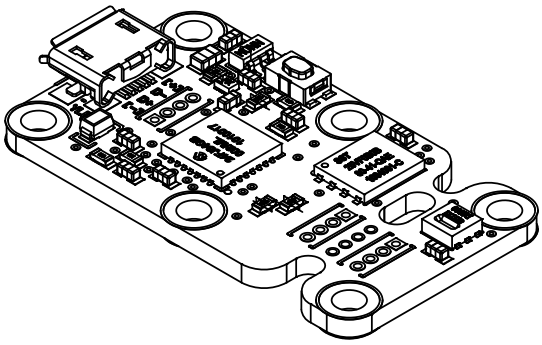
Yocto–Humidity, Mode d'emploi

Table des matières

1. Introduction

Le module Yocto–Humidity est un module de 35.5x20mm qui permet de mesurer par USB à la fois la température et le taux d'humiditée relative. Sa précision est de 0.3°C pour la température et d'environ 4% pour le taux d'humidité.



Le module Yocto–Humidity

Yoctopuce vous remercie d'avoir fait l'acquisition de ce Yocto–Humidity et espère sincèrement qu'il vous donnera entière satisfaction. Les ingénieurs Yoctopuce se sont donnés beaucoup de mal pour que votre Yocto–Humidity soit facile à installer n'importe où et soit facile à piloter depuis un maximum de langages de programmation. Néanmoins, si ce module venait a vous décevoir n'hésitez pas à contacter le support Yoctopuce[1].

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

1.1. Prérequis

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

Un ordinateur

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

Yoctopuce fourni les librairies logicielles permettant de piloter ses modules pour les systèmes d'exploitation suivants: Windows, Mac OS X et Linux. Les modules Yoctopuce ne nécessitent

[1] support@yoctopuce.com

pas l'installation de driver (ou pilote) spécifiques, car ils utilisent le driver HID [2] fourni en standard dans tous les systèmes d'exploitation.

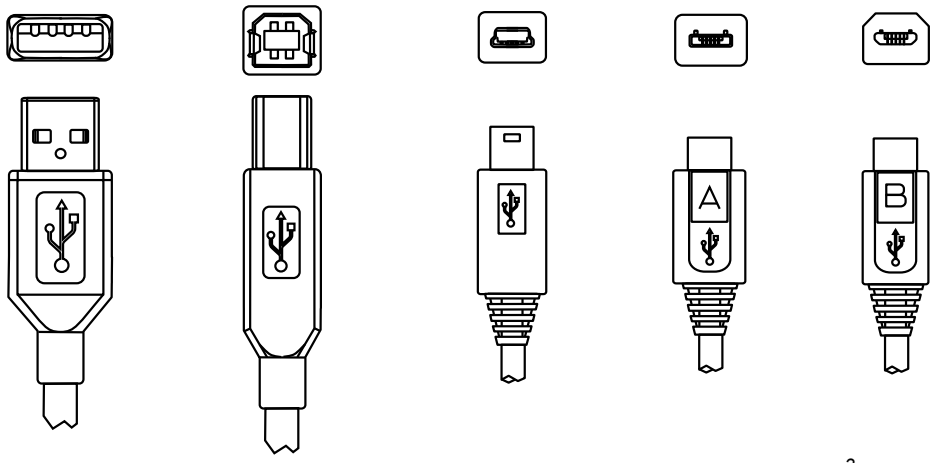
Les versions de Windows actuellement supportées sont Windows XP, Windows 2003, Windows Vista et Windows 7. Les versions 32 bit et 64 bit sont supportées. Yoctopuce teste régulièrement le bon fonctionnement des modules sur Windows XP et Windows 7.

Les versions de Mac OS X actuellement supportées sont Mac OS X 10.5 (Leopard), 10.6 (Snow Leopard) et 10.7 (Lion). Yoctopuce teste régulièrement le bon fonctionnement des modules sur Mac OS X 10.6 et 10.7.

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

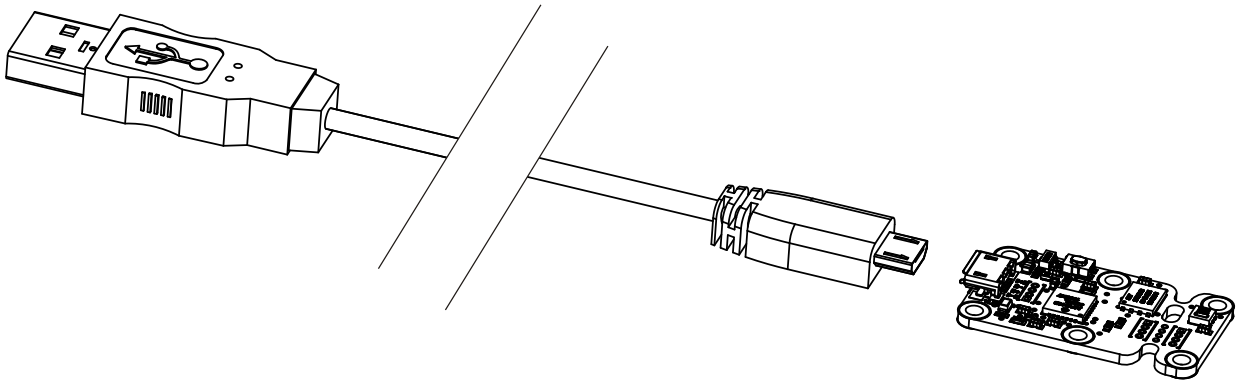
Un cable USB de type A-micro B

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



Les connecteur USB 2 les plus courants: A, B, Mini B, Micro A, Micro B.[3]

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



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

[2] Le driver HID est celui qui gère les périphériques tels que la souris, le clavier, etc.
[3] *Le connecteur Mini A a existé quelque temps, mais a été retiré du standard USB http://www.usb.org/developers/Deprecation_Announcement_052507.pdf*

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

1.2. Accessoires optionnels

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

Micro-hub USB

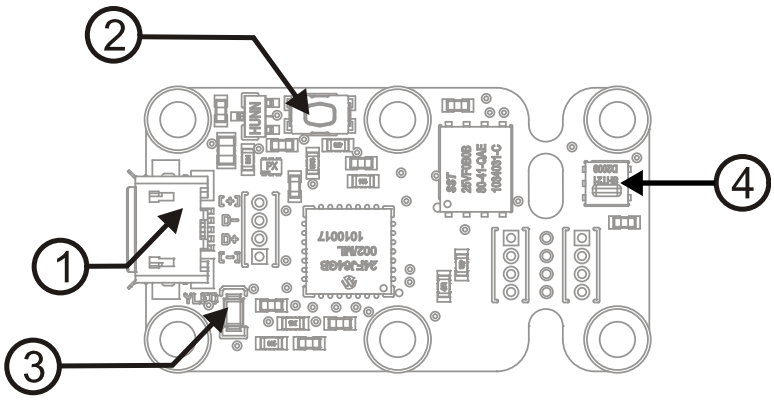
Si vous désirez placer plusieurs modules Yoctopuce dans un espace très restreint, vous pouvez les connecter ensemble à l'aide d'un micro-hub USB. Yoctopuce fabrique des hubs particulièrement petits précisément destinés à cet usage, dont la taille peut être réduite à 20mm par 36mm, et qui se montent en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

Cable nappe mono-brin

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

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

2. Présentation



1: Prise USB micro-B 3: Yocto-led
2: Yocto-bouton 4: Capteur

2.1. Les éléments communs

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

Le connecteur USB

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

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

Le Yocto-bouton

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

La Yocto-Led

En temps normal la Yocto-Led sert à indiquer le bon fonctionnement du module: elle émet alors une faible lumière bleue qui varie lentement mimant ainsi une respiration. La Yocto-Led

cesse de respirer lorsque le module ne communique plus, par exemple si il est alimenté par un hub sans connexion avec un ordinateur allumé.

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

La Yocto-Led a une troisième fonctionnalité moins plaisante: lorsque ce logiciel interne qui contrôle le module rencontre une erreur fatale, elle se met à flasher SOS en morse[4]. 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 contacter le support Yoctopuce[5].

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–Humidity ce numéro commence par HUMSENS1. Le module peut être piloté par logiciel en utilisant ce numéro de série. Ce numéro de série ne peut pas être changé.

Le nom logique

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

2.2. Les éléments spécifiques

Le capteur

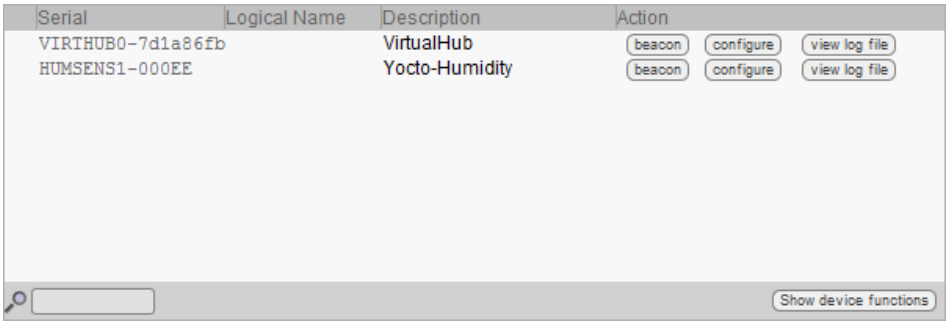
Ce capteur est un SHT21 fabriqué par Sensirion. Il est capable de mesurer avec précision à la fois la température et le taux d'humidité. Ce capteur est relativement délicat, faîte en sorte qu'aucun liquide ne puisse entrer à l'intérrieur, n'utilisez jamais de solvant pour le nettoyer. Si vous comptez tropicaliser votre module, protégez soigneusement le capteur avant, vous trouverez plus de détails sur le site de Sensirion[6]

[4] court-court-court long-long-long court-court-court
[5] support@yoctopuce.com
[6] http://www.sensirion.com/en/pdf/product_information/Handling_Instructions_V1.1_C1.pdf

3. Premiers pas

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

Rendez-vous sur le site de Yoctopuce et téléchargez le programme *Virtual Hub* [7], Il est disponible pour Windows, Linux et Mac OS X. En temps normal le programme Virtual Hub sert de couche d'abstraction pour les langages qui ne peuvent pas accéder aux couches matérielles de votre ordinateur. Mais il offre aussi une interface sommaire pour configurer vos modules et tester les fonctions de base, on accède à cette interface à l'aide d'un simple browser web [8]. Lancez le *Virtual Hub* en ligne de commande, ouvrez votre browser préféré et tapez l'adresse *http:// 127.0.0.1:4444* . Vous devriez voir apparaître la liste des modules Yoctopuce raccordés à votre ordinateur.



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

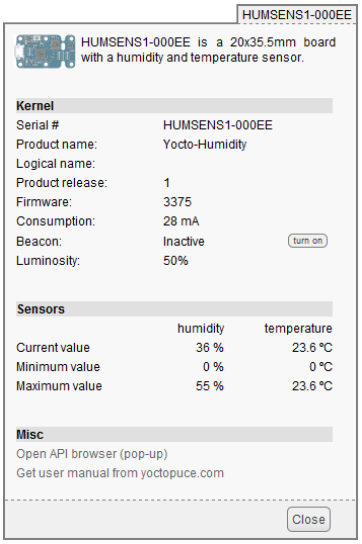
3.1. Localisation

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

3.2. Test du module

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

[7] www.yoctopuce.com/FR/virtualhub.php
[8] L'interface a été testée avec FireFox 3+, IE 6+, Safari et Chrome, elle ne fonctionne pas avec Opéra

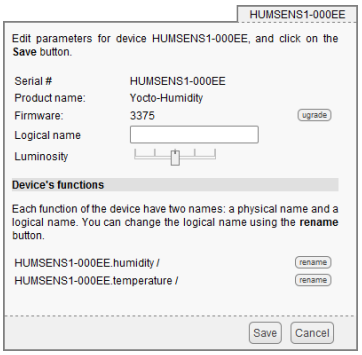


Propriétés du module Yocto–Humidity.

Cette fenêtre vous permet entre autres de jouer avec avec votre module pour en vérifier sont fonctionement, les valeurs de température et d'humidité y sont en effet affichées en temps réel.

3.3. Configuration

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



Configuration du module Yocto–Humidity.

Firmware

Le firmware du module peut être facilement être mis à jour à l'aide de l'interface. Pour ce faire, vous devez au préalable disposer du firmware adéquat sur votre disque local. Les firmware 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, débrancher puis rebrancher le module puis recommencer la procédure devrait résoudre le problème la plupart du temps. Si le module a été débranché alors qu'il était en cours de reprogrammation, il ne fonctionnera probablement plus ne sera plus listé dans l'interface. Mais il sera toujours possible de le reprogrammer correctement en utilisant le programme *Virtual Hub*[9].

Nom logique du module

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

[9] www.yoctopuce.com/FR/virtualhub.php

Luminosité

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

Nom logique des fonctions

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

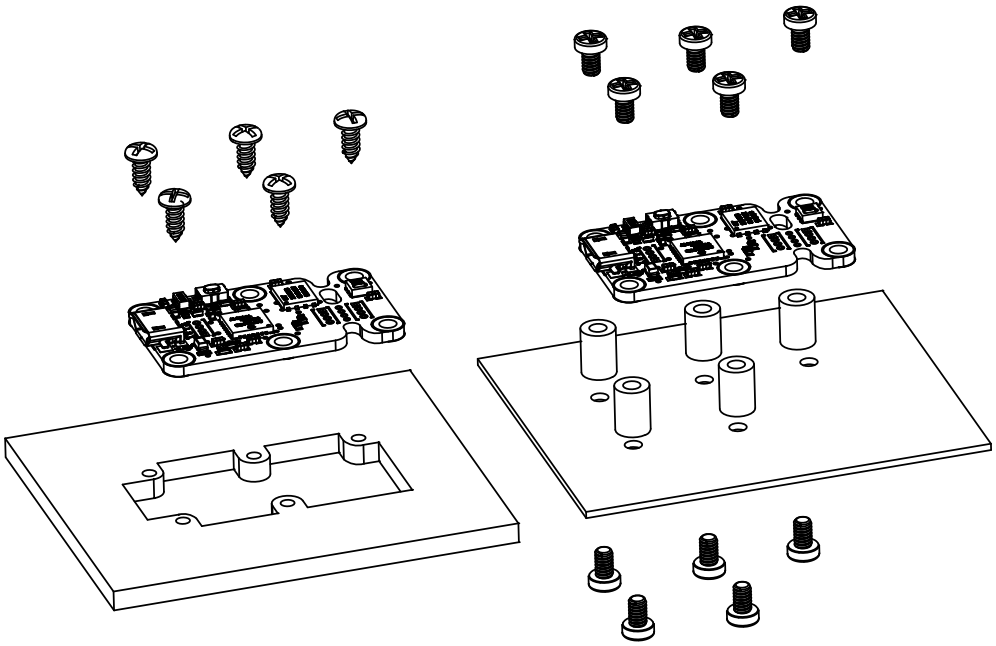
Le module Yocto–Humidity dispose de deux fonction: "humidity" et "température". Cliquez simplement sur le bouton "rename" correspondant our leur affecter un nouveau nom logique.

4. Montage et connectique

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

4.1. Fixation

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



Exemples de montage sur un support.

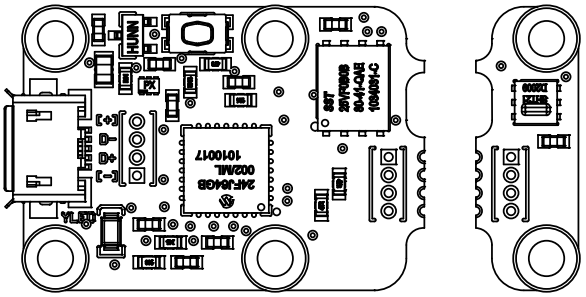
Le module Yocto–Humidity dispose de trous de montage 2.5mm. Vous pouvez utiliser ces trous pour y passer des vis. Le diamètre de la tête de ces vis ne devra pas dépasser 4.5mm, sous peine d'endommager les circuits du module. Veillez à que la surface inférieure du module ne soit pas en contact avec le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

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

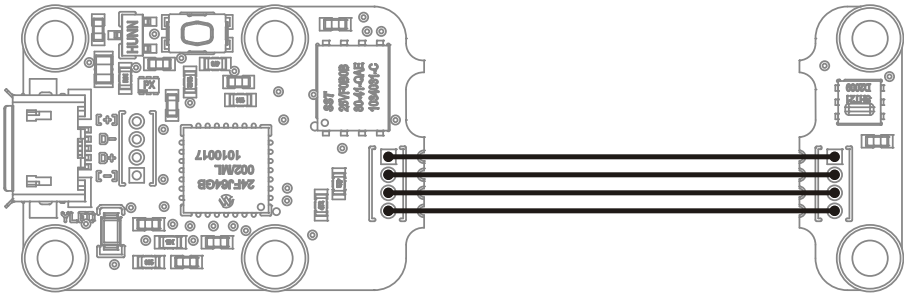
sûr provoquer un court-circuit: il y a des pads à nu sous votre module. Du simple ruban adhésif d'emballage devrait faire l'affaire.

4.2. Déporter le capteur

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

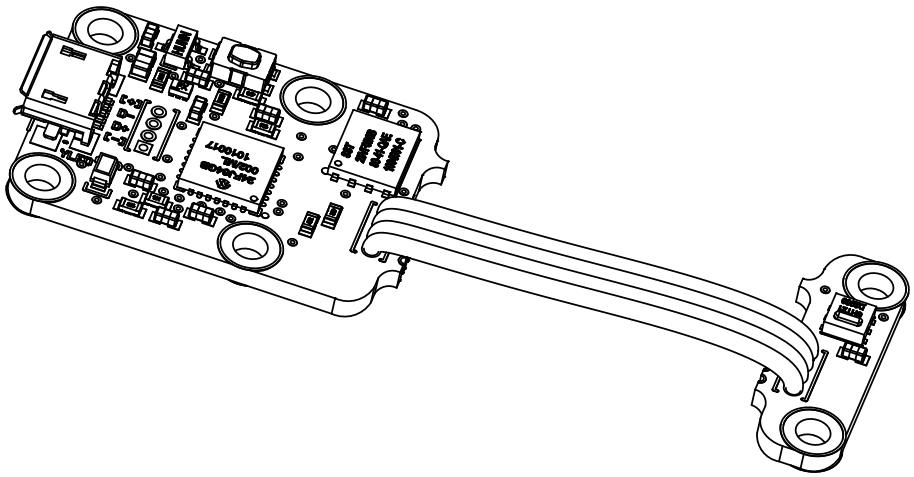


Le module Yocto–Humidity est conçu pour pouvoir être séparé en deux parties.



Câblage des sous module une fois ceux-ci séparés.

Une fois les modules séparés vous allez devoir les recâbler. Plusieurs solutions s'offrent à vous. Vous pouvez raccorder les sous-modules en soudant des fils électriques tout simples, mais vous obtiendrez un meilleur résultat avec du cable nappe au pas 1.27 mm. Utilisez de préférence du cable avec des conducteurs mono-brin plutôt que du multi-brin: les câble mono-brin sont un peu moins souples, mais nettement plus facile à souder.



Déport du capteur à l'aide de câble nappe.

Attention, les modules Yoctopuce sécables ont souvent des systèmes de connectique très semblables. Cependant les sous-modules ne sont pas du tout compatibles entre modèles différents. Si vous raccordez un sous module de votre Yocto–Humidity à un autre type de module, par exemple un Yocto-Temperature, cela ne marchera pas, et vous risquez fort d'endommager votre matériel.

4.3. Contraintes d'alimentation par USB

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

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

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

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

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

5. Programmation, concepts généraux

Chaque module Yoctopuce possède une interface de programmation pour contrôler le coeur du module, et une ou plusieurs interfaces distinctes pour contrôler les fonctions spécifiques au module. Quelque soit le langage de programmation utilisé, l'interaction avec le Yocto–Humidity se fera toujours en lisant ou modifiant les attributs de ces interfaces.

Le module Yocto–Humidity offre une instance de la fonction Temperature, correspondant au capteur de température, et une instance de la fonction Humidity, correspondant à la sonde d'humidité relative. La précision du capteur de température est de 0.3 degrés Celsius, et une celle de la sonde d'humidité est d'environ 2 %RH.

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	Nombre entier	lecture seule
realmHTTP	Texte	modifiable
adminPassword	Texte	modifiable
userPassword	Texte	modifiable
rebootCountdown	Nombre entier	modifiable
usbBandwidth	Type énuméré	modifiable

humidity : Humidity

attribut	type	modifiable ?
logicalName	Texte	modifiable
advertisedValue	Texte	lecture seule
currentValue	Nombre (virgule fixe)	lecture seule
lowestValue	Nombre (virgule fixe)	modifiable
highestValue	Nombre (virgule fixe)	modifiable

temperature : Temperature

attribut	type	modifiable ?
logicalName	Texte	modifiable
advertisedValue	Texte	lecture seule
currentValue	Nombre (virgule fixe)	lecture seule

lowestValue	Nombre (virgule fixe)	modifiable
highestValue	Nombre (virgule fixe)	modifiable

5.1. Interface de contrôle du module

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

productName

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

serialNumber

Chaine de caractères contenant le numéro de série, unique et préprogrammé en usine. Pour un module Yocto–Humidity, ce numéro de série commence toujours par HUMSENS1. Il peut servir comme point de départ pour accéder par programmation à un module particulier.

logicalName

Chaine de caractères contenant le nom logique du module, initialement vide. Cet attribut peut être changé au bon vouloir de l'utilisateur. Un 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 17 en usine.

productRelease

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

firmwareRelease

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

persistentSettings

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

luminosity

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

beacon

Etat de la balise de localisation du module.

upTime

Temps écoulé depuis la dernière mise sous tension du module.

usbCurrent

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

realmHTTP

Nom du domaine d'identification utilisé par le module, lorsqu'on désire sécuriser l'accès au module par un mot de passe. La valeur d'usine est "YoctoDevices".

adminPassword

Mot de passe pour l'utilisateur "admin", autorisé à changer la configuration du module. A la sortie de l'usine, aucun mot de passe n'est exigé pour changer la configuration.

userPassword

Mot de passe pour l'utilisateur "user", exigé pour toute utilisation du module lorsqu'il est configuré. A la sortie de l'usine, aucun mot de passe n'est exigé pour l'utilisation du module.

rebootCountdown

Compte à rebours pour déclencher un redémarrage spontané du module.

usbBandwidth

Nombre d'interfaces utilisé par USB. L'option **DOUBLE** permet de doubler le débit USB mais peut saturer un hub USB. N'oubliez pas d'appeler la méthode `saveToFlash()` et de redémarrer le module pour que le paramètre soit appliqué.

5.2. Interface de la fonction Humidity

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

logicalName

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

advertisedValue

Courte chaîne de caractères résumant l'état actuel du capteur d'humidité, et qui sera publiée automatiquement jusqu'au hub parent. Pour un capteur d'humidité, la valeur publiée est la valeur mesurée (nombre à virgule fixe, avec une décimale).

currentValue

Humidité actuelle, en %RH, sous forme de nombre à virgule.

lowestValue

Humidité minimale observée, sous forme de nombre à virgule.

highestValue

Humidité maximale observée, sous forme de nombre à virgule.

5.3. Interface de la fonction Temperature

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

logicalName

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

advertisedValue

Courte chaîne de caractères résumant l'état actuel du capteur de température, et qui sera publiée automatiquement jusqu'au hub parent. Pour un capteur de température, la valeur publiée est la valeur mesurée (nombre à virgule fixe, avec une décimale).

currentValue

Température actuelle, en degrés Celsius, sous forme de nombre à virgule.

lowestValue

Température minimale observée, sous forme de nombre à virgule.

highestValue

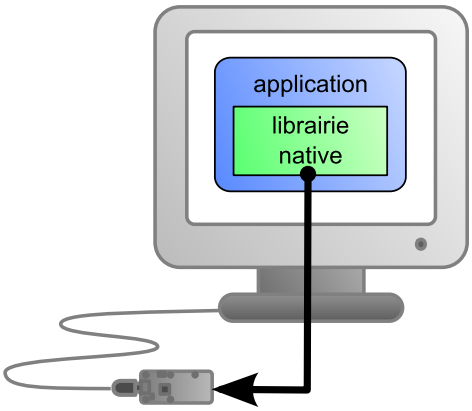
Température maximale observée, sous forme de nombre à virgule.

5.4. Quelle interface: Native, DLL ou Service?

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

Contrôle natif

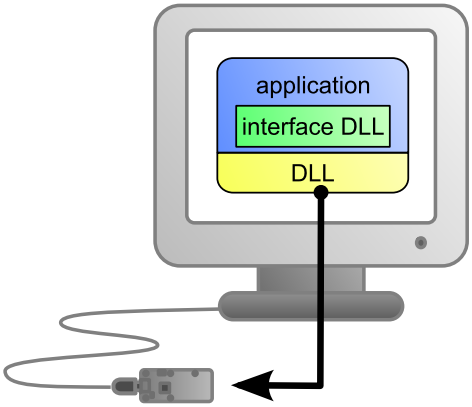
Dans ce cas de figure le programme pilotant votre projet est directement compilé avec une librairie qui offre le contrôle des modules. C'est objectivement la solution la plus simple et la plus élégante pour l'utilisateur final. Il lui suffira de brancher le câble USB et de lancer votre programme pour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



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

Contrôle par DLL

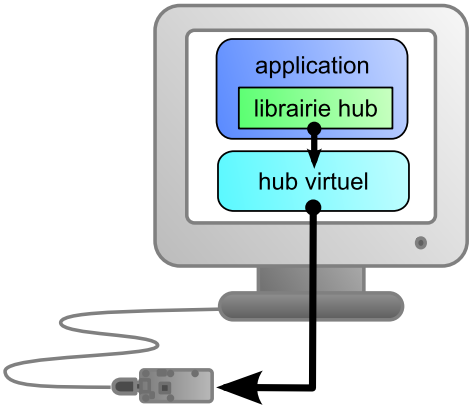
Ici l'essentiel du code permettant de contrôler les modules se trouve dans une DLL, et le programme est compilé avec un petite librairie permettant de contrôler cette DLL. C'est la manière la plus rapide pour coder le support des modules dans un language particulier. En effet la partie "utile" du code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



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

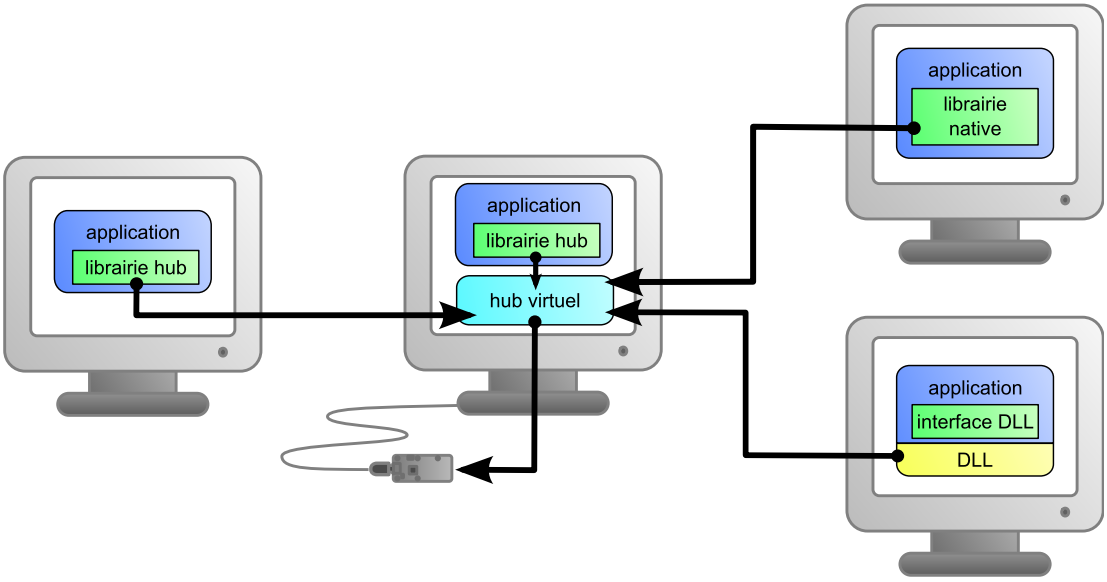
Contrôle par un service

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



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

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



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

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

Language	*Natif*	*DLL*	*Hub virtuel*
C++	•	•	•
Delphi	-	•	•
Javascript	-	-	•
PHP	-	-	•

Méthode de support pour les différents langages.

Limitation des librairies Yoctopuce

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

5.5. Interface haut niveau ou bas niveau ?

Selon vos besoins et vos préférences, il est possible d'utiliser la librairie Yoctopuce avec des fonctions de haut niveau ou des fonctions de bas niveau.

Par fonctions de haut niveau, on entend des fonctions et des objets différenciés par module, dont les méthodes fournissent explicitement accès aux différentes fonctions et attributs.

Par fonctions de bas niveau, on entend on contraire une fonction très générique qui permet un accès au module indépendant de son type, mais qui n'offre aucune abstraction pour accéder aux différentes fonctions et attributs.

Le principal avantage à utiliser les fonctions de haut niveau est qu'elles permettent d'écrire en général du code plus simple, moins sujet aux erreurs [10] . Le prix à payer pour cette simplification du code est de devoir lire la documentation de ces fonctions et classes pour les utiliser. C'est l'information que vous trouverez dans les chapitres suivants.

L'avantage des fonctions de bas niveau est qu'elles permettent aux développeurs expérimentés d'obtenir le résultat désiré en dépendant le moins possible d'une librairie tierce. Dans le cas des modules Yoctopuce, qui implémentent une interface de type REST, il est même possible de se passer entièrement de librairie pour certains types de projet, et de

[10] Un autre avantage des fonctions de haut-niveau de la librairie Yoctopuce est qu'elles permettent d'écrire du code (quasiment) portable d'un langage à un autre, car la librairie Yoctopuce utilise autant que possible les mêmes noms de fonctions, classes et constantes pour tous les langages.

communiquer directement par HTTP avec l'API REST. Vous trouverez plus de détail sur les fonctions de bas niveau et leur utilisation dans une documentation séparée, prochainement disponible sur le site de Yoctopuce.

6. Utilisation du Yocto–Humidity en Javascript

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

Au moment de l'écriture de ce manuel, la librairie Javascript fonctionne avec n'importe quel browser récent... sauf Opera. Il est probable que qu'Opera finira un jour par fonctionner avec la librairie Yoctopuce[11], mais pour l'instant ce n'est pas le cas.

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

6.1. Préparation

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

- La librairie de programmation pour Javascript[12]
- Le programme VirtualHub[13] pour Windows, Mac OS X ou Linux selon l'OS que vous utilisez

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

6.2. Contrôle de la fonction Humidity

Ouvrez votre éditeur de texte préféré[14], recopiez le code ci dessous en remplaçant le numéro de série utilisé par celui de votre module Yocto–Humidity, sauvez-le dans le même répertoire que les fichiers de la librairie, et ouvrez-le avec votre browser favori.

```
<HTML>
<HEAD>
 <TITLE> Hello World</TITLE>
 <SCRIPT type="text/javascript" src="file:yocto_api.js"></SCRIPT>
 <SCRIPT type="text/javascript" src="file:yocto_humidity.js"></SCRIPT>
 <SCRIPT language='javascript1.5'  type='text/JavaScript'>
 <!--
```

[11] En fait dès qu'Opera implémentera le support pour le header HTTP Access-Control-Allow-Origin
[12] www.yoctopuce.com/FR/libraries.php
[13] www.yoctopuce.com/FR/virtualhub.php
[14] Si vous n'avez pas d'éditeur de texte, utilisez Notepad plutôt que Microsoft Word.

```
 yRegisterHub("http://127.0.0.1:4444/");
 var serial  = "HUMSENS1-123456"; // use serial number or logical name
 var hum = yFindHumidity(serial+".humidity");

 function refresh()
 {
     if (hum.isOnline()) {
         document.getElementById("hum-val").value = hum.get_currentValue();
         setTimeout('refresh()',1000);
     } else {
         alert('Module not connected (check identification and USB cable)');
     }
 }
 -->
 </SCRIPT>
</HEAD>
<BODY onload='refresh();'>
Current humidity:    <input id='hum-val' size=2 readonly> %RH<br>
</BODY>
</HTML>
```

L'exemple ci-dessous est codé pour être utilisé directement sur votre machine, sans avoir à passer par un serveur web. Pour le faire fonctionner à travers un serveur web, il vous suffira simplement d'ajuster le chemin d'accès aux librairies Javascript en enlevant le `file:`.

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

yocto_api.js et yocto_humidity.js

Ces deux includes Javascript permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api.js` doit toujours être inclus, `yocto_humidity.js` est nécessaire pour gérer les modules contenant un capteur d'humidité, comme le Yocto-Humidity.

yRegisterHub

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

yFindHumidity

La fonction `yFindHumidity`, permet de retrouver un capteur d'humidité 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-Humidity avec le numéros de série *HUMSENS1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *humidity* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
var humidity = yFindHumidity("HUMSENS1-123456.humidity");
var humidity = yFindHumidity("HUMSENS1-123456.MaFonction");
var humidity = yFindHumidity("MonModule.humidity");
var humidity = yFindHumidity("MonModule.MaFonction");
var humidity = yFindHumidity("MaFonction");
```

`yFindHumidity` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur d'humidité.

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `yFindHumidity` permet d'obtenir l'humidité relative actuelle mesurée par le capteur. La valeur de retour est un nombre flottant, représentant directement le nombre de %RH.

6.3. Contrôle de la partie module

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

```
<HTML>
<HEAD>
 <TITLE> Module Control</TITLE>
 <SCRIPT type="text/javascript" src="file:yocto_api.js"></SCRIPT>

 <SCRIPT language='javascript1.5'  type='text/JavaScript'>
 <!--

 var module = yFindModule("HUMSENS1-123456"); // use serial number or logical name

 yRegisterHub("http://127.0.0.1:4444/");

 function beacon(state)
 {
     module.set_beacon(state);
     refresh();
 }

 function refresh()
 {
     html = 'Module not connected (check identification and USB cable)';
     if (module.isOnline()) {
         html='serial: '+module.get_serialNumber()+'<br>';
         html+='logical name: '+module.get_logicalName()+'<br>';
         html+='luminosity:'+module.get_luminosity()+'%<br>';
         html+='beacon:';
         if (module.get_beacon()==Y_BEACON_ON)
             html+="ON <a href='javascript:beacon(Y_BEACON_OFF)'>switch off</a><br>";
         else
             html     +="OFF <a href='javascript:beacon     (Y_BEACON_ON)'>switch on</
a><br>";
         html+='upTime: '+parseInt(module.get_upTime()/1000)+'sec<br>';
         html+='USB current: '+module.get_usbCurrent()+'mA<br>';
         html+="<a href='javascript:refresh()'>refresh</a>";
     }
     document.body.innerHTML=html;
 }
 -->
 </SCRIPT>
</HEAD>

<BODY></BODY>

<SCRIPT language='javascript1.5'  type='text/JavaScript'>
 <!--
 refresh();
 -->
</SCRIPT>

</HTML>
```

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

Modifications des réglages du module

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

```
<HTML>
<HEAD>
 <TITLE>Change module settings</TITLE>
 <SCRIPT type="text/javascript" src="file:yocto_api.js"></SCRIPT>
```

```
 <SCRIPT language='javascript1.5'  type='text/JavaScript'>
 <!--
 yRegisterHub("http://127.0.0.1:4444/");
 var serial ='HUMSENS1-123456';  //  put your module serial#/logical name here
 var module = yFindModule(serial);
 function save()
 {
     var newname = document.forms['devform'].elements['newlname'].value;
     if (!yCheckLogicalName(newname)) {
         alert('invalid logical name');
         return;
     }
     module.set_logicalName(newname);
     module.saveToFlash();
     refresh();
 }

 function refresh()
 {
     if (module.isOnline()) {
         document.getElementById('curName').innerHTML=module.get_logicalName();
     } else {
         alert('Module not connected (check identification and USB cable)');
     }
 }
 -->
 </SCRIPT>
</HEAD>

<BODY>
<FORM name='devform' onsubmit='save()'>
Current name: <span id='curName'></span><br>
New logical name: <input name='newlname'>
<a href='javascript:save();'>Save</a>
</FORM>
</BODY>
<SCRIPT language='javascript1.5'  type='text/JavaScript'>
 <!--
 refresh();
 -->
</SCRIPT>
</HTML>
```

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

Énumération des modules

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

```
<HTML>
<HEAD>
 <TITLE>Modules inventory</TITLE>
 <SCRIPT type="text/javascript" src="file:yocto_api.js"></SCRIPT>

</HEAD>
<BODY>
<H1>Device list</H1>
<tt><span id='list'></span></tt>
</BODY>
<SCRIPT language='javascript1.5'  type='text/JavaScript'>
 <!--
 yRegisterHub("http://127.0.0.1:4444/");
 var htmlcode = '';
 var module   = yFirstModule();
 while (module!=null) {
     htmlcode=htmlcode + module.get_serialNumber()
                       +'('+module.get_productName()+")<br>";
     module=module.nextModule();
 }
 document.getElementById('list').innerHTML=htmlcode;
 -->
</SCRIPT>
</HTML>
```

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

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

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

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

7. Utilisation du Yocto–Humidity en PHP

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

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

Pour démarrer vos essais en PHP, vous allez avoir besoin d'un serveur PHP 5.x [15] de préférence en local sur votre machine. Si vous souhaiter utiliser celui qui se trouve chez votre provider internet, c'est possible, mais vous devrez probablement configurer votre routeur ADSL pour qu'il accepte et forwarde les requêtes TCP sur le port 4444.

7.1. Préparation

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

- La librairie de programmation pour PHP[16]
- Le programme VirtualHub[17] pour Windows, Mac OS X ou Linux selon l'OS que vous utilisez

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

7.2. Contrôle de la fonction Humidity

Ouvrez votre éditeur de texte préféré[18], recopiez le code ci dessous en remplaçant le numéro de série utilisé par celui de votre module Yocto–Humidity, sauvez-le dans le même répertoire que les fichiers de la librairie, et ouvrez-le avec votre browser favori.

```
<HTML>
<HEAD>
 <TITLE> Hello World</TITLE>
</HEAD>
<BODY>
```

[15] Quelques serveurs PHP gratuits: easyPHP pour windows, MAMP pour Mac Os X
[16] www.yoctopuce.com/FR/libraries.php
[17] www.yoctopuce.com/FR/virtualhub.php
[18] Si vous n'avez pas d'éditeur de texte, utilisez Notepad plutôt que Microsoft Word.

```
<?php
  include('yocto_api.php');
  include('yocto_humidity.php');
  yRegisterHub('http://127.0.0.1:4444/');
  $serial = 'HUMSENS1-123456'; // use serial number or logical name
  $hum = yFindHumidity("$serial.humidity");

  if ($hum->isOnline()) {
      $value = $hum->get_currentValue();
      Print("Humidity: $value %RH<br>");
      // trigger auto-refresh after one second
      Print("<script language='javascript1.5' type='text/JavaScript'>\n");
      Print("setTimeout('window.location.reload()',1000);");
      Print("</script>\n");
  } else {
      Print("Module not connected (check identification and USB cable)<br>");
  }
?>
</BODY>
</HTML>
```

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

yocto_api.php et yocto_humidity.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_humidity.php` est nécessaire pour gérer les modules contenant un capteur d'humidité, comme le Yocto–Humidity.

yRegisterHub

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

yFindHumidity

La fonction `yFindHumidity`, permet de retrouver un capteur d'humidité 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–Humidity avec le numéros de série *HUMSENS1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *humidity* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
$humidity = yFindHumidity("HUMSENS1-123456.humidity");
$humidity = yFindHumidity("HUMSENS1-123456.MaFonction");
$humidity = yFindHumidity("MonModule.humidity");
$humidity = yFindHumidity("MonModule.MaFonction");
$humidity = yFindHumidity("MaFonction");
```

`yFindHumidity` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur d'humidité.

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `yFindHumidity` permet d'obtenir l'humidité relative actuelle mesurée par le capteur. La valeur de retour est un nombre flottant, représentant directement le nombre de %RH.

7.3. Contrôle de la partie module

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

```
<HTML>
<HEAD>
 <TITLE> Module Control</TITLE>
</HEAD>
<BODY>

<?php
    include('yocto_api.php');
    yRegisterHub('http://127.0.0.1:4444/');
    $module = yFindModule('HUMSENS1-123456');  // use serial or logical name

    if ($module->isOnline()) {
        if (isset($_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 ($module->get_beacon()==Y_BEACON_ON)
            printf("ON <a href='?beacon=OFF'>switch off</a><br>");
        else
            printf("OFF <a href='?beacon=ON'>switch on</a><br>");
        printf('upTime: %s sec<br>',intVal($module->get_upTime()/1000));
        printf('USB current: %smA<br>',$module->get_usbCurrent());
    } else {
        print('Module not connected (check identification and USB cable)<br>');
    }
 ?>

<a href='?'>refresh</a>
</BODY>
</HTML>
```

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

Modifications des réglages du module

Lorsque que vous souhaitez modifier les réglages d'un module, il suffit d'appeler la fonction `set_xxx()` correspondante, cependant cette modification n'a lieu que dans la mémoire vive du module: si le module redémarre, les modifications seront perdue. 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 name='myform'  target='<?php Print($_SERVER['PHP_SELF'])?>' method='post'>
<?php
    include('yocto_api.php');

    yRegisterHub("http://127.0.0.1:4444/");

    $serial ='HUMSENS1-123456';  // use serial or logical name
    $module = yFindModule($serial);

    if ($module->isOnline()) {
        if (isset($_POST['newname'])){
            $newname= $_POST['newname'];
            if (!yCheckLogicalName($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>");
        print("<input type='submit' value='Submit'>");
   } else {
        print("Module not connected (check identification and USB cable)");
   }
?>

</FORM>
</BODY>
</HTML>
```

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

Enumération des modules

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

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

7.4. Gestion des erreurs

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

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

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

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

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

8. Utilisation du Yocto–Humidity en C++

Le C++ n'est pas le langage le plus simple à maîtriser. Pourtant, si on prend soin à se limiter aux fonctionnalités essentielles, c'est un langage tout à fait utilisable pour des petits programmes vite faits, et qui a l'avantage d'être très portable d'un système d'exploitation à l'autre. Sous Windows, tous les exemples et les modèles de projet sont testés avec Microsoft Visual Studio 2010 Express, disponible gratuitement sur le site de Microsoft [19]. Sous Mac OS X, tous les exemples et les modèles de projet sont testés avec XCode 4, disponible sur l'App Store. Par ailleurs, aussi bien sous Mac OS X que sous Linux, vous pouvez compiler les exemples en ligne de commande avec GCC en utilisant le `GNUmakefile` fourni. De même, sous Windows, un `Makefile` pour permet de compiler les exemples en ligne de commande, et en pleine connaissance des arguments de compilation et link.

Les librairies Yoctopuce[20] pour C++ vous sont fournies au format source dans leur intégralité. Une partie de la librairie de bas-niveau est écrite en C pur sucre, mais vous n'aurez à priori pas besoin d'interagir directement avec elle: tout a été fait pour que l'interaction soit le plus simple possible depuis le C++. La librairie vous est fournie bien entendu aussi sous forme binaire, de sorte à pouvoir la linker directement si vous le préférez.

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

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

8.1. Contrôle de la fonction Humidity

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

```
#include "yocto_api.h"
#include "yocto_humidity.h"
#include <iostream>

using namespace std;

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

[19] http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express
[20] www.yoctopuce.com/FR/libraries.php

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

    string serial = "HUMSENS1-123456";  // use serial number or logical name
    YHumidity *sensor = yFindHumidity(serial + ".humidity");

    while(1) {
        if(!sensor->isOnline()) {
            cout << "Module not connected (check identification and USB cable)";
            break;
        }

        cout << "Current humidity: " << sensor->get_currentValue() << "%";
        cout << "  (press Ctrl-C to exit)" << endl;
        ySleep(1000,errmsg);
    };

    return 0;
}
```

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

yocto_api.h et yocto_humidity.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_humidity.h` est nécessaire pour gérer les modules contenant un capteur d'humidité, comme le Yocto–Humidity.

yRegisterHub

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

yFindHumidity

La fonction `yFindHumidity`, permet de retrouver un capteur d'humidité 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–Humidity avec le numéros de série *HUMSENS1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *humidity* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
YHumidity *humidity = yFindHumidity("HUMSENS1-123456.humidity");
YHumidity *humidity = yFindHumidity("HUMSENS1-123456.MaFonction");
YHumidity *humidity = yFindHumidity("MonModule.humidity");
YHumidity *humidity = yFindHumidity("MonModule.MaFonction");
YHumidity *humidity = yFindHumidity("MaFonction");
```

`yFindHumidity` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur d'humidité.

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `yFindHumidity` permet d'obtenir l'humidité relative actuelle mesurée par le capteur. La valeur de retour est un nombre flottant, représentant directement le nombre de %RH.

8.2. 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;

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


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

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

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

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

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

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

Modifications des réglages du module

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

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

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

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

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

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

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

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

Enumeration des modules

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

```
#include <iostream>

#include "yocto_api.h"

using namespace std;

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

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

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

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

```
    return 0;
}
```

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

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

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

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

8.4. Intégration de la librairie Yoctopuce en C++

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

Intégration au format source

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

- Elle garanti le respect des conventions de compilation de votre projet (32/64 bits, inclusion des symboles de debug, caractères unicode ou ASCII, etc.);
- Elle facilite le déboggage si vous cherchez la cause d'un problème lié à la librairie Yoctopuce

- Elle réduit les dépendances sur des composants tiers, par exemple pour parer au cas où vous pourriez être mené à recompiler ce projet pour une architecture différente dans de nombreuses années.
- Elle ne requiert pas l'installation d'une librairie dynamique spécifique à Yoctopuce sur le système final, tout est dans l'exécutable.

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

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

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

Intégration en librairie statique

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

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

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

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

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

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

Intégration en librairie dynamique

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

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

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

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

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

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

9. Utilisation du Yocto–Humidity en Delphi

Delphi est l'héritier de Turbo-Pascal. A l'origine, Delphi était produit par Borland, mais c'est maintenant Embarcadero qui l'édite. Sa force réside dans sa facilité d'utilisation, il permet à quiconque ayant des notions de Pascal de programmer une application Windows en deux temps trois mouvements. Son seul défaut est d'être payant[21].

Les librairies pour Delphi sont fournies non pas sous forme de composants VCL, mais directement sous forme de fichiers source. Ces fichiers sont compatibles avec la plupart des version de Delphi [22].

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

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

9.1. Préparation

Connectez-vous sur le site de Yoctopuce et téléchargez la la librairie Yoctopuce pour Delphi[23]. Décompressez le tout dans le répertoire de votre choix, et ajoutez le sous-répertoire *sources* de l'archive dans la liste des répertoires des librairies de Delphi[24].

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

9.2. Contrôle de la fonction Humidity

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

```
program helloworld;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Windows,
  yocto_api,
  yocto_humidity;
```

[21] 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.
[22] Les librairies Delphi sont régulièrement testées avec Delphi 5 et Delphi XE2
[23] www.yoctopuce.com/FR/libraries.php
[24] Utilisez le menu **outils / options d'environement**

```
const
  serial = 'HUMSENS1-123456'; // use serial number or logical name

var
  sensor : TYHumidity;
  errmsg : string;
  done   : boolean;

begin

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

  sensor := yFindHumidity(serial+'.humidity');

  done := false;
  repeat
    if (sensor.isOnline()) then
     begin
       Write('Current humidity: '+FloatToStr(sensor.get_currentValue())+'%');
       Writeln('   (press Ctrl-C to exit)');
       Sleep(1000);
     end
    else
     begin
       Writeln('Module not connected (check identification and USB cable)');
       done := true;
     end;
  until done;

end.
```

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

yocto_api et yocto_humidity

Ces deux unités permettent d'avoir accès aux fonctions permettant de gérer les modules Yoctopuce. `yocto_api` doit toujours être utilisé, `yocto_humidity` est nécessaire pour gérer les modules contenant un capteur d'humidité, comme le Yocto–Humidity.

yRegisterHub

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

yFindHumidity

La fonction `yFindHumidity`, permet de retrouver un capteur d'humidité 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–Humidity avec le numéros de série *HUMSENS1-123456* que vous auriez appelé "*MonModule*" et dont vous auriez nommé la fonction *humidity* "*MaFonction*", les cinq appels suivants seront strictement équivalents (pour autant que *MaFonction* ne soit définie qu'une fois, pour éviter toute ambiguïté):

```
humidity := yFindHumidity("HUMSENS1-123456.humidity");
humidity := yFindHumidity("HUMSENS1-123456.MaFonction");
humidity := yFindHumidity("MonModule.humidity");
humidity := yFindHumidity("MonModule.MaFonction");
humidity := yFindHumidity("MaFonction");
```

`yFindHumidity` renvoie un objet que vous pouvez ensuite utiliser à loisir pour contrôler le capteur d'humidité.

isOnline

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

get_currentValue

La méthode `get_currentValue()` de l'objet renvoyé par `yFindHumidity` permet d'obtenir l'humidité relative actuelle mesurée par le capteur. La valeur de retour est un nombre flottant, représentant directement le nombre de %RH.

9.3. 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 = 'HUMSENS1-123456'; // 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('');
       Writeln('r : refresh / b:beacon ON / space : beacon off');
     end
    else Writeln('Module not connected (check identification and USB cable)');
  end;


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

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

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

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

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

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

Modifications des réglages du module

Lorsque que vous souhaitez modifier les réglages d'un module, il suffit d'appeler la fonction `set_xxx()` correspondante, cependant cette modification n'a lieu que dans la mémoire vive du module: si le module redémarre, les modifications seront perdue. 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 = 'HUMSENS1-123456'; // use serial number or logical name

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

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

  module := yFindModule(serial);
  if (not(module.isOnline)) then
   begin
     writeln('Module not connected (check identification and USB cable)');
     exit;
   end;

  Writeln('Current logical name : '+module.get_logicalName());
  Write('Enter new name : ');
  Readln(newname);
  if (not(yCheckLogicalName(newname))) then
   begin
     Writeln('invalid logical name');
     exit;
   end;
  module.set_logicalName(newname);
  module.saveToFlash();

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

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

Énumération des modules

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

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

var
  module : TYModule;
  errmsg : string;
```

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

  Writeln('Device list');

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

end.
```

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

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

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

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

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

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

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

10.1. Fonctions générales

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js    <script type='text/javascript' src='yocto_api.js'></script>
php   require_once('yocto_api.php');
cpp   #include "yocto_api.h"
m     #include "yocto_api.h"
pas   uses yocto_api;
```

Fonctions globales
yCheckLogicalName(**name**) Vérifie si un nom donné est valide comme nom logique pour un module ou une fonction.
yDisableExceptions() Désactive l'utilisation d'exceptions pour la gestion des erreurs.
yEnableExceptions() Réactive l'utilisation d'exceptions pour la gestion des erreurs.
yFreeAPI()

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

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

yHandleEvents(**errmsg**)
Maintient la communication de la librairie avec les modules Yoctopuce.

yInitAPI(**mode**, **errmsg**)
Initialise la librairie de programmation de Yoctopuce explicitement.

yRegisterHub(**url**, **errmsg**)
Configure la librairie Yoctopuce pour utiliser les modules connectés sur une machine donnée.

ySleep(**ms_duration**, **errmsg**)
Effectue une pause dans l'exécution du programme pour une durée spécifiée.

yUpdateDeviceList(**errmsg**)
Force une mise-à-jour de la liste des modules Yoctopuce connectés.

yCheckLogicalName()

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

js function **yCheckLogicalName**(**name**)
php function **yCheckLogicalName**(**$name**)
cpp bool **yCheckLogicalName**(const string& **name**)
m bool **yCheckLogicalName**(NSString * **name**)
pas function **yCheckLogicalName**(**name**: string): boolean

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

Paramètres :
name une chaîne de caractères contenant le nom vérifier.

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

yDisableExceptions()

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

js function **yDisableExceptions**( )
php function **yDisableExceptions**( )
cpp void **yDisableExceptions**( )
m void **yDisableExceptions**( )
pas procedure **yDisableExceptions**( )

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

yEnableExceptions()

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

js function **yEnableExceptions**( )
php function **yEnableExceptions**( )

cpp void **yEnableExceptions**( )

m void **yEnableExceptions**( )

pas procedure **yEnableExceptions**( )

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

yFreeAPI()

Libère la mémoire dynamique utilisée par la librairie Yoctopuce.

js function **yFreeAPI**( )

php function **yFreeAPI**( )

cpp void **yFreeAPI**( )

m void **yFreeAPI**( )

pas procedure **yFreeAPI**( )

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

yGetTickCount()

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

js function **yGetTickCount**( )

php function **yGetTickCount**( )

cpp u64 **yGetTickCount**( )

m u64 **yGetTickCount**( )

pas function **yGetTickCount**( ): u64

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

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

yHandleEvents()

Maintient la communication de la librairie avec les modules Yoctopuce.

cpp YRETCODE **yHandleEvents**(string& **errmsg**)

m YRETCODE **yHandleEvents**(NSString** **errmsg**)

pas function **yHandleEvents**(var **errmsg**: string): integer

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

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

yInitAPI()

Initialise la librairie de programmation de Yoctopuce explicitement.

js function **yInitAPI**(**mode**, **errmsg**)
php function **yInitAPI**(**$mode**, **&$errmsg**)
cpp YRETCODE **yInitAPI**(int **mode**, string& **errmsg**)
m YRETCODE **yInitAPI**(int **mode**, NSString** **errmsg**)
pas function **yInitAPI**(**mode**: integer, var **errmsg**: string): integer

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

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

Paramètres :

mode un entier spécifiant le type de détection automatique de modules à utiliser. Les valeurs possibles sont Y_DETECT_NONE, Y_DETECT_USB, Y_DETECT_NET et Y_DETECT_ALL.

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

yRegisterHub()

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

js function **yRegisterHub**(**url**, **errmsg**)
php function **yRegisterHub**(**$url**, **&$errmsg**)
cpp YRETCODE **yRegisterHub**(const string& **url**, string& **errmsg**)
m YRETCODE **yRegisterHub**(NSString * **url**, NSString** **errmsg**)
pas function **yRegisterHub**(**url**: string, var **errmsg**: string): integer

Dans le cas d'une utilisation avec la passerelle VirtualHub, vous devez donner en paramètre l'adresse de la machine où tourne le VirtualHub (typiquement "http://127.0.0.1:4444", qui désigne la machine locale). Si vous utilisez un langage qui a un accès direct à USB, vous pouvez utiliser la pseudo-adresse "usb" à la place.

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

Paramètres :

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

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

Retourne :

YAPI_SUCCESS si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

ySleep()

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

cpp YRETCODE **ySleep**(unsigned **ms_duration**, string& **errmsg**)
m void **ySleep**(unsigned **ms_duration**, NSString** **errmsg**)
pas function **ySleep**(**ms_duration**: integer, var **errmsg**: string): integer

L'attente est passive, c'est-à-dire qu'elle n'occupe pas significativement le processeur, de sorte à le laisser disponible pour les autres processus fonctionnant sur la machine. Durant l'attente, la librairie va néanmoins continuer à lire périodiquement les informations en provenance des modules Yoctopuce en appelant la fonction `yHandleEvents()` afin de se maintenir à jour.

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

Paramètres :

ms_duration un entier correspondant à la durée de la pause, en millisecondes

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

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

yUpdateDeviceList()

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

js function **yUpdateDeviceList**(**errmsg**)
php function **yUpdateDeviceList**(**&$errmsg**)
cpp YRETCODE **yUpdateDeviceList**(string& **errmsg**)
m YRETCODE **yUpdateDeviceList**(NSString** **errmsg**)
pas function **yUpdateDeviceList**(var **errmsg**: string): integer

La librairie va vérifier sur les machines ou ports USB précédemment enregistrés en utilisant la fonction `yRegisterHub` si un module a été connecté ou déconnecté, et le cas échéant appeler les fonctions de callback définies par l'utilisateur.

Cette fonction peut être appelée aussi souvent que désiré, afin de rendre l'application réactive aux événements de hot-plug.

Paramètres :

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

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

10.2. Interface de contrôle du module

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js <script type='text/javascript' src='yocto_api.js'></script>
php require_once('yocto_api.php');
cpp #include "yocto_api.h"
m #include "yocto_api.h"
```

```pas
uses yocto_api;
```

Fonctions globales

yFindModule(**func**)
Permet de retrouver un module d'après son numéro de série ou son nom logique.

yFirstModule()
Commence l'énumération des modules accessibles par la librairie.

Méthodes des objets `YModule`

describe()
Retourne un court texte décrivant le module.

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

errType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de l'objet module.

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

isOnline_async(**callback**, **context**)
Vérifie si le module est joignable, sans déclencher d'erreur.

load(**msValidity**)
Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

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

functionId(**functionIndex**)
Retourne l'identifiant matériel de la *n*ième fonction du module.

functionName(**functionIndex**)
Retourne le nom logique de la *n*ième fonction du module.

functionValue(**functionIndex**)
Retourne la valeur publiée par la *n*ième fonction du module.

get_adminPassword()
Retourne une chaîne formée d'étoiles si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide.

get_beacon()
Retourne l'état de la balise de localisation.

get_firmwareRelease()
Retourne la version du logiciel embarqué du module.

get_logicalName()
Retourne le nom logique du module.

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

get_persistentSettings()
Retourne l'état courant des réglages persistents du module.

get_productId()
Retourne l'identifiant USB du module, préprogrammé en usine.

get_productName()
Retourne le nom commercial du module, préprogrammé en usine.

get_productRelease()
Retourne le numéro de version matériel du module, préprogrammé en usine.

get_realmHTTP()
Retourne le nom du domaine d'identification utilisé par le module.

get_rebootCountdown()
Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

get_serialNumber()
Retourne le numéro de série du module, préprogrammé en usine.

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

get_usbBandwidth()
Retourne le nombre d'interface USB utilisé par le module.

get_usbCurrent()
Retourne le courant consommé par le module sur le bus USB, en milliampères.

get_userPassword()
Retourne une chaîne formée d'étoiles si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

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

reboot(**secBeforeReboot**)
Agende un simple redémarrage du module dans un nombre donné de secondes.

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

saveToFlash()
Sauve les réglages courants dans la mémoire non volatile du module.

set_adminPassword(**newval**)
Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute modification de la configuration du module.

set_beacon(**newval**)
Allume ou éteint la balise de localisation du module.

set_logicalName(**newval**)
Change le nom logique du module.

set_luminosity(**newval**)
Modifie la luminosité des leds informatives du module.

set_realmHTTP(**newval**)
Modifie le nom du domaine d'identification utilisé par le module.

set_usbBandwidth(**newval**)
Modifie le nombre d'interface USB utilisé par le module.

set_userPassword(**newval**)

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

triggerFirmwareUpdate(**secBeforeReboot**)
Agende un redémarrage du module en mode spécial de reprogrammation du logiciel embarqué.

yFindModule()

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

js function **yFindModule**(**func**)
php function **yFindModule**(**$func**)
cpp YModule * **yFindModule**(const string& **func**)
m YModule* **yFindModule**(NSString* **func**)
pas function **yFindModule**(**func**: string): TYModule

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

Paramètres :
func une chaîne de caractères contenant soit le numéro de série, soit le nom logique du module désiré

Retourne :
un objet de classe `YModule` qui permet ensuite de contrôler le module ou d'obtenir de plus amples informations sur le module.

yFirstModule()

Commence l'énumération des modules accessibles par la librairie.

js function **yFirstModule**( )
php function **yFirstModule**( )
cpp YModule * **yFirstModule**( )
m YModule* **yFirstModule**( )
pas function **yFirstModule**( ): TYModule

Utiliser la fonction `YModule.nextModule()` pour itérer sur les autres modules.

Retourne :
un pointeur sur un objet `YModule`, correspondant au premier module accessible en ligne, ou `null` si aucun module n'a été trouvé.

YFunction.describe()

Retourne un court texte décrivant le module.

Ce texte peut contenir soit le nom logique du module, soit son numéro de série.

Retourne :
une chaîne de caractères décrivant le module

YFunction.errMessage()

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

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

Retourne :
une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation du module

YFunction.errType()

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

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

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

YFunction.isOnline()

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

Si les valeurs des attributs du module en cache sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si le module est joignable, `false` sinon

YFunction.isOnline_async()

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

Si les valeurs des attributs du module en cache sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/ sortie bloquants.

Paramètres :
callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet module concerné et le résultat booléen
context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

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

YFunction.load()

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

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

Paramètres :

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

Retourne :
`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

YFunction.load_async()

Met en cache les valeurs courantes du module, avec une durée de validité spécifiée.

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/ sortie bloquants.

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet module concerné et le code d'erreur (ou `YAPI_SUCCESS`)

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

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

YModule.functionCount()

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

`js` function **functionCount**( )
`php` function **functionCount**( )
`cpp` int **functionCount**( )
`m` -(int) **functionCount**
`pas` function **functionCount**( ): integer

Retourne :
le nombre de fonctions sur le module

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

YModule.functionId()

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

`js` function **functionId**(**functionIndex**)
`php` function **functionId**(**$functionIndex**)
`cpp` string **functionId**(int **functionIndex**)
`m` -(NSString*) **functionId** : (int) **functionIndex**
`pas` function **functionId**(**functionIndex**: integer): string

Paramètres :

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

Retourne :

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

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

YModule.functionName()

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

```
js    function functionName( functionIndex)
php   function functionName( $functionIndex)
cpp   string functionName( int functionIndex)
m     -(NSString*) functionName : (int) functionIndex
pas   function functionName( functionIndex: integer): string
```

Paramètres :

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

Retourne :

une chaîne de caractères correspondant au nom logique de la fonction désirée

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

YModule.functionValue()

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

```
js    function functionValue( functionIndex)
php   function functionValue( $functionIndex)
cpp   string functionValue( int functionIndex)
m     -(NSString*) functionValue : (int) functionIndex
pas   function functionValue( functionIndex: integer): string
```

Paramètres :

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

Retourne :

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

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

YModule.get_adminPassword()

Retourne une chaîne formée d'étoiles si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide.

```
js    function get_adminPassword( )
php   function get_adminPassword( )
cpp   string get_adminPassword( )
m     -(NSString*) get_adminPassword
pas   function get_adminPassword( ): string
```

Retourne :

une chaîne de caractères représentant une chaîne formée d'étoiles si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide

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

YModule.get_beacon()

Retourne l'état de la balise de localisation.

js function **get_beacon**( )
php function **get_beacon**( )
cpp Y_BEACON_enum **get_beacon**( )
m -(Y_BEACON_enum) **get_beacon**
pas function **get_beacon**( ): Integer

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

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

YModule.get_firmwareRelease()

Retourne la version du logiciel embarqué du module.

js function **get_firmwareRelease**( )
php function **get_firmwareRelease**( )
cpp string **get_firmwareRelease**( )
m -(NSString*) **get_firmwareRelease**
pas function **get_firmwareRelease**( ): string

Retourne :
une chaîne de caractères représentant la version du logiciel embarqué du module

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

YModule.get_logicalName()

Retourne le nom logique du module.

js function **get_logicalName**( )
php function **get_logicalName**( )
cpp string **get_logicalName**( )
m -(NSString*) **get_logicalName**
pas function **get_logicalName**( ): string

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

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

YModule.get_luminosity()

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

js function **get_luminosity**( )
php function **get_luminosity**( )
cpp int **get_luminosity**( )
m -(int) **get_luminosity**
pas function **get_luminosity**( ): LongInt

Retourne :
un entier représentant la luminosité des leds informatives du module (valeur entre 0 et 100)

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

YModule.get_persistentSettings()

Retourne l'état courant des réglages persistents du module.

js function **get_persistentSettings**( )
php function **get_persistentSettings**( )
cpp Y_PERSISTENTSETTINGS_enum **get_persistentSettings**( )
m -(Y_PERSISTENTSETTINGS_enum) **get_persistentSettings**
pas function **get_persistentSettings**( ): Integer

Retourne :
une valeur parmi `Y_PERSISTENTSETTINGS_LOADED`, `Y_PERSISTENTSETTINGS_SAVED` et `Y_PERSISTENTSETTINGS_MODIFIED` représentant l'état courant des réglages persistents du module

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

YModule.get_productId()

Retourne l'identifiant USB du module, préprogrammé en usine.

js function **get_productId**( )
php function **get_productId**( )
cpp int **get_productId**( )
m -(int) **get_productId**
pas function **get_productId**( ): LongInt

Retourne :
un entier représentant l'identifiant USB du module, préprogrammé en usine

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

YModule.get_productName()

Retourne le nom commercial du module, préprogrammé en usine.

js function **get_productName**( )
php function **get_productName**( )
cpp string **get_productName**( )
m -(NSString*) **get_productName**
pas function **get_productName**( ): string

Retourne :
une chaîne de caractères représentant le nom commercial du module, préprogrammé en usine

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

YModule.get_productRelease()

Retourne le numéro de version matériel du module, préprogrammé en usine.

js function **get_productRelease**( )
php function **get_productRelease**( )
cpp int **get_productRelease**( )
m -(int) **get_productRelease**
pas function **get_productRelease**( ): LongInt

Retourne :
un entier représentant le numéro de version matériel du module, préprogrammé en usine

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

YModule.get_realmHTTP()

Retourne le nom du domaine d'identification utilisé par le module.

js function **get_realmHTTP**( )
php function **get_realmHTTP**( )
cpp string **get_realmHTTP**( )
m -(NSString*) **get_realmHTTP**
pas function **get_realmHTTP**( ): string

Retourne :
une chaîne de caractères représentant le nom du domaine d'identification utilisé par le module

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

YModule.get_rebootCountdown()

Retourne le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé.

js function **get_rebootCountdown**( )
php function **get_rebootCountdown**( )
cpp int **get_rebootCountdown**( )
m -(int) **get_rebootCountdown**
pas function **get_rebootCountdown**( ): LongInt

Retourne :
un entier représentant le nombre de secondes restantes avant un redémarrage du module, ou zéro si aucun redémarrage n'a été agendé

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

YModule.get_serialNumber()

Retourne le numéro de série du module, préprogrammé en usine.

js function **get_serialNumber**( )
php function **get_serialNumber**( )
cpp string **get_serialNumber**( )
m -(NSString*) **get_serialNumber**
pas function **get_serialNumber**( ): string

Retourne :
une chaîne de caractères représentant le numéro de série du module, préprogrammé en usine

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

YModule.get_upTime()

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

js function **get_upTime**( )
php function **get_upTime**( )
cpp unsigned **get_upTime**( )
m -(unsigned) **get_upTime**
pas function **get_upTime**( ): LongWord

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

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

YModule.get_usbBandwidth()

Retourne le nombre d'interface USB utilisé par le module.

js function **get_usbBandwidth**( )
php function **get_usbBandwidth**( )
cpp Y_USBBANDWIDTH_enum **get_usbBandwidth**( )
m -(Y_USBBANDWIDTH_enum) **get_usbBandwidth**
pas function **get_usbBandwidth**( ): Integer

Retourne :
soit `Y_USBBANDWIDTH_SIMPLE`, soit `Y_USBBANDWIDTH_DOUBLE`, selon le nombre d'interface USB utilisé par le module

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

YModule.get_usbCurrent()

Retourne le courant consommé par le module sur le bus USB, en milliampères.

js function **get_usbCurrent**( )
php function **get_usbCurrent**( )
cpp unsigned **get_usbCurrent**( )
m -(unsigned) **get_usbCurrent**
pas function **get_usbCurrent**( ): LongWord

Retourne :
un entier représentant le courant consommé par le module sur le bus USB, en milliampères

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

YModule.get_userPassword()

Retourne une chaîne formée d'étoiles si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

js function **get_userPassword**( )
php function **get_userPassword**( )
cpp string **get_userPassword**( )
m -(NSString*) **get_userPassword**
pas function **get_userPassword**( ): string

Retourne :
une chaîne de caractères représentant une chaîne formée d'étoiles si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide

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

YModule.nextModule()

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

js function **nextModule**( )
php function **nextModule**( )
cpp YModule * **nextModule**( )
m -(YModule*) **nextModule**
pas function **nextModule**( ): TYModule

Retourne :
un pointeur sur un objet `YModule` accessible en ligne, ou `null` lorsque l'énumération est terminée.

YModule.reboot()

Agende un simple redémarrage du module dans un nombre donné de secondes.

```
js    function reboot( secBeforeReboot)
php   function reboot( $secBeforeReboot)
cpp   int reboot( int secBeforeReboot)
m     -(int) reboot : (int) secBeforeReboot
pas   function reboot( secBeforeReboot: integer): integer
```

Paramètres :
secBeforeReboot nombre de secondes avant de redémarrer

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

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

YModule.revertFromFlash()

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

```
js    function revertFromFlash( )
php   function revertFromFlash( )
cpp   int revertFromFlash( )
m     -(int) revertFromFlash
pas   function revertFromFlash( ): integer
```

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

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

YModule.saveToFlash()

Sauve les réglages courants dans la mémoire non volatile du module.

```
js    function saveToFlash( )
php   function saveToFlash( )
cpp   int saveToFlash( )
m     -(int) saveToFlash
pas   function saveToFlash( ): integer
```

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

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

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

YModule.set_adminPassword()

Modifie le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute modification de la configuration du module.

js function **set_adminPassword**(**newval**)

php function **set_adminPassword**(**$newval**)

cpp string **set_adminPassword**(const string& **newval**)

m -(NSString*) **set_adminPassword** : (NSString*) **newval**

pas function **set_adminPassword**(**newval**: string): string

Si la valeur fournie est une chaîne vide, plus aucun mot de passe n'est nécessaire. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le mot de passe pour l'utilisateur "admin", qui devient alors instantanément nécessaire pour toute modification de la configuration du module

Retourne :

la nouvelle valeur effective, telle que confirmée par le module.

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

YModule.set_beacon()

Allume ou éteint la balise de localisation du module.

js function **set_beacon**(**newval**)

php function **set_beacon**(**$newval**)

cpp Y_BEACON_enum **set_beacon**(Y_BEACON_enum **newval**)

m -(Y_BEACON_enum) **set_beacon** : (Y_BEACON_enum) **newval**

pas function **set_beacon**(**newval**: Integer): Integer

Paramètres :

newval soit `Y_BEACON_OFF`, soit `Y_BEACON_ON`

Retourne :

la nouvelle valeur effective, telle que confirmée par le module.

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

YModule.set_logicalName()

Change le nom logique du module.

js function **set_logicalName**(**newval**)

php function **set_logicalName**(**$newval**)

cpp string **set_logicalName**(const string& **newval**)

m -(NSString*) **set_logicalName** : (NSString*) **newval**

pas function **set_logicalName**(**newval**: string): string

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères

Retourne :

la nouvelle valeur effective, telle que confirmée par le module.

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

YModule.set_luminosity()

Modifie la luminosité des leds informatives du module.

js function **set_luminosity**(**newval**)
php function **set_luminosity**(**$newval**)
cpp int **set_luminosity**(int **newval**)
m -(int) **set_luminosity** : (int) **newval**
pas function **set_luminosity**(**newval**: LongInt): LongInt

Le paramêtre est une valeur entre 0 et 100. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :
newval un entier représentant la luminosité des leds informatives du module

Retourne :
la nouvelle valeur effective, telle que confirmée par le module.

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

YModule.set_realmHTTP()

Modifie le nom du domaine d'identification utilisé par le module.

js function **set_realmHTTP**(**newval**)
php function **set_realmHTTP**(**$newval**)
cpp string **set_realmHTTP**(const string& **newval**)
m -(NSString*) **set_realmHTTP** : (NSString*) **newval**
pas function **set_realmHTTP**(**newval**: string): string

N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :
newval une chaîne de caractères représentant le nom du domaine d'identification utilisé par le module

Retourne :
la nouvelle valeur effective, telle que confirmée par le module.

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

YModule.set_usbBandwidth()

Modifie le nombre d'interface USB utilisé par le module.

js function **set_usbBandwidth**(**newval**)
php function **set_usbBandwidth**(**$newval**)
cpp Y_USBBANDWIDTH_enum **set_usbBandwidth**(Y_USBBANDWIDTH_enum **newval**)
m -(Y_USBBANDWIDTH_enum) **set_usbBandwidth** : (Y_USBBANDWIDTH_enum) **newval**
pas function **set_usbBandwidth**(**newval**: Integer): Integer

Paramètres :
newval soit `Y_USBBANDWIDTH_SIMPLE`, soit `Y_USBBANDWIDTH_DOUBLE`, selon le nombre d'interface USB utilisé par le module

Retourne :
la nouvelle valeur effective, telle que confirmée par le module.

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

YModule.set_userPassword()

Modifie le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module.

js function **set_userPassword**(**newval**)

php function **set_userPassword**(**$newval**)

cpp string **set_userPassword**(const string& **newval**)

m -(NSString*) **set_userPassword** : (NSString*) **newval**

pas function **set_userPassword**(**newval**: string): string

Si la valeur fournie est une chaîne vide, plus aucun mot de passe n'est nécessaire. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le mode de passe pour l'utilisateur "user", qui devient alors instantanément nécessaire pour tout accès au module

Retourne :

la nouvelle valeur effective, telle que confirmée par le module.

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

YModule.triggerFirmwareUpdate()

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

js function **triggerFirmwareUpdate**(**secBeforeReboot**)

php function **triggerFirmwareUpdate**(**$secBeforeReboot**)

cpp int **triggerFirmwareUpdate**(int **secBeforeReboot**)

m -(int) **triggerFirmwareUpdate** : (int) **secBeforeReboot**

pas function **triggerFirmwareUpdate**(**secBeforeReboot**: integer): integer

Paramètres :

secBeforeReboot nombre de secondes avant de redémarrer

Retourne :

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

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

10.3. Interface de la fonction Humidity

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

js <script type='text/javascript' src='yocto_humidity.js'></script>

php require_once('yocto_humidity.php');

cpp #include "yocto_humidity.h"

m #include "yocto_humidity.h"

pas uses yocto_humidity;

Fonctions globales

yFindHumidity(**func**)

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

yFirstHumidity()

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

Méthodes des objets `YHumidity`

describe()

Retourne un court texte décrivant la fonction.

errMessage()
Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

errType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

get_advertisedValue()
Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

get_currentValue()
Retourne la valeur mesurée actuelle.

get_highestValue()
Retourne la valeur maximale observée.

get_logicalName()
Retourne le nom logique du capteur d'humidité.

get_lowestValue()
Retourne la valeur minimale observée.

isOnline()
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

load(**msValidity**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

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

module_async(**callback**, **context**)
Retourne l'objet `YModule` correspondant au module Yoctopuce qui héberge la fonction.

nextHumidity()
Continue l'énumération des capteurs d'humidité commencée à l'aide de `yFirstHumidity()`.

set_highestValue(**newval**)
Modifie la mémoire de valeur maximale observée.

set_logicalName(**newval**)
Modifie le nom logique du capteur d'humidité.

set_lowestValue(**newval**)
Modifie la mémoire de valeur minimale observée.

yFindHumidity()

Permet de retrouver un capteur d'humidité d'après un identifiant donné.

`js` function **yFindHumidity**(**func**)
`php` function **yFindHumidity**(**$func**)
`cpp` YHumidity * **yFindHumidity**(const string& **func**)
`m` YHumidity* **yFindHumidity**(NSString* **func**)
`pas` function **yFindHumidity**(**func**: string): TYHumidity

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

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

Paramètres :
func une chaîne de caractères qui référence le capteur d'humidité sans ambiguïté

Retourne :
un objet de classe `YHumidity` qui permet ensuite de contrôler le capteur d'humidité.

---

yFirstHumidity()

Commence l'énumération des capteurs d'humidité accessibles par la librairie.

js function **yFirstHumidity**( )
php function **yFirstHumidity**( )
cpp YHumidity * **yFirstHumidity**( )
m YHumidity* **yFirstHumidity**( )
pas function **yFirstHumidity**( ): TYHumidity

Utiliser la fonction `YHumidity.nextHumidity()` pour itérer sur les autres capteurs d'humidité.

Retourne :
un pointeur sur un objet `YHumidity`, correspondant au premier capteur d'humidité accessible en ligne, ou `null` si il n'y a pas de capteurs d'humidité disponibles.

YHumidity.describe()

Retourne un court texte décrivant la fonction.

js function **describe**( )
php function **describe**( )
cpp string **describe**( )
m -(NSString*) **describe**
pas function **describe**( ): string

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

YHumidity.errMessage()

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

js function **errMessage**( )
php function **errMessage**( )

cpp string **errMessage**( )

m -(NSString*) **errMessage**

pas function **errMessage**( ): string

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

Retourne :
une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la fonction

YHumidity.errType()

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

js function **errType**( )

php function **errType**( )

cpp YRETCODE **errType**( )

m -(YRETCODE) **errType**

pas function **errType**( ): YRETCODE

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

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

YHumidity.get_advertisedValue()

Retourne la valeur courante du capteur d'humidité (pas plus de 6 caractères).

js function **get_advertisedValue**( )

php function **get_advertisedValue**( )

cpp string **get_advertisedValue**( )

m -(NSString*) **get_advertisedValue**

pas function **get_advertisedValue**( ): string

Retourne :
une chaîne de caractères représentant la valeur courante du capteur d'humidité (pas plus de 6 caractères)

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

YHumidity.get_currentValue()

Retourne la valeur mesurée actuelle.

js function **get_currentValue**( )

php function **get_currentValue**( )

cpp double **get_currentValue**( )

m -(double) **get_currentValue**

pas function **get_currentValue**( ): double

Retourne :
une valeur numérique représentant la valeur mesurée actuelle

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

YHumidity.get_highestValue()

Retourne la valeur maximale observée.

js function **get_highestValue**( )
php function **get_highestValue**( )
cpp double **get_highestValue**( )
m -(double) **get_highestValue**
pas function **get_highestValue**( ): double

Retourne :
une valeur numérique représentant la valeur maximale observée

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

YHumidity.get_logicalName()

Retourne le nom logique du capteur d'humidité.

js function **get_logicalName**( )
php function **get_logicalName**( )
cpp string **get_logicalName**( )
m -(NSString*) **get_logicalName**
pas function **get_logicalName**( ): string

Retourne :
une chaîne de caractères représentant le nom logique du capteur d'humidité

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

YHumidity.get_lowestValue()

Retourne la valeur minimale observée.

js function **get_lowestValue**( )
php function **get_lowestValue**( )
cpp double **get_lowestValue**( )
m -(double) **get_lowestValue**
pas function **get_lowestValue**( ): double

Retourne :
une valeur numérique représentant la valeur minimale observée

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

YHumidity.isOnline()

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

js function **isOnline**( )
php function **isOnline**( )
cpp bool **isOnline**( )
m -(BOOL) **isOnline**
pas function **isOnline**( ): boolean

Si les valeurs des attributs en cache de la fonction sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :

`true` si la fonction est joignable, `false` sinon

YHumidity.isOnline_async()

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

js function **isOnline_async**(**callback**, **context**)

Si les valeurs des attributs en cache de la fonction sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/ sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

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

YHumidity.load()

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

js function **load**(**msValidity**)

php function **load**(**$msValidity**)

cpp YRETCODE **load**(int **msValidity**)

m -(YRETCODE) **load** : (int) **msValidity**

pas function **load**(**msValidity**: integer): YRETCODE

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

Paramètres :

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

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

YHumidity.load_async()

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

js function **load_async**(**msValidity**, **callback**, **context**)

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/ sortie bloquants.

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

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

YHumidity.module()

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

`js` function **module**( )

`php` function **module**( )

`cpp` YModule * **module**( )

`m` -(YModule*) **module**

`pas` function **module**( ): TYModule

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

Retourne :

une instance de `YModule`

YHumidity.module_async()

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

`js` function **module_async**(**callback**, **context**)

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/ sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

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

YHumidity.nextHumidity()

Continue l'énumération des capteurs d'humidité commencée à l'aide de `yFirstHumidity()`.

`js` function **nextHumidity**( )

`php` function **nextHumidity**( )

`cpp` YHumidity * **nextHumidity**( )

`m` -(YHumidity*) **nextHumidity**

`pas` function **nextHumidity**( ): TYHumidity

Retourne :

un pointeur sur un objet `YHumidity` accessible en ligne, ou `null` lorsque l'énumération est terminée.

YHumidity.set_highestValue()

Modifie la mémoire de valeur maximale observée.

js function **set_highestValue**(**newval**)
php function **set_highestValue**(**$newval**)
cpp double **set_highestValue**(double **newval**)
m -(double) **set_highestValue** : (double) **newval**
pas function **set_highestValue**(**newval**: double): double

Paramètres :
newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :
la nouvelle valeur effective, telle que confirmée par le module.

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

YHumidity.set_logicalName()

Modifie le nom logique du capteur d'humidité.

js function **set_logicalName**(**newval**)
php function **set_logicalName**(**$newval**)
cpp string **set_logicalName**(const string& **newval**)
m -(NSString*) **set_logicalName** : (NSString*) **newval**
pas function **set_logicalName**(**newval**: string): string

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :
newval une chaîne de caractères représentant le nom logique du capteur d'humidité

Retourne :
la nouvelle valeur effective, telle que confirmée par le module.

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

YHumidity.set_lowestValue()

Modifie la mémoire de valeur minimale observée.

js function **set_lowestValue**(**newval**)
php function **set_lowestValue**(**$newval**)
cpp double **set_lowestValue**(double **newval**)
m -(double) **set_lowestValue** : (double) **newval**
pas function **set_lowestValue**(**newval**: double): double

Paramètres :
newval une valeur numérique représentant la mémoire de valeur minimale observée

Retourne :
la nouvelle valeur effective, telle que confirmée par le module.

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

10.4. Interface de la fonction Temperature

La librairie de programmation Yoctopuce permet lire une valeur instantanée du capteur, ainsi que les extrémas atteints.

Pour utiliser les fonctions décrites ici, vous devez inclure:

```
js    <script type='text/javascript' src='yocto_temperature.js'></script>
php   require_once('yocto_temperature.php');
cpp   #include "yocto_temperature.h"
m     #include "yocto_temperature.h"
pas   uses yocto_temperature;
```

Fonctions globales

yFindTemperature(**func**)
Permet de retrouver un capteur de température d'après un identifiant donné.

yFirstTemperature()
Commence l'énumération des capteurs de température accessibles par la librairie.

Méthodes des objets `YTemperature`

describe()
Retourne un court texte décrivant la fonction.

errMessage()
Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

errType()
Retourne le code d'erreur correspondant à la dernière erreur survenue lors de l'utilisation de la fonction.

get_advertisedValue()
Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

get_currentValue()
Retourne la valeur mesurée actuelle.

get_highestValue()
Retourne la valeur maximale observée.

get_logicalName()
Retourne le nom logique du capteur de température.

get_lowestValue()
Retourne la valeur minimale observée.

isOnline()
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

isOnline_async(**callback**, **context**)
Vérifie si le module hébergeant la fonction est joignable, sans déclencher d'erreur.

load(**msValidity**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

load_async(**msValidity**, **callback**, **context**)
Met en cache les valeurs courantes de la fonction, avec une durée de validité spécifiée.

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

module_async(**callback**, **context**)

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

nextTemperature()
Continue l'énumération des capteurs de température commencée à l'aide de `yFirstTemperature()`.

set_highestValue(**newval**)
Modifie la mémoire de valeur maximale observée.

set_logicalName(**newval**)
Modifie le nom logique du capteur de température.

set_lowestValue(**newval**)
Modifie la mémoire de valeur minimale observée.

yFindTemperature()

Permet de retrouver un capteur de température d'après un identifiant donné.

js function **yFindTemperature**(**func**)
php function **yFindTemperature**(**$func**)
cpp YTemperature * **yFindTemperature**(const string& **func**)
m YTemperature* **yFindTemperature**(NSString* **func**)
pas function **yFindTemperature**(**func**: string): TYTemperature

L'identifiant peut être spécifié sous plusieurs formes:

- NomLogiqueFonction
- NoSerieModule.IdentifiantFonction
- NoSerieModule.NomLogiqueFonction
- NomLogiqueModule.IdentifiantMatériel
- NomLogiqueModule.NomLogiqueFonction

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

Paramètres :
func une chaîne de caractères qui référence le capteur de température sans ambiguïté

Retourne :
un objet de classe `YTemperature` qui permet ensuite de contrôler le capteur de température.

yFirstTemperature()

Commence l'énumération des capteurs de température accessibles par la librairie.

js function **yFirstTemperature**( )
php function **yFirstTemperature**( )
cpp YTemperature * **yFirstTemperature**( )
m YTemperature* **yFirstTemperature**( )
pas function **yFirstTemperature**( ): TYTemperature

Utiliser la fonction `YTemperature.nextTemperature()` pour itérer sur les autres capteurs de température.

Retourne :
un pointeur sur un objet `YTemperature`, correspondant au premier capteur de température accessible en ligne, ou `null` si il n'y a pas de capteurs de température disponibles.

YTemperature.describe()

Retourne un court texte décrivant la fonction.

```
js    function describe( )
php   function describe( )
cpp   string describe( )
m     -(NSString*) describe
pas   function describe( ): string
```

Ce texte inclut toujours le nom de la classe, et peut contenir en sus soit le nom logique de la fonction, soit son identifiant hardware.

Retourne :
une chaîne de caractères décrivant la fonction

YTemperature.errMessage()

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

```
js    function errMessage( )
php   function errMessage( )
cpp   string errMessage( )
m     -(NSString*) errMessage
pas   function errMessage( ): string
```

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

Retourne :
une chaîne de caractères correspondant au message de la dernière erreur qui s'est produit lors de l'utilisation de la fonction

YTemperature.errType()

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

```
js    function errType( )
php   function errType( )
cpp   YRETCODE errType( )
m     -(YRETCODE) errType
pas   function errType( ): YRETCODE
```

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

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

YTemperature.get_advertisedValue()

Retourne la valeur courante du capteur de température (pas plus de 6 caractères).

```
js    function get_advertisedValue( )
php   function get_advertisedValue( )
cpp   string get_advertisedValue( )
```

m -(NSString*) **get_advertisedValue**

pas function **get_advertisedValue**( ): string

Retourne :
une chaîne de caractères représentant la valeur courante du capteur de température (pas plus de 6 caractères)

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

YTemperature.get_currentValue()

Retourne la valeur mesurée actuelle.

js function **get_currentValue**( )

php function **get_currentValue**( )

cpp double **get_currentValue**( )

m -(double) **get_currentValue**

pas function **get_currentValue**( ): double

Retourne :
une valeur numérique représentant la valeur mesurée actuelle

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

YTemperature.get_highestValue()

Retourne la valeur maximale observée.

js function **get_highestValue**( )

php function **get_highestValue**( )

cpp double **get_highestValue**( )

m -(double) **get_highestValue**

pas function **get_highestValue**( ): double

Retourne :
une valeur numérique représentant la valeur maximale observée

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

YTemperature.get_logicalName()

Retourne le nom logique du capteur de température.

js function **get_logicalName**( )

php function **get_logicalName**( )

cpp string **get_logicalName**( )

m -(NSString*) **get_logicalName**

pas function **get_logicalName**( ): string

Retourne :
une chaîne de caractères représentant le nom logique du capteur de température

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

YTemperature.get_lowestValue()

Retourne la valeur minimale observée.

js function **get_lowestValue**( )

php function **get_lowestValue**( )

cpp double **get_lowestValue**( )

m -(double) **get_lowestValue**

pas function **get_lowestValue**( ): double

Retourne :
une valeur numérique représentant la valeur minimale observée

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

YTemperature.isOnline()

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

js function **isOnline**( )

php function **isOnline**( )

cpp bool **isOnline**( )

m -(BOOL) **isOnline**

pas function **isOnline**( ): boolean

Si les valeurs des attributs en cache de la fonction sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Retourne :
`true` si la fonction est joignable, `false` sinon

YTemperature.isOnline_async()

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

js function **isOnline_async**(**callback**, **context**)

Si les valeurs des attributs en cache de la fonction sont valides au moment de l'appel, le module est considéré joignable. Cette fonction ne cause en aucun cas d'exception, quelle que soit l'erreur qui pourrait se produire lors de la vérification de joignabilité.

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/ sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le résultat booléen

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

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

YTemperature.load()

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

js function **load**(**msValidity**)

php function **load**(**$msValidity**)

cpp YRETCODE **load**(int **msValidity**)

m -(YRETCODE) **load** : (int) **msValidity**

pas function **load**(**msValidity**: integer): YRETCODE

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

Paramètres :

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

Retourne :

`YAPI_SUCCESS` si l'opération se déroule sans erreur. En cas d'erreur, déclenche une exception ou retourne un code d'erreur négatif.

YTemperature.load_async()

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

`js` function **load_async**(**msValidity**, **callback**, **context**)

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/ sortie bloquants.

Paramètres :

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

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et le code d'erreur (ou `YAPI_SUCCESS`)

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

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

YTemperature.module()

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

`js` function **module**( )
`php` function **module**( )
`cpp` YModule * **module**( )
`m` -(YModule*) **module**
`pas` function **module**( ): TYModule

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

Retourne :

une instance de `YModule`

YTemperature.module_async()

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

`js` function **module_async**(**callback**, **context**)

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

Cette version asynchrone n'existe qu'en Javascript. Elle utilise une fonction de callback plutôt qu'une simple valeur de retour, pour éviter de bloquer la VM Javascript de Firefox, qui n'implémente pas le passage de contrôle entre threads durant les appels d'entrée/ sortie bloquants.

Paramètres :

callback fonction de callback qui sera appelée dès que le résultat sera connu. La fonction callback reçoit trois arguments: le contexte fourni par l'appelant, l'objet fonction concerné et l'instance demandée de `YModule`

context contexte fourni par l'appelant, et qui sera passé tel-quel à la fonction de callback

Retourne :

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

YTemperature.nextTemperature()

Continue l'énumération des capteurs de température commencée à l'aide de `yFirstTemperature()`.

js function **nextTemperature**( )

php function **nextTemperature**( )

cpp YTemperature * **nextTemperature**( )

m -(YTemperature*) **nextTemperature**

pas function **nextTemperature**( ): TYTemperature

Retourne :

un pointeur sur un objet `YTemperature` accessible en ligne, ou `null` lorsque l'énumération est terminée.

YTemperature.set_highestValue()

Modifie la mémoire de valeur maximale observée.

js function **set_highestValue**(**newval**)

php function **set_highestValue**(**$newval**)

cpp double **set_highestValue**(double **newval**)

m -(double) **set_highestValue** : (double) **newval**

pas function **set_highestValue**(**newval**: double): double

Paramètres :

newval une valeur numérique représentant la mémoire de valeur maximale observée

Retourne :

la nouvelle valeur effective, telle que confirmée par le module.

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

YTemperature.set_logicalName()

Modifie le nom logique du capteur de température.

js function **set_logicalName**(**newval**)

php function **set_logicalName**(**$newval**)

cpp string **set_logicalName**(const string& **newval**)

m -(NSString*) **set_logicalName** : (NSString*) **newval**

pas function **set_logicalName**(**newval**: string): string

Vous pouvez utiliser `yCheckLogicalName()` pour vérifier si votre paramètre est valide. N'oubliez pas d'appeler la méthode `saveToFlash()` du module si le réglage doit être préservé.

Paramètres :

newval une chaîne de caractères représentant le nom logique du capteur de température

Retourne :
la nouvelle valeur effective, telle que confirmée par le module.

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

YTemperature.set_lowestValue()

Modifie la mémoire de valeur minimale observée.

```
js    function set_lowestValue( newval)
php   function set_lowestValue( $newval)
cpp   double set_lowestValue( double newval)
m     -(double) set_lowestValue : (double) newval
pas   function set_lowestValue( newval: double): double
```

Paramètres :
newval une valeur numérique représentant la mémoire de valeur minimale observée

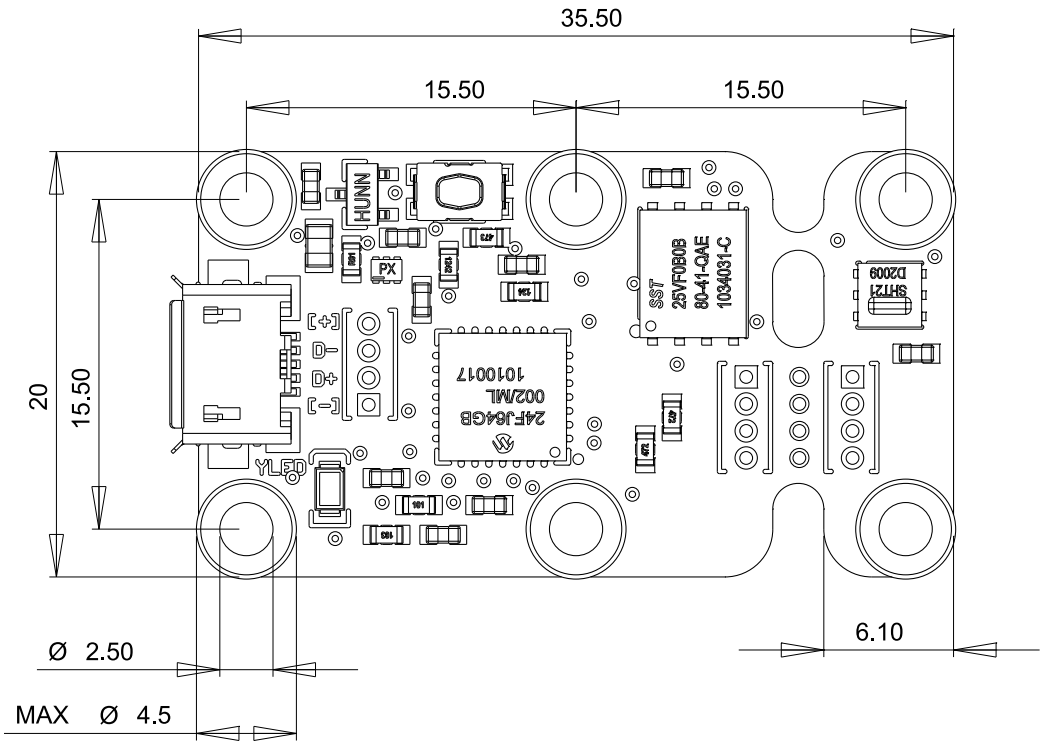
Retourne :
la nouvelle valeur effective, telle que confirmée par le module.

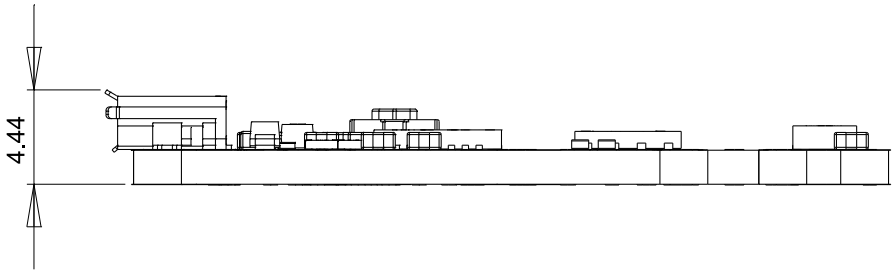
En cas d'erreur, déclenche une exception ou retourne `Y_LOWESTVALUE_INVALID`.

11. Caractéristiques

Vous trouverez résumées ci dessous les principales caractéristiques techniques de votre module Yocto–Humidity

Senseur	SHT21 (Sensirion)
Largeur	20 mm
Longueur	35.5 mm
Poids	3 g
Connecteur USB	micro-B
Plage de mesure (H)	0..100 %
Plage de mesure (T)	-40 ... +125 °C
Précision (H)	4 %
Précision (T)	0.3 °C
Sytème d'exploitation supportés	Windows, Linux, Mac OS X
Drivers	Fonctionne sans driver
API de programmation	Javascript, PHP, C++, Delphi
RoHS	oui





All dimensions are in mm
Toutes les dimensions sont en mm

Yocto-Humidity

A4

Scale
3:1
Echelle

12. Index