milliampères.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

cpp m pas vb cs java py php ts es

**module→get\_userVar()**

Retourne la valeur entière précédemment stockée dans cet attribut.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→hasFunction(funcId)**

Teste la présence d'une fonction pour le module courant.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→isOnline()**

Vérifie si le module est joignable, sans déclencher d'erreur.

cpp m pas vb cs java py php ts es dnp

**module→isOnline\_async(callback, context)**

Vérifie si le module est joignable, sans déclencher d'erreur.

**module→load(msValidity)**

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

cpp m pas vb cs java py php ts es

**module→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

**module→log(text)**

Ajoute un message arbitraire dans les logs du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→nextModule()**

Continue l'énumération des modules commencée à l'aide de yFirstModule( ).

cpp m pas vb cs java uwp py php ts es

**module→reboot(secBeforeReboot)**

Agende un simple redémarrage du module dans un nombre donné de secondes.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→registerBeaconCallback(callback)**

Enregistre une fonction de callback qui sera appelée à chaque changement d'état de la balise de localisation du module.

cpp m pas vb cs java uwp py php ts es

**module→registerConfigChangeCallback(callback)**

Enregistre une fonction de callback qui sera appelée à chaque fois qu'un réglage persistant d'un module est modifié (par exemple changement d'unité de mesure, etc.)

cpp m pas vb cs java uwp py php ts es

**module→registerLogCallback(callback)**

Enregistre une fonction de callback qui sera appelée à chaque fois le module émet un message de log.



cpp m pas vb cs java uwp py php ts es

#### module→revertFromFlash()

Recharge les réglages stockés dans le mémoire non volatile du module, comme à la mise sous tension du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### module→saveToFlash()

Sauve les réglages courants dans la mémoire non volatile du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### module→set\_allSettings(settings)

Rétablit tous les paramètres du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### module→set\_allSettingsAndFiles(settings)

Rétablit tous les paramètres de configuration et fichiers sur un module.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### module→set\_beacon(newval)

Allume ou éteint la balise de localisation du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### module→set\_logicalName(newval)

Change le nom logique du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### module→set\_luminosity(newval)

Modifie la luminosité des leds informatives du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### module→set\_userData(data)

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userData.

cpp m pas vb cs java py php ts es

#### module→set\_userVar(newval)

Stocke une valeur 32 bits dans la mémoire volatile du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### module→triggerConfigChangeCallback()

Force le déclenchement d'un callback de changement de configuration, afin de vérifier s'ils sont disponibles ou pas.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### module→triggerFirmwareUpdate(secBeforeReboot)

Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### module→updateFirmware(path)

Prepares une mise à jour de firmware du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### module→updateFirmwareEx(path, force)

Prepares une mise à jour de firmware du module.

[cpp](#) [m](#) [pas](#) [vb](#) [cs](#) [java](#) [uwp](#) [py](#) [php](#) [ts](#) [es](#) [dnp](#) [cmd](#)

**module**→**wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

[ts](#) [es](#)

## 24.3. La classe YDisplay

Interface pour interagir avec les écrans, disponibles par exemple dans le Yocto-Display, le Yocto-MaxiDisplay, le Yocto-MaxiDisplay-G et le Yocto-MiniDisplay

La classe `YDisplay` permet de piloter les écrans Yoctopuce. L'interface de contrôle des écrans Yoctopuce est conçue pour afficher facilement des informations et des images. Le module est capable de gérer seul la superposition de plusieurs couches graphiques, qui peuvent être dessinées individuellement, sans affichage immédiat, puis librement positionnées sur l'écran. Il est aussi capable de rejouer des séquences de commandes pré-enregistrées (animations).

Pour afficher du contenu sur l'écran, il faut utiliser la méthode `display.get_displayLayer` pour récupérer la (ou les) couche(s) graphique(s) dans lesquelles vous voulez dessiner, puis écrire dedans à l'aide des méthodes de la classe `YDisplayLayer`.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code>&lt;script type='text/javascript' src='yocto_display.js'&gt;&lt;/script&gt;</code>
cpp	<code>#include "yocto_display.h"</code>
m	<code>#import "yocto_display.h"</code>
pas	<code>uses yocto_display;</code>
vb	<code>yocto_display.vb</code>
cs	<code>yocto_display.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YDisplay;</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YDisplay;</code>
py	<code>from yocto_display import *</code>
php	<code>require_once('yocto_display.php');</code>
ts	<code>in HTML: import { YDisplay } from '../dist/esm/yocto_display.js'; in Node.js: import { YDisplay } from 'yoctolib-cjs/yocto_display.js';</code>
es	<code>in HTML: &lt;script src='../lib/yocto_display.js'&gt;&lt;/script&gt; in node.js: require('yoctolib-es2017/yocto_display.js');</code>
dnsp	<code>import YoctoProxyAPI.YDisplayProxy</code>
cp	<code>#include "yocto_display_proxy.h"</code>
vi	<code>YDisplay.vi</code>
ml	<code>import YoctoProxyAPI.YDisplayProxy</code>

### Fonction globales

#### `YDisplay.FindDisplay(func)`

Permet de retrouver un écran d'après un identifiant donné.

cpp m pas vb cs java uwp py php ts es dnsp

#### `YDisplay.FindDisplayInContext(yctx, func)`

Permet de retrouver un écran d'après un identifiant donné dans un Context YAPI.

java uwp ts es

#### `YDisplay.FirstDisplay()`

Commence l'énumération des écrans accessibles par la librairie.

cpp m pas vb cs java uwp py php ts es

#### `YDisplay.FirstDisplayInContext(yctx)`

Commence l'énumération des écrans accessibles par la librairie.

java uwp ts es

**YDisplay.GetSimilarFunctions()**

Enumère toutes les fonctions de type Display disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

dnp

### Propriétés des objets YDisplayProxy

**display→AdvertisedValue [lecture seule]**

Courte chaîne de caractères représentant l'état courant de la fonction.

dnp

**display→Brightness [modifiable]**

Luminosité des leds informatives du module (valeur entre 0 et 100).

dnp

**display→DisplayHeight [lecture seule]**

Hauteur de l'écran, en pixels.

dnp

**display→DisplayType [lecture seule]**

Type de l'écran: monochrome, niveaux de gris ou couleur.

dnp

**display→DisplayWidth [lecture seule]**

Largeur de l'écran, en pixels.

dnp

**display→FriendlyName [lecture seule]**

Identifiant global de la fonction au format NOM\_MODULE . NOM\_FONCTION.

dnp

**display→FunctionId [lecture seule]**

Identifiant matériel de l'écran, sans référence au module.

dnp

**display→HardwareId [lecture seule]**

Identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.

dnp

**display→IsOnline [lecture seule]**

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

dnp

**display→LayerCount [lecture seule]**

Nombre des couches affichables disponibles.

dnp

**display→LayerHeight [lecture seule]**

Hauteur des couches affichables, en pixels.

dnp

**display→LayerWidth [lecture seule]**

Largeur des couches affichables, en pixels.

dnp

**display→LogicalName [modifiable]**

Nom logique de la fonction.

dnf

**display→Orientation [modifiable]**

Orientation sélectionnée pour l'écran.

dnf

**display→SerialNumber [lecture seule]**

Numéro de série du module, préprogrammé en usine.

dnf

**display→StartupSeq [modifiable]**

Nom de la séquence à jouer à la mise sous tension de l'écran.

dnf

### Méthodes des objets YDisplay

**display→clearCache()**

Invalide le cache.

cpp m pas vb cs java py php ts es

**display→copyLayerContent(srcLayerId, dstLayerId)**

Copie le contenu d'une couche d'affichage vers une autre couche.

cpp m pas vb cs java uwp py php ts es dnf cmd

**display→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'écran au format `TYPE (NAME) =SERIAL.FUNCTIONID`.

cpp m pas vb cs java py php ts es

**display→fade(brightness, duration)**

Change la luminosité de l'écran en douceur, pour produire un effet de fade-in ou fade-out.

cpp m pas vb cs java uwp py php ts es dnf cmd

**display→get\_advertisedValue()**

Retourne la valeur courante de l'écran (pas plus de 6 caractères).

cpp m pas vb cs java uwp py php ts es dnf cmd

**display→get\_brightness()**

Retourne la luminosité des leds informatives du module (valeur entre 0 et 100).

cpp m pas vb cs java uwp py php ts es dnf cmd

**display→get\_displayHeight()**

Retourne la hauteur de l'écran, en pixels.

cpp m pas vb cs java uwp py php ts es dnf cmd

**display→get\_displayLayer(layerId)**

Retourne un objet YDisplayLayer utilisable pour dessiner sur la couche d'affichage correspondante.

cpp m pas vb cs java uwp py php ts es dnf

**display→get\_displayType()**

Retourne le type de l'écran: monochrome, niveaux de gris ou couleur.

cpp m pas vb cs java uwp py php ts es dnf cmd

**display→get\_displayWidth()**

Retourne la largeur de l'écran, en pixels.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→get\_enabled()**

Retourne vrai si le l'écran est alimenté, faux sinon.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

cpp m pas vb cs java py php ts es

**display→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'écran.

cpp m pas vb cs java py php ts es

**display→get\_friendlyName()**

Retourne un identifiant global de l'écran au format NOM\_MODULE . NOM\_FONCTION.

cpp m cs java py php ts es dnp

**display→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

cpp m pas vb cs java py php ts es

**display→get\_functionId()**

Retourne l'identifiant matériel de l'écran, sans référence au module.

cpp m vb cs java py php ts es dnp

**display→get\_hardwareId()**

Retourne l'identifiant matériel unique de l'écran au format SERIAL . FUNCTIONID.

cpp m vb cs java py php ts es dnp

**display→get\_layerCount()**

Retourne le nombre des couches affichables disponibles.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→get\_layerHeight()**

Retourne la hauteur des couches affichables, en pixels.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→get\_layerWidth()**

Retourne la largeur des couches affichables, en pixels.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→get\_logicalName()**

Retourne le nom logique de l'écran.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

cpp m pas vb cs java py php ts es dnp

**display→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**display→get\_orientation()**

Retourne l'orientation sélectionnée pour l'écran.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

**display→get\_serialNumber()**

Retourne le numéro de série du module, préprogrammé en usine.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

**display→get\_startupSeq()**

Retourne le nom de la séquence à jouer à la mise sous tension de l'écran.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

**display→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode `set_userData`.

`cpp` `m` `pas` `vb` `cs` `java` `py` `php` `ts` `es`

**display→isOnline()**

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

`cpp` `m` `pas` `vb` `cs` `java` `py` `php` `ts` `es` `dnp`

**display→isOnline\_async(callback, context)**

Vérifie si le module hébergeant l'écran est joignable, sans déclencher d'erreur.

**display→isReadOnly()**

Indique si la fonction est en lecture seule.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

**display→load(msValidity)**

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

`cpp` `m` `pas` `vb` `cs` `java` `py` `php` `ts` `es`

**display→loadAttribute(attrName)**

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp`

**display→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes de l'écran, avec une durée de validité spécifiée.

**display→muteValueCallbacks()**

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

**display→newSequence()**

Enclenche l'enregistrement de toutes les commandes d'affichage suivantes dans une séquence, qui pourra être rejouée ultérieurement.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

**display→nextDisplay()**

Continue l'énumération des écrans commencée à l'aide de `yFirstDisplay()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les écrans sont retournés.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es`

**display→pauseSequence(delay\_ms)**

Attend pour la durée spécifiée (en millisecondes) avant de jouer les commandes suivantes de la séquence active.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→playSequence(sequenceName)**

Joue une séquence d'affichage préalablement enregistrée à l'aide des méthodes newSequence ( ) et saveSequence ( ).

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

cpp m pas vb cs java uwp py php ts es

**display→resetAll()**

Efface le contenu de l'écran et remet toutes les couches à leur état initial.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→saveSequence(sequenceName)**

Termine l'enregistrement d'une séquence et la sauvegarde sur la mémoire interne de l'écran, sous le nom choisi.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→set\_brightness(newval)**

Modifie la luminosité de l'écran.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→set\_enabled(newval)**

Modifie l'état d'activité de l'écran.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→set\_logicalName(newval)**

Modifie le nom logique de l'écran.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→set\_orientation(newval)**

Modifie l'orientation de l'écran.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→set\_startupSeq(newval)**

Modifie le nom de la séquence à jouer à la mise sous tension de l'écran.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userData.

cpp m pas vb cs java py php ts es

**display→stopSequence(sequenceName)**

Arrête immédiatement la séquence d'affichage actuellement jouée sur l'écran.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→swapLayerContent(layerIdA, layerIdB)**

Permute le contenu de deux couches d'affichage.

cpp m pas vb cs java uwp py php ts es dnp cmd

**display→unmuteValueCallbacks()**

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.



`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### `display`→`upload(pathname, content)`

Télécharge un contenu arbitraire (par exemple une image GIF) vers le système de fichier de l'écran, au chemin d'accès spécifié.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### `display`→`wait_async(callback, context)`

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

`ts` `es`

## 24.4. La classe YDisplayLayer

Interface pour dessiner dans les couches graphiques des écrans, obtenue par l'appel à `display.get_displayLayer`.

Chaque `DisplayLayer` sert à contrôler une couche de contenu affichable (images, texte, etc.). Le contenu n'est visible sur l'écran que lorsque la couche est active sur l'écran (et non masquée par une couche supérieure).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code>&lt;script type='text/javascript' src='yocto_display.js'&gt;&lt;/script&gt;</code>
cpp	<code>#include "yocto_display.h"</code>
m	<code>#import "yocto_display.h"</code>
pas	<code>uses yocto_display;</code>
vb	<code>yocto_display.vb</code>
cs	<code>yocto_display.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YDisplayLayer;</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YDisplayLayer;</code>
py	<code>from yocto_display import *</code>
php	<code>require_once('yocto_display.php');</code>
ts	<code>in HTML: import { YDisplayLayer } from '../dist/esm/yocto_display.js';</code> <code>in Node.js: import { YDisplayLayer } from 'yoctolib-cjs/yocto_display.js';</code>
es	<code>in HTML: &lt;script src=" ../lib/yocto_display.js"&gt;&lt;/script&gt;</code> <code>in node.js: require('yoctolib-es2017/yocto_display.js');</code>
dnp	<code>import YoctoProxyAPI.YDisplayLayerProxy</code>
cp	<code>#include "yocto_display_proxy.h"</code>
ml	<code>import YoctoProxyAPI.YDisplayLayerProxy</code>

### Méthodes des objets YDisplayLayer

#### `displaylayer→clear()`

Efface tout le contenu de la couche de dessin, de sorte à ce qu'elle redevienne entièrement transparente.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### `displaylayer→clearConsole()`

Efface le contenu de la zone de console, et repositionne le curseur de la console en haut à gauche de la zone.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### `displaylayer→consoleOut(text)`

Affiche un message dans la zone de console, et déplace le curseur de la console à la fin du texte.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### `displaylayer→drawBar(x1, y1, x2, y2)`

Dessine un rectangle plein à une position spécifiée.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### `displaylayer→drawBitmap(x, y, w, bitmap, bgcol)`

Dessine un bitmap à la position spécifiée de la couche.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### `displaylayer→drawCircle(x, y, r)`

Dessine un cercle vide à une position spécifiée.

cpp m pas vb cs java uwp py php ts es dnp cmd

**displaylayer→drawDisc(x, y, r)**

Dessine un disque plein à une position spécifiée.

cpp m pas vb cs java uwp py php ts es dnp cmd

**displaylayer→drawImage(x, y, imagename)**

Dessine une image GIF à la position spécifiée de la couche.

cpp m pas vb cs java uwp py php ts es dnp cmd

**displaylayer→drawPixel(x, y)**

Dessine un pixel unique à une position spécifiée.

cpp m pas vb cs java uwp py php ts es dnp cmd

**displaylayer→drawRect(x1, y1, x2, y2)**

Dessine un rectangle vide à une position spécifiée.

cpp m pas vb cs java uwp py php ts es dnp cmd

**displaylayer→drawText(x, y, anchor, text)**

Affiche un texte à la position spécifiée de la couche.

cpp m pas vb cs java uwp py php ts es dnp cmd

**displaylayer→get\_display()**

Retourne l'YDisplay parent.

cpp m pas vb cs java uwp py php ts es dnp

**displaylayer→get\_displayHeight()**

Retourne la hauteur de l'écran, en pixels.

cpp m pas vb cs java uwp py php ts es dnp cmd

**displaylayer→get\_displayWidth()**

Retourne la largeur de l'écran, en pixels.

cpp m pas vb cs java uwp py php ts es dnp cmd

**displaylayer→get\_layerHeight()**

Retourne la hauteur des couches affichables, en pixels.

cpp m pas vb cs java uwp py php ts es dnp cmd

**displaylayer→get\_layerWidth()**

Retourne la largeur des couches affichables, en pixels.

cpp m pas vb cs java uwp py php ts es dnp cmd

**displaylayer→hide()**

Cache la couche de dessin.

cpp m pas vb cs java uwp py php ts es dnp cmd

**displaylayer→lineTo(x, y)**

Dessine une ligne depuis le point de dessin courant jusqu'à la position spécifiée.

cpp m pas vb cs java uwp py php ts es dnp cmd

**displaylayer→moveTo(x, y)**

Déplace le point de dessin courant de cette couche à la position spécifiée.

cpp m pas vb cs java uwp py php ts es dnp cmd

**displaylayer→reset()**

Remet la couche de dessin dans son état initial (entièrement transparente, réglages par défaut).

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **displaylayer**→**selectColorPen(color)**

Choisit la couleur du crayon à utiliser pour tous les appels suivants aux fonctions de dessin.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **displaylayer**→**selectEraser()**

Choisit une gomme plutôt qu'un crayon pour tous les appels suivants aux fonctions de dessin, à l'exception de copie d'images bitmaps.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **displaylayer**→**selectFont(fontname)**

Sélectionne la police de caractères à utiliser pour les fonctions d'affichage de texte suivantes.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **displaylayer**→**selectGrayPen(graylevel)**

Choisit le niveau de gris à utiliser pour tous les appels suivants aux fonctions de dessin.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **displaylayer**→**setAntialiasingMode(mode)**

Active ou désactive l'anti-aliasing pour tracer les lignes et les cercles.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **displaylayer**→**setConsoleBackground(bgcol)**

Configure la couleur de fond utilisée par la fonction `clearConsole` et par le défilement automatique de la console.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **displaylayer**→**setConsoleMargins(x1, y1, x2, y2)**

Configure les marges d'affichage pour la fonction `consoleOut`.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **displaylayer**→**setConsoleWordWrap(wordwrap)**

Configure le mode de retour à la ligne utilisé par la fonction `consoleOut`.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **displaylayer**→**setLayerPosition(x, y, scrollTime)**

Déplace la position de la couche de dessin par rapport au coin supérieur gauche de l'écran.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **displaylayer**→**unhide()**

Affiche la couche.

cpp m pas vb cs java uwp py php ts es dnp cmd

## 24.5. La classe YAnButton

Interface pour interagir avec les entrées analogiques, disponibles par exemple dans le Yocto-Buzzer, le Yocto-Knob, le Yocto-MaxiBuzzer et le Yocto-MaxiDisplay

La classe `YAnButton` permet d'accéder à une entrée résistive simple. Cela permet aussi bien de mesurer l'état d'un simple bouton que de lire un potentiomètre analogique (résistance variable), comme par exemple un bouton rotatif continu, une poignée de commande de gaz ou un joystick. Le module est capable de se calibrer sur les valeurs minimales et maximales du potentiomètre, et de restituer une valeur calibrée variant proportionnellement avec la position du potentiomètre, indépendamment de sa résistance totale.

Pour utiliser les fonctions décrites ici, vous devez inclure:

es	<b>in HTML:</b> <code>&lt;script src="../../lib/yocto_anbutton.js"&gt;&lt;/script&gt;</code> <b>in node.js:</b> <code>require('yoctolib-es2017/yocto_anbutton.js');</code>
js	<code>&lt;script type='text/javascript' src='yocto_anbutton.js'&gt;&lt;/script&gt;</code>
cpp	<code>#include "yocto_anbutton.h"</code>
m	<code>#import "yocto_anbutton.h"</code>
pas	<code>uses yocto_anbutton;</code>
vb	<code>yocto_anbutton.vb</code>
cs	<code>yocto_anbutton.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YAnButton;</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YAnButton;</code>
py	<code>from yocto_anbutton import *</code>
php	<code>require_once('yocto_anbutton.php');</code>
ts	<b>in HTML:</b> <code>import { YAnButton } from '../../dist/esm/yocto_anbutton.js';</code> <b>in Node.js:</b> <code>import { YAnButton } from 'yoctolib-cjs/yocto_anbutton.js';</code>
dnp	<code>import YoctoProxyAPI.YAnButtonProxy</code>
cp	<code>#include "yocto_anbutton_proxy.h"</code>
vi	<code>YAnButton.vi</code>
ml	<code>import YoctoProxyAPI.YAnButtonProxy</code>

### Fonction globales

#### `YAnButton.FindAnButton(func)`

Permet de retrouver une entrée analogique d'après un identifiant donné.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp`

#### `YAnButton.FindAnButtonInContext(yctx, func)`

Permet de retrouver une entrée analogique d'après un identifiant donné dans un Context YAPI.

`java` `uwp` `ts` `es`

#### `YAnButton.FirstAnButton()`

Commence l'énumération des entrées analogiques accessibles par la librairie.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es`

#### `YAnButton.FirstAnButtonInContext(yctx)`

Commence l'énumération des entrées analogiques accessibles par la librairie.

`java` `uwp` `ts` `es`

#### `YAnButton.GetSimilarFunctions()`

Enumère toutes les fonctions de type `AnButton` disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (`hardwareId`).

dnp

### Propriétés des objets YAnButtonProxy

#### **anbutton**→**AdvertisedValue** [*lecture seule*]

Courte chaîne de caractères représentant l'état courant de la fonction.

dnp

#### **anbutton**→**AnalogCalibration** [*modifiable*]

Permet de savoir si une procédure de calibration est actuellement en cours.

dnp

#### **anbutton**→**CalibratedValue** [*lecture seule*]

Valeur calibrée de l'entrée (entre 0 et 1000 inclus).

dnp

#### **anbutton**→**CalibrationMax** [*modifiable*]

Valeur maximale observée durant la calibration (entre 0 et 4095 inclus).

dnp

#### **anbutton**→**CalibrationMin** [*modifiable*]

Valeur minimale observée durant la calibration (entre 0 et 4095 inclus).

dnp

#### **anbutton**→**FriendlyName** [*lecture seule*]

Identifiant global de la fonction au format NOM\_MODULE . NOM\_FONCTION.

dnp

#### **anbutton**→**FunctionId** [*lecture seule*]

Identifiant matériel de l'entrée analogique, sans référence au module.

dnp

#### **anbutton**→**HardwareId** [*lecture seule*]

Identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.

dnp

#### **anbutton**→**InputType** [*modifiable*]

Type de décodage appliqué à l'entrée (entrée analogique ou entrées binaires multiplexées).

dnp

#### **anbutton**→**IsOnline** [*lecture seule*]

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

dnp

#### **anbutton**→**IsPressed** [*lecture seule*]

Vrai si l'entrée (considérée comme binaire) est active (contact fermé), et faux sinon.

dnp

#### **anbutton**→**LogicalName** [*modifiable*]

Nom logique de la fonction.

dnp

#### **anbutton**→**Sensitivity** [*modifiable*]

Sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks.

dnp

**anbutton**→**SerialNumber** [*lecture seule*]

Numéro de série du module, préprogrammé en usine.

dnp

**Méthodes des objets YAnButton****anbutton**→**clearCache()**

Invalide le cache.

cpp m pas vb cs java py php ts es

**anbutton**→**describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'entrée analogique au format `TYPE (NAME) =SERIAL . FUNCTIONID`.

cpp m pas vb cs java py php ts es

**anbutton**→**get\_advertisedValue()**

Retourne la valeur courante de l'entrée analogique (pas plus de 6 caractères).

cpp m pas vb cs java uwp py php ts es dnp cmd

**anbutton**→**get\_analogCalibration()**

Permet de savoir si une procédure de calibration est actuellement en cours.

cpp m pas vb cs java uwp py php ts es dnp cmd

**anbutton**→**get\_calibratedValue()**

Retourne la valeur calibrée de l'entrée (entre 0 et 1000 inclus).

cpp m pas vb cs java uwp py php ts es dnp cmd

**anbutton**→**get\_calibrationMax()**

Retourne la valeur maximale observée durant la calibration (entre 0 et 4095 inclus).

cpp m pas vb cs java uwp py php ts es dnp cmd

**anbutton**→**get\_calibrationMin()**

Retourne la valeur minimale observée durant la calibration (entre 0 et 4095 inclus).

cpp m pas vb cs java uwp py php ts es dnp cmd

**anbutton**→**get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'entrée analogique.

cpp m pas vb cs java py php ts es

**anbutton**→**get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'entrée analogique.

cpp m pas vb cs java py php ts es

**anbutton**→**get\_friendlyName()**

Retourne un identifiant global de l'entrée analogique au format `NOM_MODULE . NOM_FONCTION`.

cpp m cs java py php ts es dnp

**anbutton**→**get\_functionDescriptor()**

Retourne un identifiant unique de type `YFUN_DESCR` correspondant à la fonction.

cpp m pas vb cs java py php ts es

**anbutton**→**get\_functionId()**

Retourne l'identifiant matériel de l'entrée analogique, sans référence au module.

`cpp` `m` `vb` `cs` `java` `py` `php` `ts` `es` `dnp`

#### **anbutton→get\_hardwareId()**

Retourne l'identifiant matériel unique de l'entrée analogique au format SERIAL . FUNCTIONID.

`cpp` `m` `vb` `cs` `java` `py` `php` `ts` `es` `dnp`

#### **anbutton→get\_inputType()**

Retourne le type de décodage appliqué à l'entrée (entrée analogique ou entrées binaires multiplexées).

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **anbutton→get\_isPressed()**

Retourne vrai si l'entrée (considérée comme binaire) est active (contact fermé), et faux sinon.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **anbutton→get\_lastTimePressed()**

Retourne le temps absolu (nombre de millisecondes) entre la mise sous tension du module et la dernière pression observée du bouton à l'entrée (transition du contact de ouvert à fermé).

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **anbutton→get\_lastTimeReleased()**

Retourne le temps absolu (nombre de millisecondes) entre la mise sous tension du module et le dernier relâchement observée du bouton à l'entrée (transition du contact de fermé à ouvert).

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **anbutton→get\_logicalName()**

Retourne le nom logique de l'entrée analogique.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **anbutton→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

`cpp` `m` `pas` `vb` `cs` `java` `py` `php` `ts` `es` `dnp`

#### **anbutton→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

#### **anbutton→get\_pulseCounter()**

Retourne la valeur du compteur d'impulsions.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **anbutton→get\_pulseTimer()**

Retourne le timer du compteur d'impulsions (ms).

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **anbutton→get\_rawValue()**

Retourne la valeur mesurée de l'entrée telle-quelle (entre 0 et 4095 inclus).

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **anbutton→get\_sensitivity()**

Retourne la sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **anbutton→get\_serialNumber()**

Retourne le numéro de série du module, préprogrammé en usine.



cpp m pas vb cs java uwp py php ts es dnp cmd

### **anbutton**→get\_userData()

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

cpp m pas vb cs java py php ts es

### **anbutton**→isOnline()

Vérifie si le module hébergeant l'entrée analogique est joignable, sans déclencher d'erreur.

cpp m pas vb cs java py php ts es dnp

### **anbutton**→isOnline\_async(callback, context)

Vérifie si le module hébergeant l'entrée analogique est joignable, sans déclencher d'erreur.

### **anbutton**→isReadOnly()

Indique si la fonction est en lecture seule.

cpp m pas vb cs java uwp py php ts es dnp cmd

### **anbutton**→load(msValidity)

Met en cache les valeurs courantes de l'entrée analogique, avec une durée de validité spécifiée.

cpp m pas vb cs java py php ts es

### **anbutton**→loadAttribute(attrName)

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

cpp m pas vb cs java uwp py php ts es dnp

### **anbutton**→load\_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'entrée analogique, avec une durée de validité spécifiée.

### **anbutton**→muteValueCallbacks()

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

cpp m pas vb cs java uwp py php ts es dnp cmd

### **anbutton**→nextAnButton()

Continue l'énumération des entrées analogiques commencée à l'aide de yFirstAnButton( ) Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les entrées analogiques sont retournés.

cpp m pas vb cs java uwp py php ts es

### **anbutton**→registerValueCallback(callback)

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

cpp m pas vb cs java uwp py php ts es

### **anbutton**→resetCounter()

Réinitialise le compteur d'impulsions et son timer.

cpp m pas vb cs java uwp py php ts es dnp cmd

### **anbutton**→set\_analogCalibration(newval)

Enclenche ou déclenche le procédé de calibration.

cpp m pas vb cs java uwp py php ts es dnp cmd

### **anbutton**→set\_calibrationMax(newval)

Modifie la valeur maximale de calibration pour l'entrée (entre 0 et 4095 inclus), sans lancer la calibration automatique.

cpp m pas vb cs java uwp py php ts es dnp cmd

### **anbutton**→set\_calibrationMin(newval)

Modifie la valeur minimale de calibration pour l'entrée (entre 0 et 4095 inclus), sans lancer la calibration automatique.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **anbutton**→**set\_inputType(newval)**

Modifie le type de décodage appliqué à l'entrée (entrée analogique ou entrées binaires multiplexées).

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **anbutton**→**set\_logicalName(newval)**

Modifie le nom logique de l'entrée analogique.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **anbutton**→**set\_sensitivity(newval)**

Modifie la sensibilité pour l'entrée (entre 1 et 1000) pour le déclenchement de callbacks.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **anbutton**→**set\_userData(data)**

Enregistre un contexte libre dans l'attribut `userData` de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

`cpp` `m` `pas` `vb` `cs` `java` `py` `php` `ts` `es`

#### **anbutton**→**unmuteValueCallbacks()**

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp` `cmd`

#### **anbutton**→**wait\_async(callback, context)**

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

`ts` `es`

## 24.6. La classe YFiles

Interface pour interagir avec les systèmes de fichier, disponibles par exemple dans le Yocto-Color-V2, le Yocto-SPI, le YoctoHub-Ethernet et le YoctoHub-GSM-4G

La class YFiles permet d'accéder au système de fichier embarqué sur certains modules Yoctopuce. Le stockage de fichiers permet par exemple de personnaliser un service web (dans le cas d'un module connecté au réseau) ou pour d'ajouter un police de caractères (dans le cas d'un module d'affichage).

Pour utiliser les fonctions décrites ici, vous devez inclure:

js	<code>&lt;script type='text/javascript' src='yocto_files.js'&gt;&lt;/script&gt;</code>
cpp	<code>#include "yocto_files.h"</code>
m	<code>#import "yocto_files.h"</code>
pas	<code>uses yocto_files;</code>
vb	<code>yocto_files.vb</code>
cs	<code>yocto_files.cs</code>
java	<code>import com.yoctopuce.YoctoAPI.YFiles;</code>
uwp	<code>import com.yoctopuce.YoctoAPI.YFiles;</code>
py	<code>from yocto_files import *</code>
php	<code>require_once('yocto_files.php');</code>
ts	<code>in HTML: import { YFiles } from '../dist/esm/yocto_files.js'; in Node.js: import { YFiles } from 'yoctolib-cjs/yocto_files.js';</code>
es	<code>in HTML: &lt;script src='../lib/yocto_files.js'&gt;&lt;/script&gt; in node.js: require('yoctolib-es2017/yocto_files.js');</code>
dnp	<code>import YoctoProxyAPI.YFilesProxy</code>
cp	<code>#include "yocto_files_proxy.h"</code>
vi	<code>YFiles.vi</code>
ml	<code>import YoctoProxyAPI.YFilesProxy</code>

### Fonction globales

#### YFiles.FindFiles(func)

Permet de retrouver un système de fichier d'après un identifiant donné.

cpp m pas vb cs java uwp py php ts es dnp

#### YFiles.FindFilesInContext(yctx, func)

Permet de retrouver un système de fichier d'après un identifiant donné dans un Context YAPI.

java uwp ts es

#### YFiles.FirstFiles()

Commence l'énumération des systèmes de fichier accessibles par la librairie.

cpp m pas vb cs java uwp py php ts es

#### YFiles.FirstFilesInContext(yctx)

Commence l'énumération des systèmes de fichier accessibles par la librairie.

java uwp ts es

#### YFiles.GetSimilarFunctions()

Enumère toutes les fonctions de type Files disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

dnp

### Propriétés des objets YFilesProxy

**files**→**AdvertisedValue** *[lecture seule]*

Courte chaîne de caractères représentant l'état courant de la fonction.

dnf

**files**→**FilesCount** *[lecture seule]*

Nombre de fichiers présents dans le système de fichier.

dnf

**files**→**FriendlyName** *[lecture seule]*

Identifiant global de la fonction au format NOM\_MODULE . NOM\_FONCTION.

dnf

**files**→**FunctionId** *[lecture seule]*

Identifiant matériel du système de fichier, sans référence au module.

dnf

**files**→**HardwareId** *[lecture seule]*

Identifiant matériel unique de la fonction au format SERIAL . FUNCTIONID.

dnf

**files**→**IsOnline** *[lecture seule]*

Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

dnf

**files**→**LogicalName** *[modifiable]*

Nom logique de la fonction.

dnf

**files**→**SerialNumber** *[lecture seule]*

Numéro de série du module, préprogrammé en usine.

dnf

### Méthodes des objets YFiles

**files**→**clearCache()**

Invalide le cache.

cpp m pas vb cs java py php ts es

**files**→**describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format TYPE (NAME) =SERIAL . FUNCTIONID.

cpp m pas vb cs java py php ts es

**files**→**download(pathname)**

Télécharge le fichier choisi du filesystem et retourne son contenu.

cpp m pas vb cs java uwp py php ts es dnf cmd

**files**→**download\_async(pathname, callback, context)**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

**files**→**fileExist(filename)**

Test si un fichier existe dans le système de fichier du module.

cpp m pas vb cs java uwp py php ts es dnf cmd

**files**→**format\_fs()**

Rétabli le système de fichier dans on état original, défragmenté.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **files→get\_advertisedValue()**

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **files→get\_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

cpp m pas vb cs java py php ts es

#### **files→get\_errorType()**

Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

cpp m pas vb cs java py php ts es

#### **files→get\_filesCount()**

Retourne le nombre de fichiers présents dans le système de fichier.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **files→get\_freeSpace()**

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **files→get\_friendlyName()**

Retourne un identifiant global du système de fichier au format NOM\_MODULE . NOM\_FONCTION.

cpp m cs java py php ts es dnp

#### **files→get\_functionDescriptor()**

Retourne un identifiant unique de type YFUN\_DESCR correspondant à la fonction.

cpp m pas vb cs java py php ts es

#### **files→get\_functionId()**

Retourne l'identifiant matériel du système de fichier, sans référence au module.

cpp m vb cs java py php ts es dnp

#### **files→get\_hardwareId()**

Retourne l'identifiant matériel unique du système de fichier au format SERIAL . FUNCTIONID.

cpp m vb cs java py php ts es dnp

#### **files→get\_list(pattern)**

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **files→get\_logicalName()**

Retourne le nom logique du système de fichier.

cpp m pas vb cs java uwp py php ts es dnp cmd

#### **files→get\_module()**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

cpp m pas vb cs java py php ts es dnp

#### **files→get\_module\_async(callback, context)**

Retourne l'objet YModule correspondant au module Yoctopuce qui héberge la fonction.

**files→get\_serialNumber()**

Retourne le numéro de série du module, préprogrammé en usine.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→get\_userData()**

Retourne le contenu de l'attribut userData, précédemment stocké à l'aide de la méthode set\_userData.

cpp m pas vb cs java py php ts es

**files→isOnline()**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

cpp m pas vb cs java py php ts es dnp

**files→isOnline\_async(callback, context)**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

**files→isReadOnly()**

Indique si la fonction est en lecture seule.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→load(msValidity)**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

cpp m pas vb cs java py php ts es

**files→loadAttribute(attrName)**

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

cpp m pas vb cs java uwp py php ts es dnp

**files→load\_async(msValidity, callback, context)**

Met en cache les valeurs courantes du système de fichier, avec une durée de validité spécifiée.

**files→muteValueCallbacks()**

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→nextFiles()**

Continue l'énumération des systèmes de fichier commencée à l'aide de yFirstFiles() Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les systèmes de fichier sont retournés.

cpp m pas vb cs java uwp py php ts es

**files→registerValueCallback(callback)**

Enregistre la fonction de callback qui est appelée à chaque changement de la valeur publiée.

cpp m pas vb cs java uwp py php ts es

**files→remove(pathname)**

Efface un fichier, spécifié par son path complet, du système de fichier.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→set\_logicalName(newval)**

Modifie le nom logique du système de fichier.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→set\_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode get\_userData.

`cpp` `m` `pas` `vb` `cs` `java` `py` `php` `ts` `es`

#### `files`→`unmuteValueCallbacks()`

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnf` `cmd`

#### `files`→`upload(pathname, content)`

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnf` `cmd`

#### `files`→`wait_async(callback, context)`

Attend que toutes les commandes asynchrones en cours d'exécution sur le module soient terminées, et appelle le callback passé en paramètre.

`ts` `es`





## 25. Problèmes courants

### 25.1. Par où commencer ?

Si c'est la première fois que vous utilisez un module Yoctopuce et ne savez pas trop par où commencer, allez donc jeter un coup d'œil sur le blog de Yoctopuce. Il y a une section dédiée aux débutants <sup>1</sup>.

### 25.2. Linux et USB

Pour fonctionner correctement sous Linux, la librairie a besoin d'avoir accès en écriture à tous les périphériques USB Yoctopuce. Or, par défaut, sous Linux les droits d'accès des utilisateurs non-root à USB sont limités à la lecture. Afin d'éviter de devoir lancer les exécutable en tant que root, il faut créer une nouvelle règle *udev* pour autoriser un ou plusieurs utilisateurs à accéder en écriture aux périphériques Yoctopuce.

Pour ajouter une règle *udev* à votre installation, il faut ajouter un fichier avec un nom au format "`##-nomArbitraire.rules`" dans le répertoire `/etc/udev/rules.d`. Lors du démarrage du système, *udev* va lire tous les fichiers avec l'extension `.rules` de ce répertoire en respectant l'ordre alphabétique (par exemple, le fichier `51-custom.rules` sera interprété APRES le fichier `50-udev-default.rules`).

Le fichier `50-udev-default` contient les règles *udev* par défaut du système. Pour modifier le comportement par défaut du système, il faut donc créer un fichier qui commence par un nombre plus grand que 50, qui définira un comportement plus spécifique que le défaut du système. Notez que pour ajouter une règle vous aurez besoin d'avoir un accès root sur le système.

Dans le répertoire `udev_conf` de l'archive de VirtualHub<sup>2</sup> pour Linux, vous trouverez deux exemples de règles qui vous éviteront de devoir partir de rien.

#### Exemple 1: 51-yoctopuce.rules

Cette règle va autoriser tous les utilisateurs à accéder en lecture et en écriture aux périphériques Yoctopuce USB. Les droits d'accès pour tous les autres périphériques ne seront pas modifiés. Si ce scénario vous convient, il suffit de copier le fichier `51-yoctopuce_all.rules` dans le répertoire `/etc/udev/rules.d` et de redémarrer votre système.

---

<sup>1</sup> voir: [http://www.yoctopuce.com/FR/blog\\_by\\_categories/pour-les-debutants](http://www.yoctopuce.com/FR/blog_by_categories/pour-les-debutants)

<sup>2</sup> <http://www.yoctopuce.com/EN/virtualhub.php>

```
# udev rules to allow write access to all users
# for Yoctopuce USB devices
SUBSYSTEM=="usb", ATTR{idVendor}=="24e0", MODE="0666"
```

### Exemple 2: 51-yoctopuce\_group.rules

Cette règle va autoriser le groupe "yoctogroup" à accéder en lecture et écriture aux périphériques Yoctopuce USB. Les droits d'accès pour tous les autres périphériques ne seront pas modifiés. Si ce scénario vous convient, il suffit de copier le fichier "51-yoctopuce\_group.rules" dans le répertoire "/etc/udev/rules.d" et de redémarrer votre système.

```
# udev rules to allow write access to all users of "yoctogroup"
# for Yoctopuce USB devices
SUBSYSTEM=="usb", ATTR{idVendor}=="24e0", MODE="0664", GROUP="yoctogroup"
```

## 25.3. Plateformes ARM: HF et EL

Sur ARM il existe deux grandes familles d'exécutables: HF (Hard Float) et EL (EABI Little Endian). Ces deux familles ne sont absolument pas compatibles entre elles. La capacité d'une machine ARM à faire tourner des exécutables de l'une ou l'autre de ces familles dépend du hardware et du système d'exploitation. Les problèmes de compatibilité entre 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 appartient, essayez simplement de lancer un exécutable de chaque famille.

## 25.4. Les exemples de programmation n'ont pas l'air de marcher

La plupart des exemples de programmation de l'API Yoctopuce sont des programmes en ligne de commande et ont besoin de quelques paramètres pour fonctionner. Vous devez les lancer depuis l'invite de commande de votre système d'exploitation ou configurer votre IDE pour qu'il passe les paramètres corrects au programme <sup>3</sup>.

## 25.5. Module alimenté mais invisible pour l'OS

Si votre Yocto-Display est branché par USB et que sa LED bleue s'allume, mais que le module n'est pas vu par le système d'exploitation, vérifiez que vous utilisez bien un vrai câble USB avec les fils pour les données, et non pas un câble de charge. Les câbles de charge n'ont que les fils d'alimentation.

## 25.6. Another process named xxx is already using yAPI

Si lors de l'initialisation de l'API Yoctopuce, vous obtenez le message d'erreur "*Another process named xxx is already using yAPI*", cela signifie qu'une autre application est déjà en train d'utiliser les modules Yoctopuce USB. Sur une même machine, un seul processus à la fois peut accéder aux modules Yoctopuce par USB. Cette limitation peut facilement être contournée en utilisant un VirtualHub et le mode réseau <sup>4</sup>.

<sup>3</sup> voir: <http://www.yoctopuce.com/FR/article/a-propos-des-programmes-d-exemples>

<sup>4</sup> voir: <http://www.yoctopuce.com/FR/article/message-d-erreur-another-process-is-already-using-yapi>

## 25.7. Déconnexions, comportement erratique

Si votre Yocto-Display se comporte de manière erratique et/ou se déconnecte du bus USB sans raison apparente, vérifiez qu'il est alimenté correctement. Evitez les câbles d'une longueur supérieure à 2 mètres. Au besoin, intercalez un hub USB alimenté <sup>5 6</sup>.

## 25.8. Le module ne marche plus après une mise à jour ratée

Si une mise à jour du firmware de votre Yocto-Display échoue, il est possible que le module ne soit plus en état de fonctionner. Si c'est le cas, branchez votre module en maintenant le bouton Yocto-Bouton pressé. La Yocto-LED devrait s'allumer en haute intensité et rester fixe. Relâchez le bouton. Votre Yocto-Display devrait alors apparaître dans le bas de l'interface du virtualHub comme un module attendant une mise à jour de firmware. Cette mise à jour aura aussi pour effet de réinitialiser le module à sa configuration d'usine.

## 25.9. RegisterHub d'une instance de VirtualHub déconnecte la précédente

Si lorsque vous faites un `YAPI.RegisterHub` de VirtualHub et que la connexion avec un autre VirtualHub précédemment enregistré tombe, vérifiez que les machines qui hébergent ces VirtualHubs ont bien un *hostname* différent. Ce cas de figure est très courant avec les machines dont le système d'exploitation est installé avec une image monolithique, comme les Raspberry Pi par exemple. L'API Yoctopuce utilise les numéros de série Yoctopuce pour communiquer et le numéro de série d'un VirtualHub est créé à la volée à partir du *hostname* de la machine qui l'héberge.

## 25.10. Commandes ignorées

Si vous avez l'impression que des commandes envoyées à un module Yoctopuce sont ignorées, typiquement lorsque vous avez écrit un programme qui sert à configurer ce module Yoctopuce et qui envoie donc beaucoup de commandes, vérifiez que vous avez bien mis un `YAPI.FreeAPI()` à la fin du programme. Les commandes sont envoyées aux modules de manière asynchrone grâce à un processus qui tourne en arrière plan. Lorsque le programme se termine, ce processus est tué, même s'il n'a pas eu le temps de tout envoyer. En revanche `API.FreeAPI()` attend que la file d'attente des commandes à envoyer soit vide avant de libérer les ressources utilisées par l'API et rendre la main.

## 25.11. Module endommagé

Yoctopuce s'efforce de réduire la production de déchets électroniques. Si vous avez l'impression que votre Yocto-Display ne fonctionne plus, commencez par contacter le support Yoctopuce par e-mail pour poser un diagnostic. Même si c'est suite à une mauvaise manipulation que le module a été endommagé, il se peut que Yoctopuce puisse le réparer, et ainsi éviter de créer un déchet électronique.



**Déchets d'équipements électriques et électroniques (DEEE)** Si vous voulez vraiment vous débarrasser de votre Yocto-Display, ne le jetez pas à la poubelle, mais ramenez-le à l'un des points de collecte proposé dans votre région afin qu'il soit envoyé à un centre de recyclage ou de traitement spécialisé.

<sup>5</sup> voir: <http://www.yoctopuce.com/FR/article/cables-usb-la-taille-compte>

<sup>6</sup> voir: <http://www.yoctopuce.com/FR/article/combien-de-capteurs-usb-peut-on-connecter>



## 26. Caractéristiques

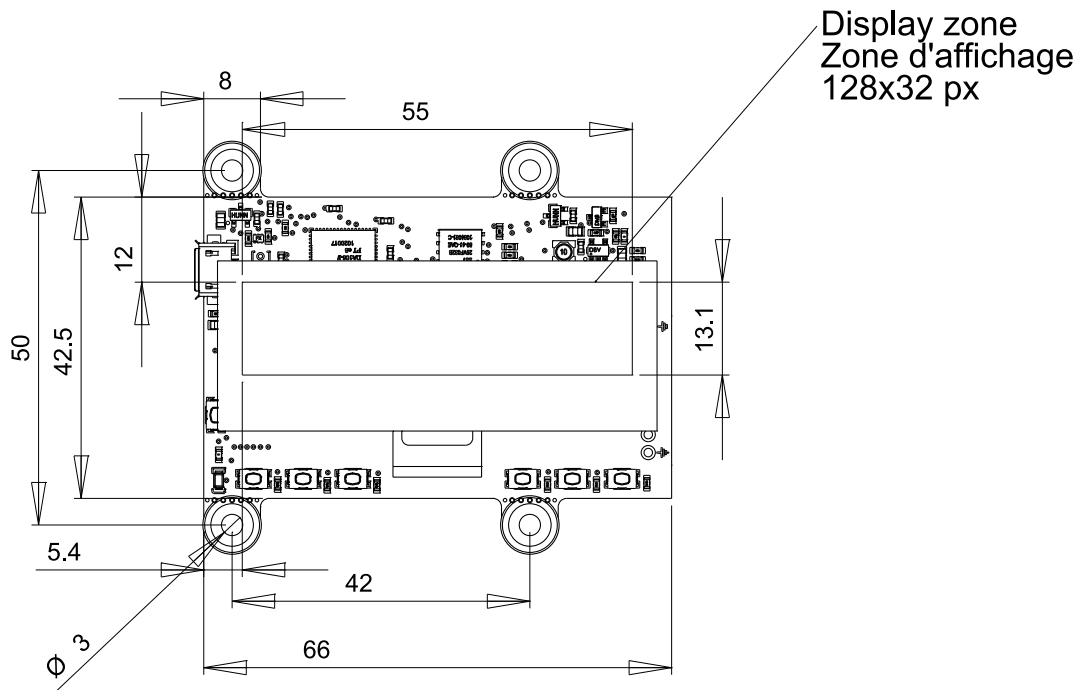
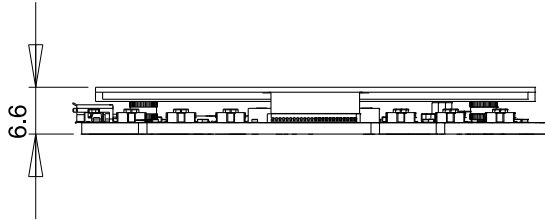
Vous trouverez résumées ci-dessous les principales caractéristiques techniques de votre module Yocto-Display

Identifiant produit	YD128X32
Révision matérielle <sup>†</sup>	
Zone affichable	55 x 13.1 mm
Connecteur USB	micro-B
Largeur	58 mm
Longueur	66 mm
Poids	19 g
Resolution	128 x 32 px
Classe de protection selon IEC 61140	classe III
Temp. de fonctionnement normale	5...40 °C
Temp. de fonctionnement étendue <sup>‡</sup>	-25...70 °C
Conformité RoHS	RoHS III (2011/65/UE+2015/863)
USB Vendor ID	0x24E0
USB Device ID	0x002D
Boîtier recommandé	YoctoBox-Display
Code tarifaire harmonisé	9032.9000
Fabriqué en	Suisse

<sup>†</sup> Ces spécifications correspondent à la révision matérielle actuelle du produit. Les spécifications des versions antérieures peuvent être inférieures.

<sup>‡</sup> La plage de température étendue est définie d'après les spécifications des composants et testée sur une durée limitée (1h). En cas d'utilisation prolongée hors de la plage de température standard, il est recommandé procéder à des tests extensifs avant la mise en production.





All dimensions are in mm  
Toutes les dimensions sont en mm

# Yocto-Display

A4

Scale  
1:1  
Echelle