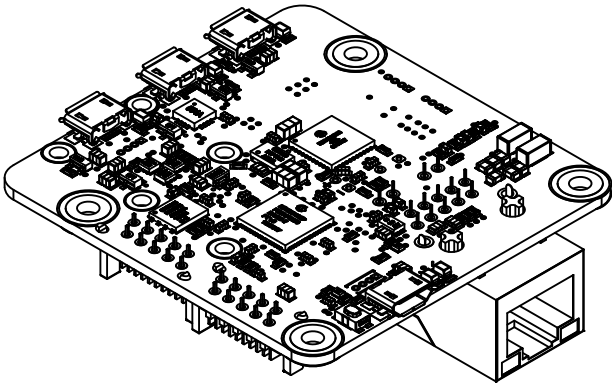
# YoctoHub-Ethernet

## Mode d'emploi

# Table des matières

# 1. Introduction

Le YoctoHub-Ethernet est un module électronique de 60x58mm qui permet de contrôler d'autres modules Yoctopuce à travers Ethernet. Vu de l'extérieur, ce module se comporte exactement comme un ordinateur classique faisant tourner VirtualHub[1]: même interface, mêmes fonctionnalités. Le YoctoHub-Ethernet peut être alimenté classiquement à l'aide d'un chargeur USB, mais il peut aussi être aussi alimenté par *Power over Ethernet*.



*Le YoctoHub-Ethernet*

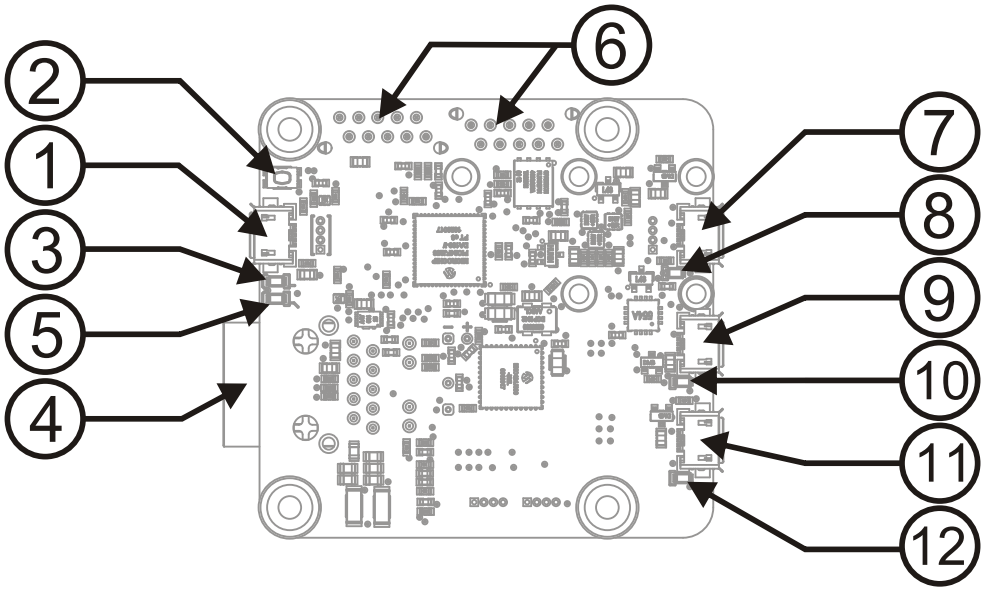
Le YoctoHub-Ethernet a été conçu pour être déployé facilement et ne pas demander de maintenance particulière. Contrairement à un mini-PC, il n'utilise pas un système d'exploitation complexe. Quelques réglages simples permettent son utilisation dans toutes sortes d'environnements réseau. Ces réglages peuvent être effectués manuellement ou de manière automatisée, par USB. Il convient de ce fait beaucoup mieux à une industrialisation qu'un mini-PC. En revanche, il ne permet pas l'exécution de programmes supplémentaires écrits par l'utilisateur.

Le YoctoHub-Ethernet n'est pas un hub USB standard avec accès réseau. Bien qu'utilisant du câblage USB, ses ports descendants utilisent un protocole propriétaire, plus simple qu'USB. Il n'est par conséquent pas possible de contrôler, ni même d'alimenter, des périphériques USB standards avec un YoctoHub-Ethernet.

Yoctopuce vous remercie d'avoir fait l'acquisition de ce YoctoHub-Ethernet et espère sincèrement qu'il vous donnera entière satisfaction. Les ingénieurs Yoctopuce se sont donnés beaucoup de mal pour que votre YoctoHub-Ethernet soit facile à installer n'importe où et soit facile utiliser en toutes circonstances. Néanmoins, si ce module venait à vous décevoir, n'hésitez pas à contacter le support Yoctopuce[2].

[1] http://www.yoctopuce.com/FR/virtualhub.php
[2] support@yoctopuce.com

# 2. Présentation




*1: Port USB de contrôle + alimentation*
*2: Yocto-bouton*
*3: Yocto-Led*
*4: Port réseau (100Mb)*
*5: Indicateur de sur-consommation*
*6: Connection dorsale*
*7: Port descendant 1*
*8: Indicateur port descendant 1*
*9: Port descendant 2*
*10: Indicateur port descendant 2*
*11: Port descendant 3*
*12: Indicateur port descendant 3*

## 2.1. Les éléments du YoctoHub-Ethernet

### Le numéro de série

Chaque Yocto-module a un numéro de série unique attribué en usine, pour les modules YoctoHub-Ethernet ce numéro commence par YHUBETH1. Le module peut être piloté par logiciel en utilisant ce numéro de série. Ce numéro de série ne peut pas être changé.

### Le nom logique

Le nom logique est similaire au numéro de série, c'est une chaîne de caractères sensée être unique qui permet référencer le module par logiciel. Cependant, contrairement au numéro de série, le nom logique peut être modifié à volonté. L'intérêt est de pouvoir fabriquer plusieurs exemplaires du même

projet sans avoir à modifier le logiciel de pilotage. Il suffit de programmer les mêmes noms logiques dans chaque exemplaire. Attention, le comportement d'un projet devient imprévisible s'il contient plusieurs modules avec le même nom logique et que le logiciel de pilotage essaye d'accéder à l'un de ces modules à l'aide de son nom logique. A leur sortie d'usine, les modules n'ont pas de nom logique assigné, c'est à vous de le définir.

## Le Yocto-bouton

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

## La Yocto-Led

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

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

La Yocto-Led a une troisième fonctionnalité moins plaisante: lorsque le logiciel interne qui contrôle le module rencontre une erreur fatale, elle se met à flasher SOS en morse[1]. Dans ce cas, débranchez puis re-branchez le module. Si le problème venait à se reproduire, vérifiez que le module contient bien la dernière version du firmware et, dans l'affirmative, contactez le support Yoctopuce[2].

## Le connecteur de contrôle et d'alimentation (Power / Control port)

Ce connecteur permet d'alimenter le YoctoHub-Ethernet et les modules qui lui sont connectés à l'aide d'un simple chargeur USB. Ce connecteur permet aussi de prendre le contrôle du YoctoHub-Ethernet par USB, exactement comme on pourrait le faire avec un module Yoctopuce classique. C'est particulièrement utile lorsque que l'on désire configurer le YoctoHub-Ethernet sans connaître son adresse IP.

## Les ports descendants

Vous pouvez connecter jusqu'à trois modules Yoctopuce sur ces ports. Ils seront alors accessibles comme s'ils étaient branchés à un ordinateur faisant tourner VirtualHub. Attention, le protocole entre le YoctoHub-Ethernet et le module Yoctopuce n'est pas de l'USB mais un protocole propriétaire plus léger. De ce fait le YoctoHub-Ethernet ne peut pas gérer des périphériques autres que des modules Yoctopuce. Un hub USB standard ne fonctionnera pas non plus[3]. Si vous désirez brancher plus de trois modules Yoctopuce, utilisez le connecteur dorsal pour y connecter un ou plusieurs YoctoHub-Shield[4].

Attention, les connecteurs USB du YoctoHub-Ethernet sont simplement soudés en surface et peuvent être arrachés si la prise USB venait à faire fortement 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, près du connecteur.

[1] court-court-court long-long-long court-court-court
[2] support@yoctopuce.com
[3] Le Micro-USB-Hub fabriqué par Yoctopuce est un hub USB standard et ne fonctionnera pas avec le YoctoHub-Ethernet.
[4] www.yoctopuce.com/FR/products/yoctohub-shield

## Le connecteur UTP (réseau)

Ce connecteur permet de connecter le YoctoHub-Ethernet à votre réseau Ethernet. La connection réseau du YoctoHub-Ethernet fonctionne en 100 Mb/s, mais le hub peut très bien être raccordé à un réseau Gigabit ou 10Mb/s.

Le YoctoHub-Ethernet peut aussi être alimenté par ce biais: il suffit de disposer de matériel réseau capable de fournir de l'énergie par *Power over Ethernet* (PoE) à la norme 802.3af. Ce peut être un switch réseau avec PoE, ou un simple injecteur par exemple. Longtemps réservé au monde professionnel, ce type de matériel est désormais largement distribué à des prix tout à fait abordables. En utilisant le PoE, vous pourrez déporter votre YoctoHub-Ethernet loin de toute prise électrique, en le raccordant uniquement par un simple câble réseau. Les câbles réseau plats et fins sont généralement compatibles avec le standard PoE 802.3af.

## Les sondes de courant

Le YoctoHub-Ethernet est capable de mesurer sa propre consommation de courant, en distinguant celle fournie par le connecteur USB et celle fournie par le Power over Ethernet. La distribution du courant sur un bus USB ou sur un réseau PoE étant relativement critique, cette fonctionnalité peut être d'un grand secours. Le YoctoHub-Ethernet est dimensionné pour pouvoir gérer jusqu'à 2A en tout (propre consommation et consommation sur les ports descendants incluses).

## Indicateur de sur-consommation

Le YoctoHub-Ethernet analyse en permanence sa consommation. S'il détecte une consommation globale de plus de 2A ou une consommation supérieure à 1.8A sur le PoE, suite à une surcharge sur un des ports descendants par exemple, il va automatiquement désactiver tous les ports descendants et allumer la Led l'indicateur de sur-consommation. Pour isoler la source du problème, vous pouvez réactiver les ports un à un, en surveillant l'augmentation de la consommation. Alternativement, si connaissez la source du problème de sur-consommation et savez l'avoir résolu, vous pouvez redémarrer le YoctoHub-Ethernet pour réactiver tous les ports.

Notez que l'indicateur de sur-consommation est une mesure de protection qui peut éviter la surchauffe, mais ce n'est pas une garantie de protection contre les court- circuits.

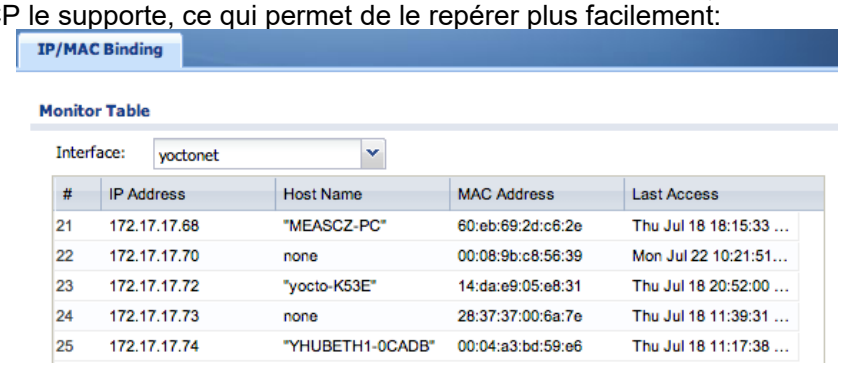
# 3. Premiers pas

Ce chapitre a pour but de vous aider à connecter et configurer votre YoctoHub-Ethernet pour la première fois.

## 3.1. Utilisation directe pour les impatients

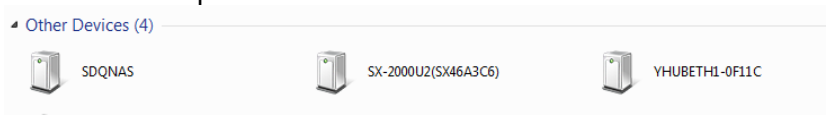
Si vous branchez votre YoctoHub-Ethernet directement au réseau sans le configurer, il est fonctionnel avec la configuration par défaut:

- L'adresse IP est obtenue par DHCP. Si aucun serveur DHCP ne répond, il utilisera une adresse d'auto-configuration dans la plage 169.254.x.y, comme la plupart des systèmes d'exploitation.
- Le YoctoHub-Ethernet indique au serveur DHCP son nom d'hôte (numéro de série), si le serveur DHCP le supporte, ce qui permet de le repérer plus facilement:



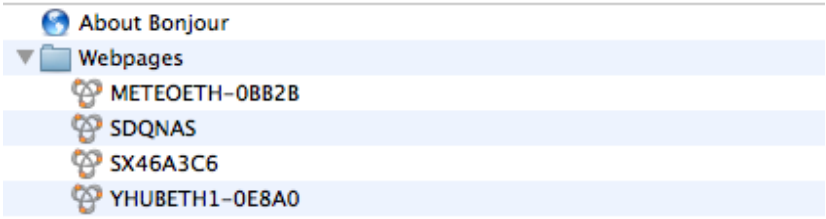

*YoctoHub-Ethernet détecté par le serveur DHCP d'un petit routeur ZyXEL*

- Le YoctoHub-Ethernet annonce sa présence sur le réseau local par le protocole *SSDP (uPNP)*, ce qui le rend visible par les machines Windows:




*YoctoHub-Ethernet détecté sur le réseau par un PC sous Windows*

- Le YoctoHub-Ethernet annonce sa présence sur le réseau local par le protocole *Bonjour*, ce qui le rend visible par les machines Mac OS X:

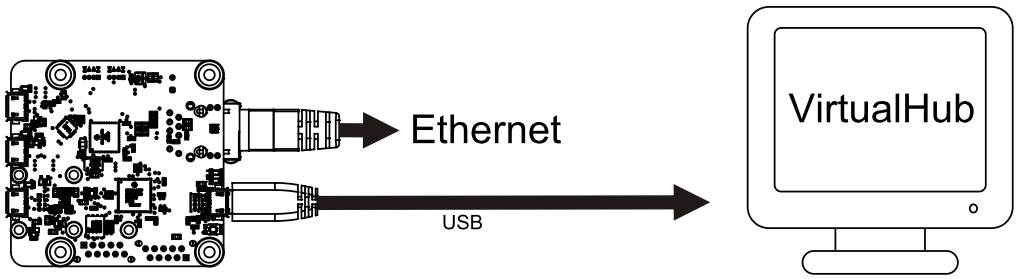



*YoctoHub-Ethernet détecté sur le réseau par un Mac*

## 3.2. Configuration manuelle

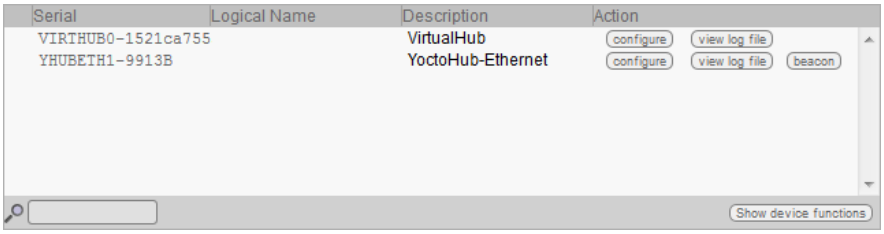
Vous pouvez configurer votre YoctoHub-Ethernet via son port de contrôle USB, en utilisant VirtualHub[1].

Lancez VirtualHub sur votre ordinateur favori et raccordez votre ordinateur au port *power / control port* du YoctoHub-Ethernet. Vous aurez besoin d'un câble USB A-MicroB. Raccordez aussi votre YoctoHub-Ethernet à un câble réseau opérationnel.




*Configuration: raccordez par USB votre YoctoHub-Ethernet à un ordinateur*

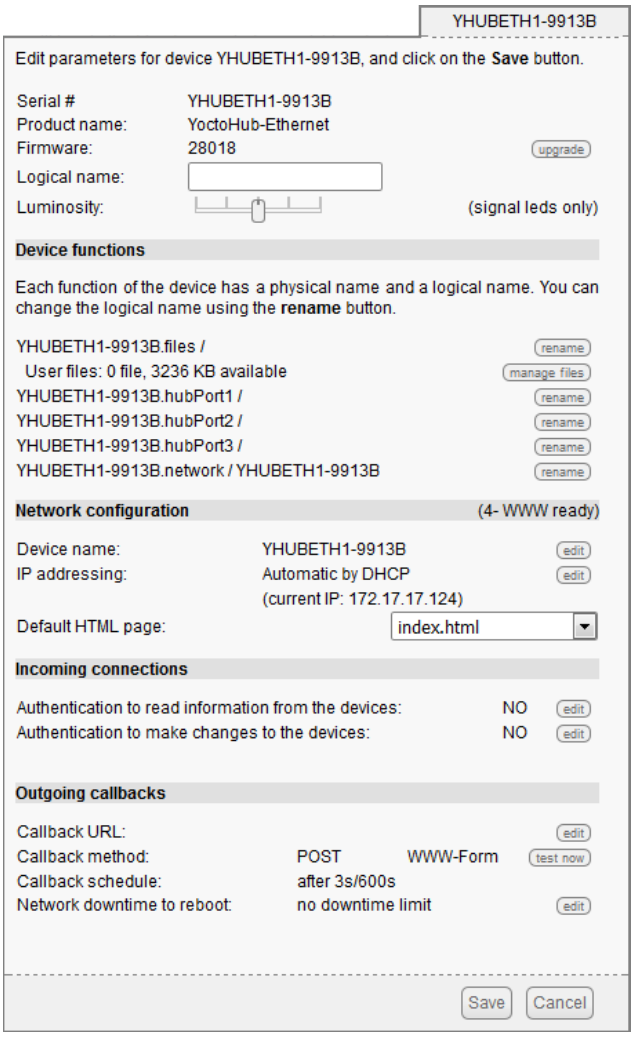
Lancez alors votre browser favori sur l'URL de votre VirtualHub. Il s'agit généralement de *http://127.0.0.1:4444*. Vous obtiendrez la liste des modules Yoctopuce connectés par USB, dont votre YoctoHub-Ethernet.




*Liste des modules Yoctopuce raccordés par USB à votre ordinateur, dont votre YoctoHub-Ethernet*

Cliquez sur le bouton **configure** correspondant à votre YoctoHub-Ethernet, votre obtiendrez la fenêtre de configuration du module. Cette fenêtre comporte une section **Network configuration**.

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

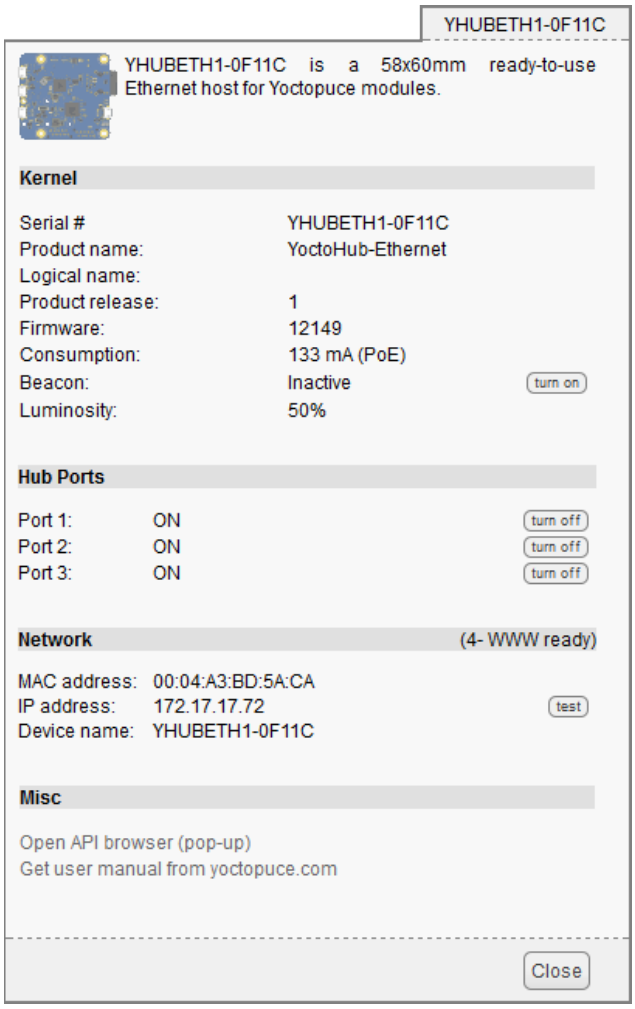



*Fenêtre de configuration du module YoctoHub-Ethernet*

Vous pouvez choisir si l'adresse IP de votre YoctoHub-Ethernet doit être attribuée par DHCP ou si elle doit être fixe. L'option DHCP est recommandée dans la mesure où cette fonctionnalité est supportée par la plupart des boîtiers ADSL (c'est la configuration par défaut). Si vous ne savez pas ce qu'est un serveur DCHP mais avez l'habitude de brancher des appareils sur votre réseau et de les voir marcher sans problème, ne touchez à rien.

Vous pouvez aussi choisir le nom réseau de votre YoctoHub-Ethernet. Vous pourrez ainsi accéder à votre YoctoHub-Ethernet en utilisant ce nom plutôt que son adresse IP. Une fois la partie réseau configurée, cliquez sur le bouton **Save**. Cela aura pour effet de sauver vos modifications et de fermer la fenêtre de configuration. Ces modifications étant sauvées dans la mémoire persistante du YoctoHub-Ethernet, il s'en rappellera même après avoir été privé de courant.

Cliquez sur le numéro de série correspondant à votre YoctoHub-Ethernet. Cela ouvrira la fenêtre des détails de votre module:




*Les propriétés du YoctoHub-Ethernet*

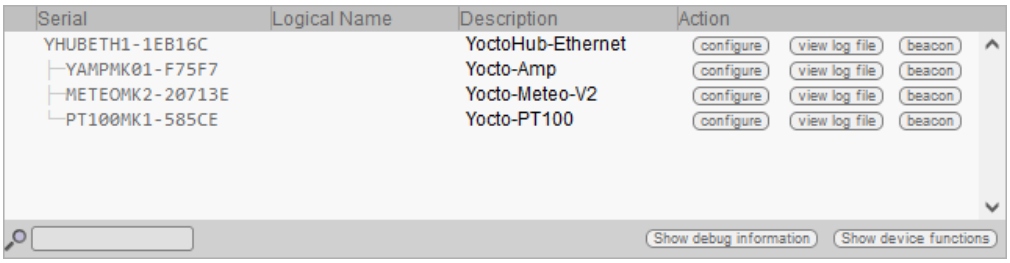
Cette fenêtre comporte une section qui relate l'état de la partie réseau du YoctoHub-Ethernet. Vous y trouverez notamment le type de connexion et son adresse IP courante. Cette section donne aussi l'état de la connexion réseau. Ces états peuvent être:

- 0- search for link: Le module cherche une liaison réseau. Si cet état persiste, il est probable que le câble réseau n'est pas branché
- 2- network linked: une liaison à été détectée.
- 3- LAN ready: le réseau local est opérationnel.
- 4- WWW ready: le module a pu vérifier la connectivité à Internet en se connectant à un serveur de temps (NTP).

Après avoir vérifié que votre module a bien une adresse IP valide, vous pouvez fermer la fenêtre détails, arrêter VirtualHub et débrancher le câble USB de contrôle de votre ordinateur: il suffira que le module soit alimenté pour l'utiliser.

Vous pouvez désormais accéder à votre YoctoHub-Ethernet en tapant directement son adresse IP dans la barre d'adresse de votre browser favori. Le module répond au port HTTP standard, mais aussi au port 4444 utilisé par VirtualHub. Si l'adresse IP de votre module est *192.168.0.10*, vous pourrez donc le joindre avec l'URL *http://192.168.0.10*.

Vous obtiendrez alors directement l'interface du YoctoHub-Ethernet. Cette interface est en tout point identique à celle de VirtualHub. Vous retrouvez le YoctoHub-Ethernet sur la première ligne et les modules connectés au YoctoHub-Ethernet sur les suivantes.




*L'interface du YoctoHub-Ethernet est identique à celle de VirtualHub.*

Si vous avez attribué un nom à votre YoctoHub-Ethernet, vous pouvez aussi utiliser ce nom sur le réseau local. Par exemple, si vous avez utilisé le nom réseau *yoctohub*, vous pouvez contacter le module avec l'URL *http://yoctohub* sous Windows et avec l'URL *http://yoctohub.local* sous Mac OS X et Linux. Notez que cette technique est limitée au sous-réseau du YoctoHub-Ethernet. Si vous voulez contacter le module par nom depuis un autre réseau, vous devez utiliser une infrastructure DNS classique.

# 3.3. Configuration automatisée

Il est possible d'industrialiser la configuration réseau du YoctoHub-Ethernet. Vous trouverez dans les chapitres suivants de cette documentation la description des fonctions de programmation permettant de relire l'adresse Ethernet d'un module (adresse MAC), et de configurer tous ses paramètres réseau.

Les fonctions de configuration réseau sont aussi accessibles par ligne de commande, en utilisant l'utilitaire `YNetwork` disponible dans la librairie de programmation en ligne de commande[2].

Après avoir effectué un changement de réglage par programmation, prenez garde à appeler la fonction `saveToFlash()` pour vous assurez que les réglages soient sauvés de manière permanente dans la mémoire flash du module.

# 3.4. Connexions

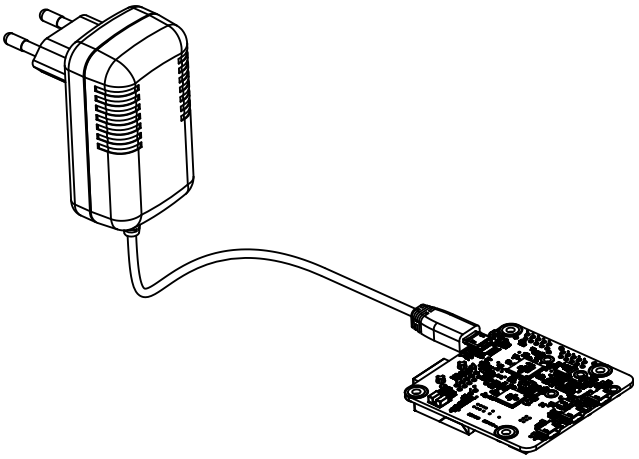
## Alimentation

Le YoctoHub-Ethernet peut être alimenté de deux manières différentes:

### USB

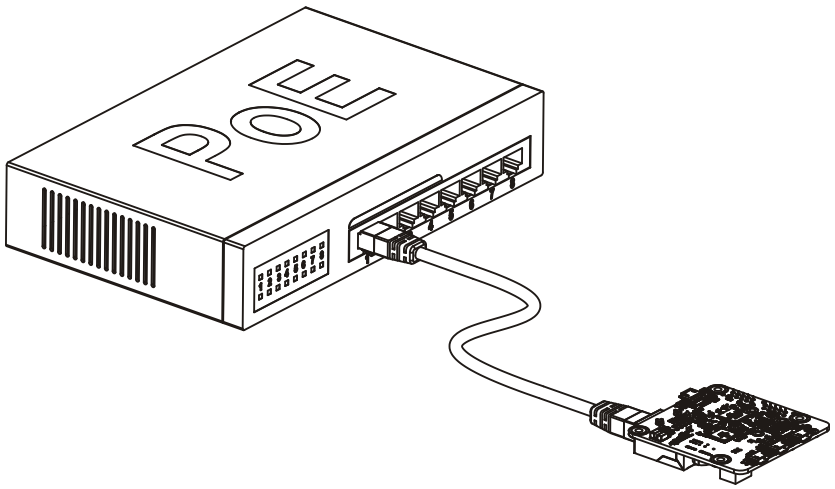
Branchez simplement un chargeur USB dans le port *power / control port*, assurez-vous tout de même que le chargeur soit d'une puissance électrique suffisante: le YoctoHub-Ethernet consomme environ 130mA, auxquels il faudra ajouter la consommation de chaque sous-module. Le YoctoHub-Ethernet est conçu pour gérer 2A au maximum, c'est pourquoi un chargeur USB capable de délivrer au moins 2A est recommandé. Par ailleurs, vous devrez veiller à ce que la consommation totale de l'ensemble hub + sous-modules ne dépasse pas cette limite.

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



*Le YoctoHub-Ethernet peut être alimenté par un chargeur USB*

**Power over Ethernet (PoE)**
Votre YoctoHub-Ethernet est compatible *Power over Ethernet* (PoE). Cette technologie consiste à faire passer le courant d'alimentation directement dans le câble réseau. Pour cela vous avez besoin d'un équipement réseau qui alimente le câble Ethernet selon le standard 802.3af [3] Ces équipements se présentent généralement sous la forme d'un *Switch PoE*, combinant les fonctions "switch Ethernet" et "injecteur de puissance", ou d'un simple *Injecteur PoE*, permettant d'injecter la puissance sur un seul câble Ethernet. Il existe aussi des routeurs ADSL avec PoE, mais ceux-ci sont relativement rares. Si vous disposez d'un câble réseau relié à l'un de ces équipements, il vous suffit alors de connecter ce câble au YoctoHub-Ethernet pour que celui-ci soit immédiatement opérationnel.




*Le YoctoHub-Ethernet peut être alimenté par le câble Ethernet (PoE)*

Longtemps réservés au monde professionnel, ces équipements sont désormais facilement accessibles au grand public à des prix tout à fait raisonnables (quelques dizaines d'Euros pour un injecteur, et un peu plus de 100 EUR pour un Switch PoE avec plusieurs ports alimentés). Les câbles réseau plats et fins sont généralement compatibles avec le standard PoE 802.3af.

Le composant PoE du YoctoHub-Ethernet permet de délivrer environ 1.8 A à 5V. Si le YoctoHub-Ethernet détecte une surconsommation sur le PoE, il coupera automatiquement l'alimentation des ports descendants pour se protéger. Ces ports devront être réactivés explicitement, soit par logiciel soit grâce l'interface web du YoctoHub-Ethernet. La sur-consommation est physiquement signalée par la Led rouge *overload*.

Attention, la distribution de courant entre équipements PoE fait l'objet de négociations. Le YoctoHub-Ethernet va tenter de négocier 9W (1.8A) auprès de l'équipement qui fourni le courant sur le câble.

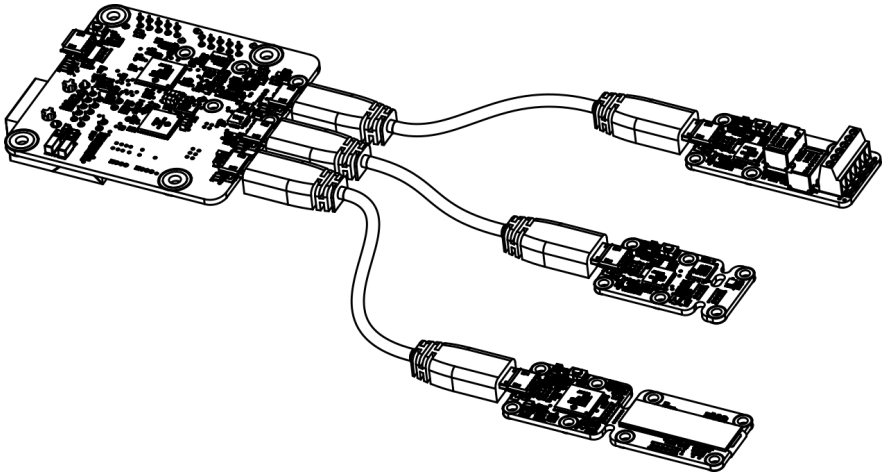
[3] Le YoctoHub-Ethernet extrait la puissance du câble Ethernet à l'aide d'un composant *AG9705M*, qui implémente les deux alternatives A et B décrites dans le standard 802.3af. L'isolation galvanique entre le module et le réseau supporte 1500V.

Veillez à ne pas dépasser la puissance disponible sur l'équipement qui fournit le courant. Reportez-vous à la documentation de l'équipement en question pour plus de détails.

Il n'est pas recommandé de brancher en même temps un câble PoE et un chargeur USB sur votre YoctoHub-Ethernet. A priori, cela ne pose pas de problème majeur pour une utilisation très temporaire, mais en utilisation normale vous devez choisir l'une ou l'autre des sources d'alimentation.

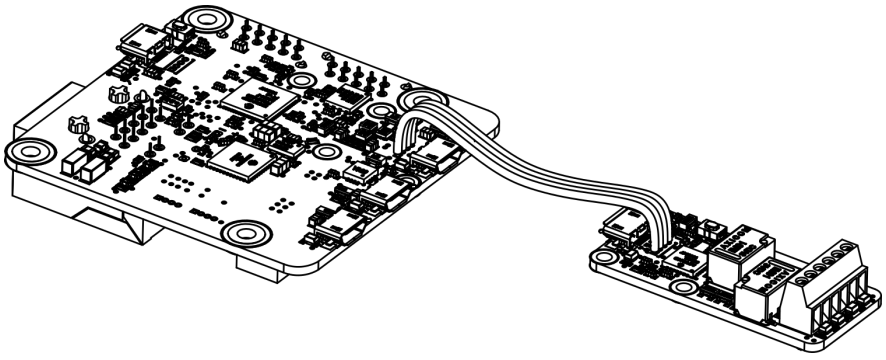
## Sous-modules

Le YoctoHub-Ethernet est capable de piloter tous les modules Yoctopuce de la gamme *Yocto*. Ces modules peuvent êtres connectés directement aux ports descendants, ils seront détectés automatiquement. Vous aurez besoin pour cela de câbles USB MicroB-MicroB. Vous pouvez utiliser des câbles OTG ou non, cela n'a pas d'importance.



*Connexion des sous-modules à l'aide de câbles USB*

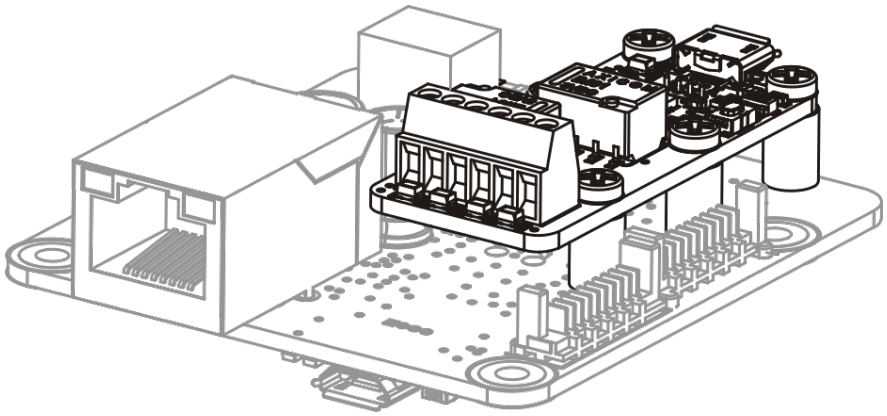
Alternativement, vous pouvez connecter vos modules de manière plus compacte à l'aide de câbles au pas 1.27mm: tous les modules Yoctopuce disposent en effet de contacts à cet effet. Vous pouvez soit souder des connecteurs 1.27mm sur les modules et utiliser des câbles avec connecteurs 1.27mm, soit souder directement du câble plat au pas 1.27mm. Si vous choisissez cette dernière option, il est recommandé d'utiliser du câble plat mono-brin, moins souple que le multi-brin mais beaucoup plus facile à souder. Soyez particulièrement attentif aux polarités: Le YoctoHub-Ethernet, tout comme l'ensemble de modules de la gamme Yoctopuce, n'est pas protégé contre les inversions de polarité. Une telle inversion a toutes les chances de détruire vos équipements. Assurez-vous que la position du contact carré de part et d'autre du câble correspondent.



*Connexion des sous-modules à l'aide de câble nappe*

Le YoctoHub-Ethernet est conçu pour que vous puissiez fixer un module simple largeur directement dessus. Vous aurez besoin de vis, d'entretoises[4] et d'un connecteur au pas 1.27mm[5]. Vous pouvez ainsi transformer un module Yoctopuce USB en en module réseau tout en gardant un format très compact.

[4] http://www.yoctopuce.com/FR/products/accessoires-et-connectique/fix-2-5mm
[5] http://www.yoctopuce.com/FR/products/accessoires-et-connectique/board2board-127



*Fixation d'un module directement sur le hub*

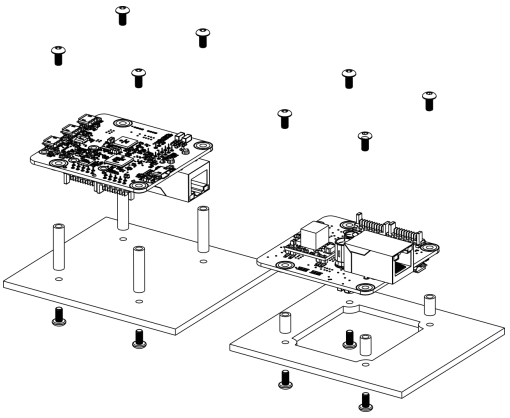
Attention, le YoctoHub-Ethernet est conçu pour piloter des modules Yoctopuce uniquement. En effet le protocole utilisé entre le YoctoHub-Ethernet et les sous-modules n'est pas de l'USB mais un protocole propriétaire, beaucoup plus léger. Si d'aventure vous branchez un périphérique autre qu'un module Yoctopuce sur un des ports descendants du YoctoHub-Ethernet, le port en question sera automatiquement désactivé pour éviter d'endommager le périphérique.

# 4. Montage

Ce chapitre fournit des explications importantes pour utiliser votre module YoctoHub-Ethernet 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 hub 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.



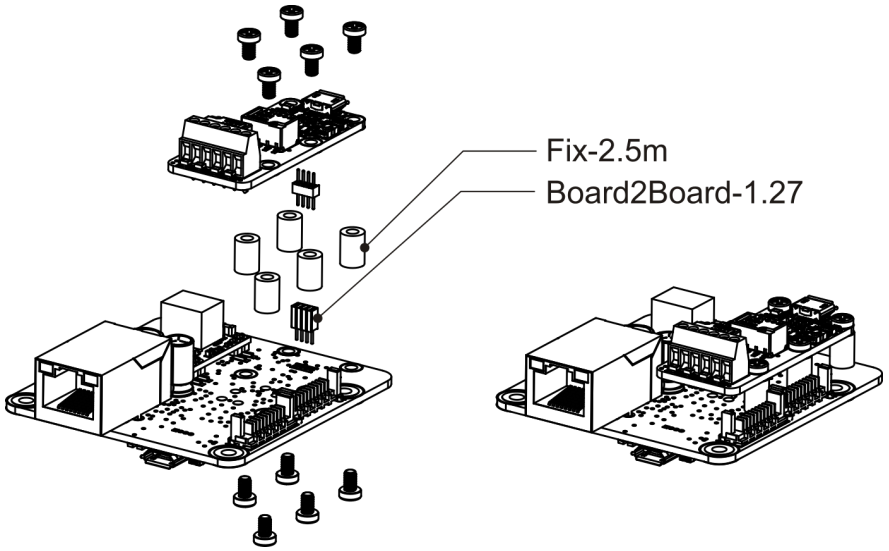
*Exemples de montage sur un support.*

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

Veillez à que l'électronique module ne soit pas en contact avec le support. La méthode recommandée consiste à utiliser des entretoises. Vous pouvez monter le module dans le sens qui vous convient: mais vous devez être conscient du fait que les composants électroniques du YoctoHub-Ethernet, la partie réseau en particulier, dégagent de la chaleur. Vous devrez donc faire en sorte que la chaleur ne puisse pas s'accumuler.

## 4.2. Fixation d'un sous-module

Le YoctoHub-Ethernet est conçu pour que vous puissiez visser un module simple largeur directement dessus. Par simple largeur, on entend les modules de 20 mm de large. Tous les modules simple largeur ont leurs 5 trous de fixation et le connecteur USB au même endroit. Le sous-module peut être fixé à l'aide de vis et d'entretoises. Il y a derrière les connecteurs USB du YoctoHub-Ethernet et du sous-module un ensemble de 4 contacts qui permettent d'effectuer la connexion électrique entre le hub et le sous-module. Si vous ne vous sentez pas suffisamment à l'aise avec un fer à souder, vous pouvez aussi aussi utiliser un simple câble USB MicroB-MicroB, OTG ou non.



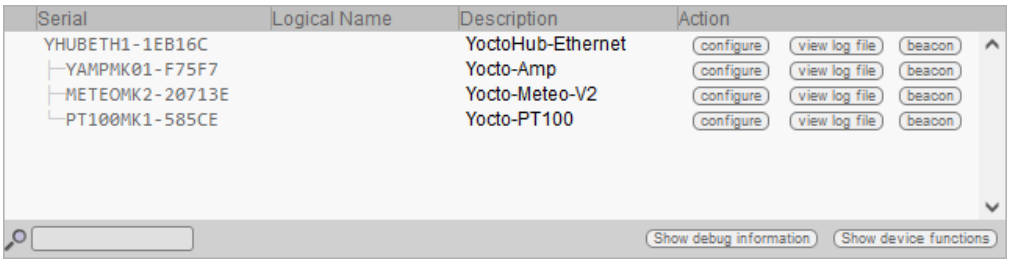

*Fixation d'un module directement sur le hub*

Prenez garde à bien monter le module sur la face prévue, comme illustré ci-dessus. Les 5 trous du module doivent correspondre aux 5 trous du YoctoHub-Ethernet, et le contact carré sur sur le module doit être connecté au contact carré sur le port descendant du YoctoHub-Ethernet. Si vous montez un module sur l'autre face ou d'une autre manière, la polarité du connecteur sera inversée et vous risquez fort d'endommager définitivement votre matériel.

Tous les accessoires nécessaires à la fixation d'un module sur votre YoctoHub-Ethernet sont relativement courants. Vous pourrez les trouver sur le site de Yoctopuce tout comme sur la plupart des sites vendant du matériel électronique. Attention cependant, la tête des vis servant à fixer le sous-module devra avoir un diamètre maximum de 4.5 millimètres, sous peine d'endommager les composants électroniques.

# 5. Configuration et test des modules

Une fois installé et configuré, YoctoHub-Ethernet permet de tester et configurer vos modules Yoctopuce. Pour ce faire, ouvrez votre navigateur internet favori[1]. Connectez-vous en *HTTP* sur l'interface Web du YoctoHub-Ethernet, comme décrit au chapitre "Premiers pas". La liste des modules connectés au hub devrait apparaître.




*YoctoHub-Ethernet: Interface Web*

En bas de page se trouvent deux boutons. Le premier bouton, **Show debug information**, permet d'afficher et ensuite d'enregistrer toutes les informations nécessaires pour débugger un problème lié au YoctoHub-Ethernet, c'est-à-dire:

- La liste de tous les modules détectés.
- La valeur de tous les paramètres de tous les modules (sans les mots de passe).
- Les logs de tous les modules.
- La liste de tous les fichiers uploadés sur les modules, mais pas leur contenu.
- Le contenu des éventuels core dump de YoctoHub-Ethernet.

Si vous devez contacter le support, il est important de télécharger ces informations et de les envoyer avec votre demande.

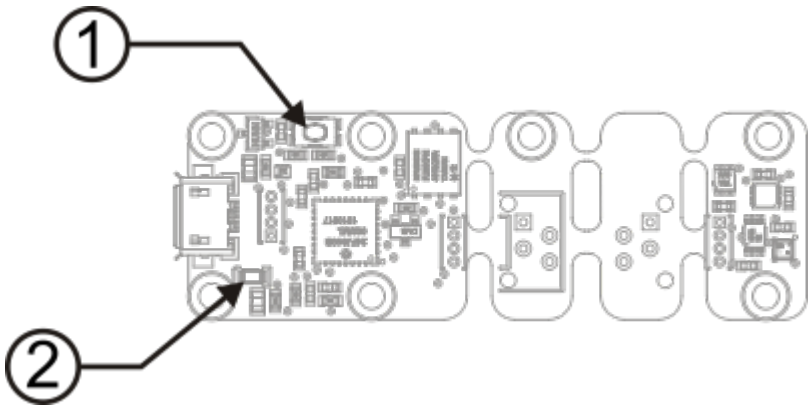
Le deuxième bouton, **Show device functions**, montre toutes les fonctions du YoctoHub-Ethernet et de chacun des modules connectés au YoctoHub-Ethernet.

## 5.1. Localisation des modules

L'interface principale vous montre une ligne par module connecté, si vous avez plusieurs modules du même modèle, vous pouvez localiser un module particulier en cliquant sur le bouton **beacon**

[1] L'interface du YoctoHub-Ethernet est régulièrement testée sur Firefox, Chrome, Opera et Brave. Elle fonctionne probablement avec Safari.

correspondant: cela aura pour effet de faire clignoter la led bleue du module et d'afficher sur l'interface une pastille bleue au début de la ligne correspondante. Vous pouvez faire la même manipulation en appuyant sur le Yocto-bouton d'un module connecté.

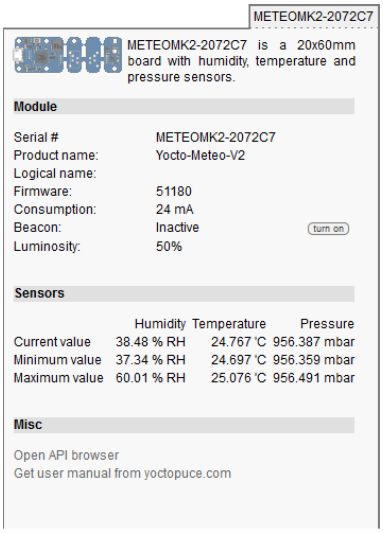



*Yocto-bouton (1) et led de localisation (2) d'un module Yocto-Meteo-V2. Ces deux éléments sont toujours placés au même endroit, quelque soit le module.*

## 5.2. Test des modules

Pour tester un module, cliquez simplement sur le numéro de série d'un module dans l'interface, une fenêtre spécifique au module s'ouvrira. Cette fenêtre permet généralement d'activer les fonctions principales du module. Reportez vous au manuel du module correspondant pour plus de détails.

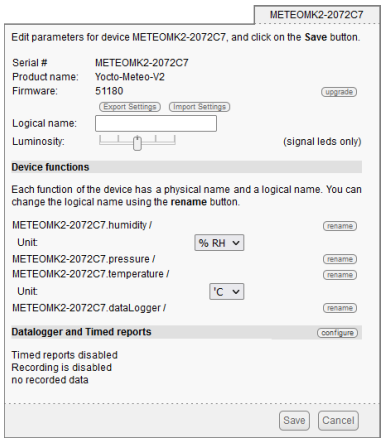
En général, vous n'êtes pas obligé d'avoir une version de YoctoHub-Ethernet plus récente que le module que vous voulez tester/configurer: la plupart des éléments spécifiques aux interfaces des modules sont stockés dans le firmware des modules eux-même. Il y a toutefois quelques exceptions, donc si vous rencontrez une erreur dans l'interface Web d'un module, vérifiez si une mise à jour de YoctoHub-Ethernet est disponible, et le cas échéant installez-là. Il peut ensuite être nécessaire de recharger la page dans le navigateur avec Shift-Reload, ou de vider le cache de votre navigateur, afin de forcer la mise à jour du code JavaScript.




*Fenêtre "détails" du module Yocto-Meteo-V2*

## 5.3. Configuration des modules

Vous pouvez configurer un module en cliquant sur le bouton **configure** correspondant dans l'interface principale, une fenêtre spécifique au module s'ouvre alors. Cette fenêtre permet au minimum de donner un nom logique au module ainsi que de mettre à jour son firmware. Reportez-vous au manuel du module correspondant pour plus de détails.




*Fenêtre "configure" du module Yocto-Meteo-V2*

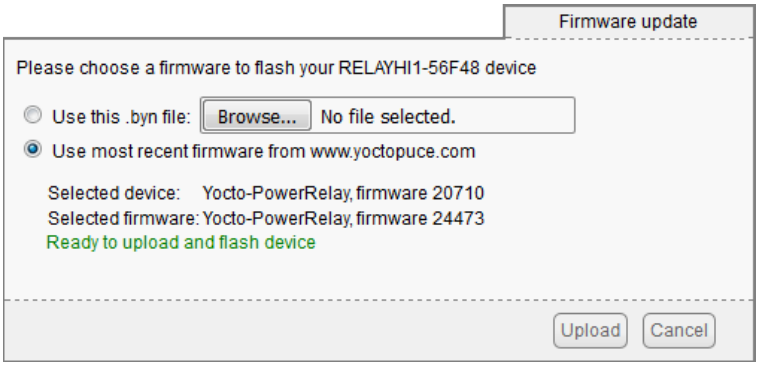
# 5.4. Mise à jour des firmwares

Les modules Yoctopuce sont en fait de véritables ordinateurs, ils contiennent même un petit serveur web. Et comme tous les ordinateurs, il est possible de mettre à jour leur logiciel de contrôle (firmware). Des nouveaux firmwares pour chaque module sont régulièrement publiés, ils permettent généralement d'ajouter de nouvelles fonctionnalités au module, et/ou de corriger d'éventuels bugs[2].

## Méthode recommandée

Pour mettre à jour le firmware d'un module, il suffit de d'ouvrir, dans l'interface de YoctoHub-Ethernet, la fenêtre de configuration du module à mettre à jour, puis cliquer sur le bouton **upgrade**. Si vous cliquez simplement sur le bouton **Update**, YoctoHub-Ethernet utilisera la version le plus récente du firmware publiée sur le site Web de Yoctopuce et l'installera.

YoctoHub-Ethernet vous permet aussi de choisir un fichier .byn que vous avez préalablement téléchargé depuis le site web de Yoctopuce, par exemple pour réinstaller une version antérieure.




*Fenêtre de mise à jour du firmware*

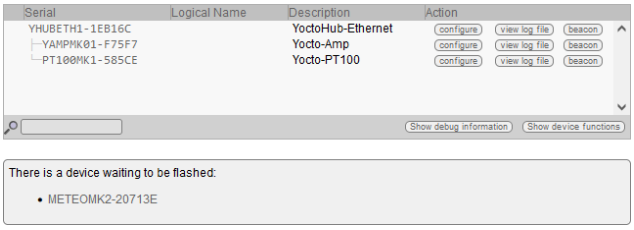
Une fois que vous avez cliqué sur **Update**, tout est automatique, YoctoHub-Ethernet fait redémarrer le module en mode "mise à jour", met à jour le firmware, puis rédémarre le module en mode normal. Les réglages de configuration du module seront préservés. Ne débranchez pas le module pendant la procédure de mise à jour.

## Méthode alternative

Si la mise à jour d'un module se passe mal, en particulier si le module a été débranché pendant le processus, il risque fort de ne plus fonctionner et de ne plus apparaître dans la listes des modules. Dans ce cas débranchez-le, attendez quelques secondes, et rebranchez-le en maintenant le Yocto-bouton appuyé. Cela a pour effet de faire démarrer le module en mode "mise à jour". Ce mode de fonctionnement est protégé contre les corruptions et devrait toujours être accessible. Une fois le

[2] Ne faites jamais confiance à des gens qui vous disent que leur logiciel n'a pas de bug :-)

module rebranché, provoquez un rafraîchissement de la liste des modules dans l'interface de YoctoHub-Ethernet et votre module devrait être listé dans le bas de l'interface. Cliquez dessus pour mettre à jour son firmware. Ce mode de mise à jour est une procédure de récupération, elle ne sauvegarde pas les réglages du module.




*Les modules en mode "mise à jour" sont listés dans l'interface.*

La mise à jour normale préserve la configuration des modules, en revanche la méthode alternative avec le Yocto-bouton remet le module dans sa configuration d'usine. Ce qui fait que vous pouvez aussi utiliser cette méthode pour réinitialiser complètement un module.

## Par ligne de commande ou programmation

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

L'exemple suivant, qui utilise la librairie Yoctopuce en ligne de commande, télécharge automatiquement les derniers firmwares depuis le site web de Yoctopuce et met à jour tous les modules Yoctopuce connectés par USB:

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

Ce second exemple installe le firmware LIGHTMK3.51180.byn, stocké dans le répertoire local C:\tmp \yfirmware, sur le module dont le numéro de série est LIGHTMK3-23BBDF. Les fichiers de firmware peuvent être téléchargés manuellement depuis le site web de Yoctopuce[3].

```
C:\>ymodule LIGHTMK3-23BBDF updateFirmware C:\tmp\yfirmware\LIGHTMK3.51180.byn
OK: LIGHTMK3-23BBDF.module.updateFirmware = 36% Wait for device.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 50% Flash zone.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 57% Flash zone.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 64% Flash zone.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 72% Flash zone.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 73% Device info retrieved.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 90% Firmware updated.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 100% success.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 100% Firmware Updated Successfully in: 14.305s.
```

Par défaut, `downloadAndUpdate` met à jour les firmwares uniquement vers une version plus récente, grâce à un second argument optionnel, `onlyNew`, qui est toujours vrai s'il est omis. Si vous voulez installer un firmware plus ancien (downgrade), vous devez passer le firmware comme premier paramètre et `false` comme deuxième.

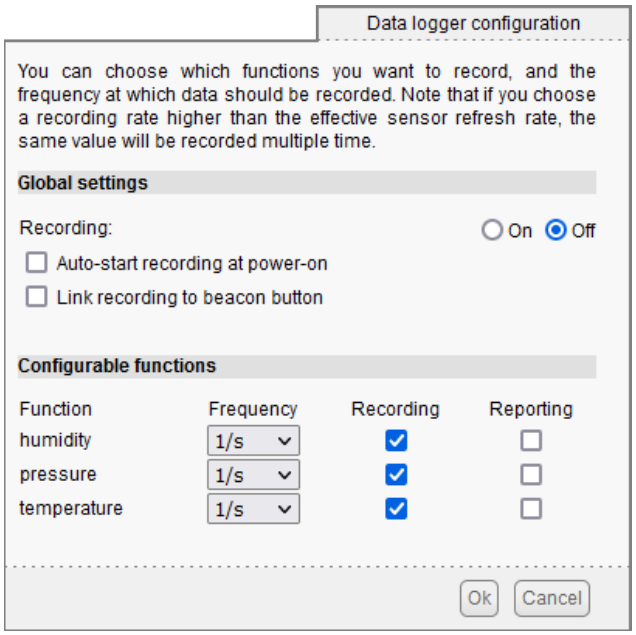
Il est également possible de mettre à jour le firmware de vos modules en utilisant la librairie de programmation Yoctopuce, en particulier à l'aide des méthodes `YModule.checkFirmware` et `YModule.updateFirmware`. Vous trouverez plus d'information à ce sujet dans les chapitres de programmation avancée figurant dans la documentation de chaque module.

[3] www.yoctopuce.com/FR/firmwares.php

## 5.5. Accès à l'enregistreur de données des capteurs

Pour tous les capteurs Yoctopuce qui incluent un enregistreur de données, la fenêtre de configuration inclut une section spéciale permettant de configurer l'enregistrement et de charger les données brutes contenues dans l'enregistreur, module par module.

Cliquez sur le bouton **configure** de la section **Datalogger and Timed reports**




*Fenêtre de configuration du datalogger*

Il est aussi possible d'installer comme plug-in de YoctoHub-Ethernet l'outil *Yocto-Visualization (for Web)*, qui offre des possibilité bien plus étendues pour visualiser les données sous forme de graphiques, et charger un fichier CSV correspondant à tous les capteurs connectés. Pour plus de détails, référez-vous au chapitre qui est consacré à l'installation de *Yocto-Visualization (for Web)* à la fin de ce manuel.

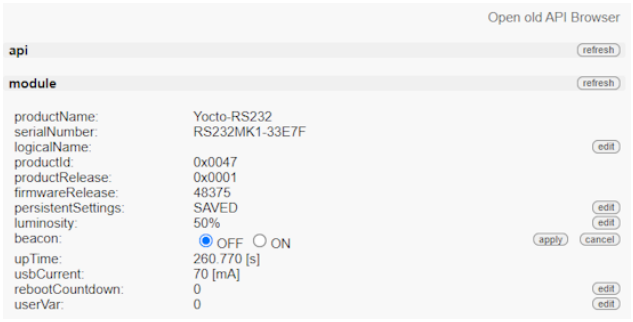
## 5.6. Passerelle REST

Vous pouvez également utiliser YoctoHub-Ethernet comme passerelle REST. Cela consiste à envoyer au module des requêtes HTTP à travers YoctoHub-Ethernet.

Pour expérimenter cette fonctionnalité, utilisez le lien **Open API Browser** disponible en bas de la fenêtre d'interface de votre module, à l'aide d'un navigateur Web.

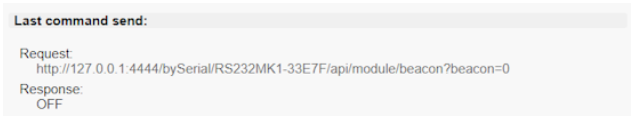

*Lien pour ouvrir l'interface REST*

Dans la fenêtre qui s'ouvre, vous pouvez alors modifier chaque attribut du module, à l'aide du bouton **edit**, puis en appliquant votre changement avec le bouton **apply**:




*Modification d'un attribut manuellement*

Après avoir effectué un changement, si vous descendez tout en bas de la page, vous verrez la requête HTTP qui a été effectuée pour appliquer le changement demandé:




*Requête HTTP correspondante*

Vous pourrez ainsi facilement découvrir comment accéder aux fonctions essentielles des modules par des requêtes HTTP: c'est là tout l'intérêt d'une interface REST. Si la sémantique d'un attribut en particulier vous échappe, vous trouverez des explications dans le manuel du module.

## 5.7. Passerelle OpenMetrics (Prometheus)

Il est aussi possible d'utiliser YoctoHub-Ethernet comme source de données pour un serveur Prometheus, afin de centraliser les mesures des capteurs Yoctopuce dans la même base de donnée que les informations sur l'état des infrastructures informatiques.

Les sources de données Prometheus sont appelées des exportateurs: l'idée est que chaque système ou service important peut mettre à disposition de Prometheus ses données vitales, permettant à l'administrateur système de les collecter et de les surveiller. Chaque exportateur est une URL accessible par HTTP, qui fournit ses données selon le standard OpenMetrics: une mesure par ligne, avec des conventions de nommage et de classification qui permettent à l'administrateur système de s'y retrouver parmi les centaines de mesures qu'il aura à disposition lorsqu'il configure son tableau de bord.

YoctoHub-Ethernet a la capacité d'être un exportateur OpenMetrics par l'intermédiaire de l'interface REST. Pour obtenir des données dans le format OpenMetrics, il suffit de charger l'URL `/api/services.om` de votre YoctoHub-Ethernet, soit `http://127.0.0.1:4444/api/services.om`. Par exemple, si vous accéder à cette URL alors que quelques capteurs sont connectés en local, vous obtiendrez quelque chose du genre (la présentation a été ici modifiée pour faciliter la lecture, mais en réalité chaque mesure tient sur une seule ligne):

```
yocto_temperature_advertisedValue{
    productName="Yocto-Thermocouple",
    serialNumber="THRMCPL1-16397A",
    deviceName="insideProbes",
    functionId="temperature1"} 21.78
yocto_temperature_advertisedValue{
    productName="Yocto-PT100",
    serialNumber="PT100MK1-BA496",
    functionId="temperature"} 28.57
yocto_temperature_advertisedValue{
    productName="Yocto-RangeFinder",
    serialNumber="YRNGFND1-1D1567",
    deviceName="rf",
    functionId="temperature1",
    functionName="rfTemp"} 25.13
yocto_lightSensor_advertisedValue{
    productName="Yocto-RangeFinder",
    serialNumber="YRNGFND1-1D1567",
    deviceName="rf",
```

```
    functionId="lightSensor1"} 56
yocto_rangeFinder_advertisedValue{
    productName="Yocto-RangeFinder",
    serialNumber="YRNGFND1-1D1567",
    deviceName="rf",
    functionId="rangeFinder1"} 1456
# EOF
```

Pour indiquer à Prometheus de collecter ces données directement depuis votre YoctoHub-Ethernet, il suffit donc d'ajouter à votre fichier `prometheus.yml` une section comme celle-ci:

```
  - job_name: "yoctohub_sensors"
    scrape_interval: 60s
    metrics_path: "/api/services.om"
    static_configs:
      - targets: ['127.0.0.1:4444']
```

Comme l'exportateur OpenMetrics est intégré au coeur de l'interface REST de YoctoHub-Ethernet, vous pouvez aussi l'utiliser pour obtenir des informations plus détaillées par capteur, en utilisant une URL qui pointe vers un capteur spécifique. Par exemple, si vous connectez un Yocto-Thermocouple au YoctoHub-Ethernet et que vous lui donnez le nom logique `tcProbes`, l'URL `/byName/tcProbes/api.om` donnera une réponse du genre:

```
yocto_module_luminosity{...,functionId="module",functionName="tcProbes"} 50
yocto_module_beacon{...,functionId="module",functionName="tcProbes"} 0
yocto_module_usbCurrent_mA{...,functionId="module",functionName="tcProbes"} 23
yocto_module_rebootCountdown{...,functionId="module",functionName="tcProbes"} 0
yocto_module_userVar{...,functionId="module",functionName="tcProbes"} 0
yocto_temperature_currentValue_degC{[...],functionName="heatSink"} 21.99
yocto_temperature_lowestValue_degC{[...],functionName="heatSink"} 20.51
yocto_temperature_highestValue_degC{[...],functionName="heatSink"} 22.25
yocto_temperature_currentRawValue_degC{[...],functionName="heatSink"} 21.988
yocto_temperature_signalValue_mV{[...],functionName="heatSink"} -0.162
yocto_temperature_signalValue_mV{[...],functionId="temperature2"} 999.999
yocto_dataLogger_currentRunIndex{[...],functionId="dataLogger"} 0
yocto_dataLogger_autoStart{[...],functionId="dataLogger"} 0
yocto_dataLogger_beaconDriven{[...],functionId="dataLogger"} 0
yocto_dataLogger_usage{[...],functionId="dataLogger"} 0
# EOF
```

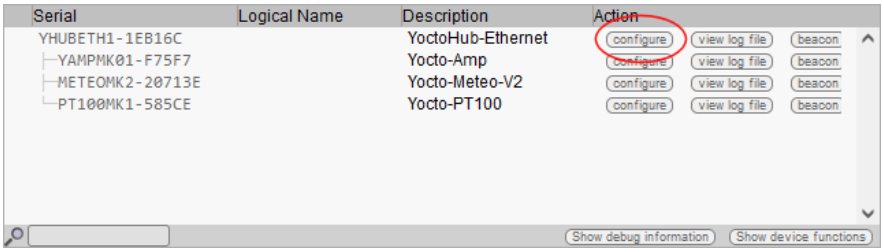
Ainsi, tous les attributs numériques du Yocto-Thermocouple sont mis à disposition. Vous pouvez donc connaître les valeurs min/max rencontrées, la tension mesurée aux bornes du thermocouple, etc. Notez aussi que dans ce cas, le symbole exporté inclut l'unité, comme recommandé par OpenMetrics. Lorsque le module détecte qu'une entrée n'est pas connectée (comme la fonction `temperature2` ci-dessus), les métriques qui ne peuvent être calculées sont automatiquement supprimées pour qu'elles soient manquantes, plutôt que de garder la dernière valeur mesurée.

Pour obtenir ces données supplémentaire par capteur, il suffit donc d'ajouter au fichier `prometheus.yml` une section supplémentaire, référençant le capteur soit par son numéro de série (*bySerial*) soit par son nom logique (*byName*):

```
  - job_name: "thermocouple_probes"
    scrape_interval: 60s
    metrics_path: "/byName/tcProbes/api.om"
    static_configs:
      - targets: ['127.0.0.1:4444']
```

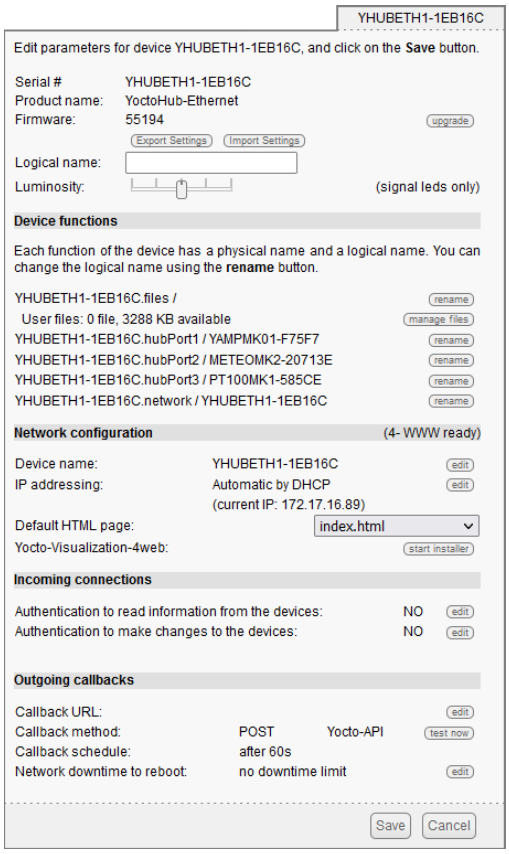
# 6. Contrôle d'accès

YoctoHub-Ethernet vous permet d'instaurer un contrôle d'accès à vos modules Yoctopuce. Pour ce faire cliquez simplement sur le bouton **Configure** de la ligne du YoctoHub-Ethernet dans l'interface.




*Cliquez sur le bouton **configure** de la première ligne*

Cela aura pour effet de faire apparaître la fenêtre de configuration de YoctoHub-Ethernet.




*La fenêtre de configuration de YoctoHub-Ethernet*

Ce contrôle d'accès est contrôlé depuis la section **Incoming connections**. Il peut se faire à deux niveaux distincts.

## 6.1. Accès "admin"

Le mot de passe *admin* verrouille les accès en écriture sur les modules. Lorsqu'il est configuré, seuls les accès de type *admin* permettent d'accéder aux modules en lecture et en écriture. Les utilisateurs utilisant le login *admin* pourront éditer la configuration des modules vus par YoctoHub-Ethernet comme ils le souhaitent.

## 6.2. Accès "user"

Le mot de passe *user* verrouille toute utilisation des modules. Lorsqu'il est configuré, toute utilisation sans mot de passe devient impossible.

Si vous configurez uniquement un mot de passe *user* sans configurer de mot de passe *admin*, tous les utilisateurs devront donner un mot de passe pour accéder aux modules, mais une fois autorisés, ils pourront aussi éditer la configuration des modules.

Si vous configurez simultanément un contrôle d'accès de type *user* et de type *admin*, les utilisateurs utilisant le login *user* ne pourront pas modifier la configuration des modules vus par YoctoHub-Ethernet. Les accès de type *user* ne permetteront d'accéder aux modules qu'en lecture seule, c'est-à-dire seulement pour consulter l'état des modules. Seuls les utilisateurs qui utilisent le login *admin* pourront changer la configuration des modules.

Si vous configurez uniquement un accès *admin* sans configurer d'accès *user*, tous les utilisateurs pourront continuer à consulter vos modules en lecture sans avoir à entrer de mot de passe, et seuls ceux qui connaîtront le mote de passe *admin* pourront changer la configuration des modules.

## 6.3. Influence sur les API

Attention, le contrôle d'accès agira aussi sur les API Yoctopuce qui tenteront de se connecter à YoctoHub-Ethernet. Dans les API Yoctopuce, la gestion des droits d'accès est réalisée au niveau de l'appel à la fonction `RegisterHub()` : vous devrez donner l'adresse de YoctoHub-Ethernet sous la forme *login:password@adresse:port*, par exemple:

```
YAPI.RegisterHub("admin:mypass@192.168.1.2:4444",errmsg);
```

# 7. Envoi de données vers l'extérieur

YoctoHub-Ethernet est capable de se connecter à des services externes pour communiquer l'état des modules qui lui sont raccordés.

YoctoHub-Ethernet sait comment poster ses données au format accepté par quelques services Cloud tiers, tels que

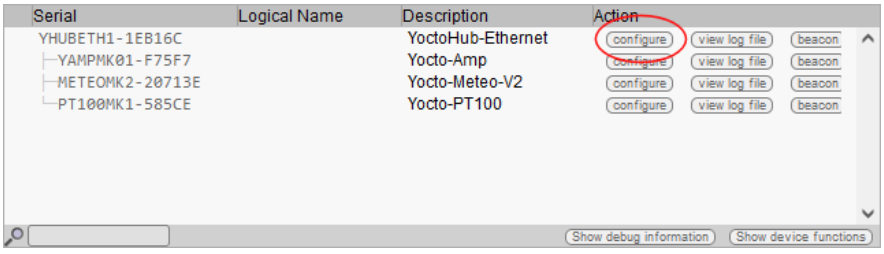
- Emoncms
- InfluxDB (versions 1.0 et 2.0)
- PRTG
- Valarm.net

YoctoHub-Ethernet peut aussi se connecter à des services externes à l'aide de protocoles avancés qui permettent une interaction plus poussée avec les modules Yoctopuce, mais qui vous demanderont un peu plus de connaissances pour pouvoir en tirer parti:

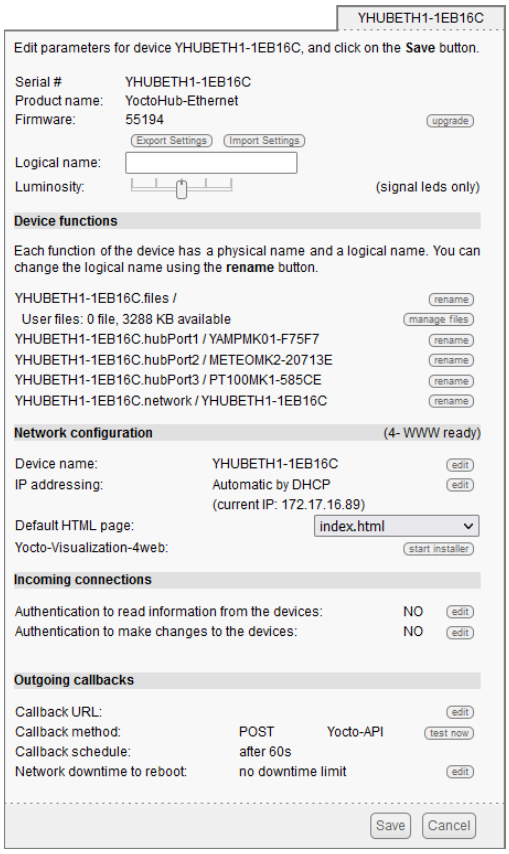
- MQTT
- Yocto-API

## 7.1. Configuration

Pour utiliser cette fonctionnalité, cliquez simplement sur le bouton **configure** de la ligne correspondant à YoctoHub-Ethernet dans l'interface, puis cliquez sur le bouton **edit** de la section **Outgoing callbacks**.




*Cliquez sur le bouton **configure** correspondant*




*Puis éditez la section Outgoing callbacks*

La fenêtre de configuration des callbacks apparaît. Cette fenêtre vous permet de définir comment YoctoHub-Ethernet peut interagir avec un serveur web externe. Vous avez plusieurs type d'interactions à votre disposition.

# 7.2. Callbacks HTTP vers des services tiers

YoctoHub-Ethernet est capable de poster sur des serveurs externes les valeurs des capteurs Yoctopuce à intervalles régulier et/ou à chaque fois qu'une valeur change de manière significative. Cette fonctionnalité vous permettra de stocker vos mesures et de tracer des graphiques sans écrire la moindre ligne de code.

Yoctopuce n'est en aucune manière affilié à ces services tiers et ne peut donc ni garantir leur pérennité, ni proposer des améliorations à ces services.

## Emoncms

Emoncms est un service de Cloud open-source qui permet de poster les données des capteurs Yoctopuce et ensuite de les visualiser. Il est aussi possible d'installer son propre serveur en local.

Les paramètres à fournir sont la clé d'API Emoncms, le numéro de nœud que vous désirez utiliser, ainsi que l'adresse du serveur Emoncms si vous utilisez un serveur local.

Il est possible de personaliser les noms associés aux mesures postées sur Emoncms. Pour plus de détails, voir le paragraphe intitulé "Noms associés aux valeur postées" ci-dessous.

## InfluxDB 1.0 and 2.0

InfluxDB est une base de données open-source dédiée spécifiquement à stocker des séries temporelles de mesures et d'événements. Notez que seules les installations locales sont supportées. En effet, le service *InfluxDB Cloud* n'est pas supporté car il nécessite une connexion SSL.

Les paramètres pour la version 1.0 d'InfluxDB sont l'adresse du serveur et le nom de la base de données.

La version 2.0 d'InfluxDB utilise une API différente et le YoctoHub a besoin de trois paramètres (`organization`, `bucket` et `token`) ainsi que l'adresse du serveur.

Il est possible de personaliser les noms associés aux mesures postées sur InfluxDB. Pour plus de détail, voir le paragraphe intitulé "Noms associés aux valeur postées" ci-dessous.

### PRTG

PRTG est une solution commerciale, destinée à la supervision des systèmes et des applications. Il est possible d'enregistrer les mesures et obtenir des graphiques de vos capteurs avec ce service.

Les paramètres à fournir sont l'adresse du serveur PRTG et le `token` qui permet d'identifier YoctoHub-Ethernet.

Il est possible de personaliser les noms associés aux mesures postées sur PRTG. Pour plus de détail, voir le paragraphe intitulé "Noms associés aux valeur postées" ci-dessous.

### Valarm.net

Valarm est un service de Cloud professionnel qui permet d'enregistrer les données des capteurs Yoctopuce mais permet aussi des fonctions plus élaborées comme la possibilité de géolocaliser les mesures ou de configurer les modules Yoctopuce à distance.

Le seul paramètre à fournir est un `Routing code` qui permet d'identifier YoctoHub-Ethernet.

## 7.3. Callbacks vers un broker MQTT

MQTT est un protocole de l'Internet des Objets permettant à des capteurs et des actuateurs de communiquer entre eux, via un serveur central appelé *broker MQTT*. MQTT est particulièrement utilisé en domotique, où il permet de fédérer de nombreuses technologies pour les rendre accessible à un système de contrôle central comme *Home Assistant*.

Les paramètres de base à fournir pour la configuration du callback MQTT sont l'adresse du broker MQTT, le client ID, le *root_topic* ainsi que les paramètres d'authentification. Notez que l'encapsulation du protocole MQTT dans une connexion SSL n'est pas supportée, ce qui exclut son utilisation avec les services comme AWS IoT Core.

Lorsqu'un callback MQTT est actif, YoctoHub-Ethernet est capable de publier des messages avec l'état des capteurs et actuateurs, et recevoir des messages de commande et de configuration, ce qui permet au système de contrôle central d'interagir pleinement avec les modules.

Le reste de cette section décrit en détail les messages MQTT supportés. Elle n'intéressera que les développeurs qui désirent développer leur propre intégration avec des modules Yoctopuce via MQTT. Si vous comptez simplement utiliser *Home Assistant*, vous pouvez sauter cette section et grâce au mécanisme *MQTT Discovery*, vos modules devraient automatiquement apparaître dans *Home Assistant*.

### Racine commune des messages

Le *topic* de tous les messages commence par une partie commune, qui identifie le module Yoctopuce et la fonction particulière de ce module concernée par le message. Elle a la structure suivante:

***root_topic*/*deviceName*/*functionName***

Le *root_topic* peut être configuré librement, par exemple à la valeur `yoctopuce`. Si vous connectez plusieurs hubs au même *broker* MQTT, vous pouvez soit utiliser le même *root_topic* pour tous, soit un topic différent par hub. L'utilisation d'un *root_topic* distinct est recommandée si le hub est destiné à recevoir beaucoup de commandes par MQTT.

Le *deviceName* correspond au nom logique que vous avez donné au module Yoctopuce concerné. Si aucun nom logique n'a été configuré, le numéro de série du module est utilisé à la place du nom logique (par exemple `METEOMK2-012345`).

Le *functionName* correspond au nom logique que vous avez donné à la fonction concernée. Si aucun nom logique n'a été configuré, l'identifiant de la fonction est utilisé (par exemple `genericSensor1`).

Le fait d'utiliser les noms logique plutôt que les noms matériels dans la racine du *topic* a l'avantage de permettre d'identifier les modules et les fonctions par leur rôle et de ne pas devoir indiquer explicitement l'identifiant matériel de chaque module au client MQTT qui devra interagir avec ces modules. Le désavantage est que si vous décidez de changer le nom logique de vos modules ou de vos fonctions sans y penser, les *topics* MQTT utilisés changeront en conséquence.

## Topic /api: état complet de la fonction

Sous `root_topic/deviceName/functionName/api`, chaque fonction publie une structure JSON décrivant l'état complet de la fonction, tous attributs compris. Dans cet encodage JSON,

- les booléens sont représentés par 0 et 1
- les types énumérés sont représentés par des constantes numériques
- les nombres réels tels que les mesures sont représentés sous forme d'entiers, après multiplication par 65536. Il faut donc les diviser par 65536.0 pour obtenir la valeur réelle.

Ce message est publié lorsque l'une des conditions suivante se produit:

- à l'établissement de la connexion du hub avec le broker MQTT
- toutes les cinq minutes
- après un changement de configuration du module
- en réponse à la réception d'une commande par cette fonction
- pour la fonction *module*, en cas d'activation ou de désactivation de la balise (*beacon*)

## Topic de base: état instantané

Sous `root_topic/deviceName/functionName`, chaque fonction publie un résumé textuel de son état. Il ne s'agit pas de JSON mais d'une simple chaîne de caractères, correspondant à la valeur de l'attribut *advertisedValue* de la fonction. Par exemple, pour un capteur, il correspond à la valeur instantanée du capteur, alors que pour un relais il correspond à la lettre A ou B en fonction de l'état de commutation.

Ce message est publié lorsque l'une des conditions suivante se produit:

- à l'établissement de la connexion du hub avec le broker MQTT
- toutes les cinq minutes
- à chaque changement de l'attribut *advertisedValue*

Pour éviter de surcharger le broker MQTT avec les changements de valeurs instantanée des capteurs, il est possible de désactiver globalement l'envoi des messages de valeurs instantanée pour les capteurs uniquement, dans la configuration MQTT.

## Topics /avg, /min, /max: valeurs moyenne et extrêmes

Sous `root_topic/deviceName/functionName/avg`, les fonctions de type capteur (sous-classes de *Sensor*) publient périodiquement la valeur moyenne observée durant l'intervalle de temps précédent, directement sous forme de nombre réel.

Sous `root_topic/deviceName/functionName/min`, la valeur minimale observée durant l'intervalle de temps précédent.

Sous `root_topic/deviceName/functionName/max`, la valeur maximale observée durant l'intervalle de temps précédent.

Ces messages sont l'équivalent direct des *timed reports* documentés dans le manuel de ces module. L'intervalle de temps doit avoir été configuré préalablement dans l'attribut *reportFrequency*. Dans le cas contraire, ces messages ne sont pas envoyés.

### Topics /set/*attributeName*: envoi de commande et configuration

Sous `root_topic/deviceName/functionName/set/`***attributeName***, il est possible d'envoyer des message pour modifier les attributs des fonctions, dans le but de modifier leur état ou leur configuration. La valeur du message correspond à la nouvelle valeur désirée, telle quelle. Le format est identique à celui utilisé par la asserelle REST de YoctoHub-Ethernet (voir la section "Passerelle REST" de ce manuel).

Par exemple, on pourrait commuter un relais en envoyant un message au *topic*
`yoctopuce/RelaiPompe/relay1/set/state`
avec la valeur 1.

On pourrait aussi déclencher une impulsion de 1500ms sur le même relais en envoyant un message au *topic*
`yoctopuce/RelaiPompe/relay1/set/pulseTimer`
avec la valeur 1500.

La réception de commande et de changements de configuration par MQTT doit avoir été activée explicitement dans la configuration MQTT sur le hub Yoctopuce. Par sécurité, le comportement de base du mode MQTT reste le mode en lecture seule.

### Topic /rdy: état de connectivité

Sous `root_topic/deviceName/module/rdy`, la fonction module publie une indication binaire de l'état de disponibilité du module. La valeur est à 1 lorsque le module est en ligne, et à 0 lorsqu'il est hors ligne.

Ce message est publié par le hub pour son propre module lorsque l'une des conditions suivante se produit:

- à l'établissement de la connexion du hub avec le broker MQTT
- à la déconnexion du hub du broker MQTT

Ce message est publié pour les modules autres que le hub lorsque l'une des conditions suivante se produit:

- à l'établissement de la connexion du hub avec le broker MQTT
- au branchement d'un module sur le hub
- au débranchement d'un module du hub

Pour déterminer si un module est réellement atteignable, il faut donc vérifier son propre topic `/rdy`, pour savoir si il a été déconnecté, et le topic `/rdy` du hub, pour savoir si la connexion MQTT est active.

### MQTT discovery

De plus, des messages particuliers sont publiés sous le topic `homeassistant/` juste après l'établissement de la connexion du hub avec le broker MQTT, et répétés toutes les 5 minutes, pour permettre la détection automatique des fonctionalités offertes, grace au mécanisme *MQTT discovery* supporté par Home Assistant et openHab.

## 7.4. Callbacks de type Yocto-API

Les callbacks de type Yocto-API utilisent un protocole spécifique défini par Yoctopuce, qui permet une interaction très poussée avec les modules Yoctopuce. A l'aide de certains langages commes PHP, TypeScript, JavaScript ou Java, ils permettent au programmeur du service Web d'utiliser directement les fonctions de la librairie de programmation Yoctopuce pour interagir avec les modules qui se connectent par callback HTTP. Cela permet en particulier de contrôler depuis un site web public des modules Yoctopuce installés derrière un router ADSL privé. Il est par exemple possible de commuter la sortie d'un relais en fonction de la valeur d'un capteur, tout en gardant le contrôle complet du système sur un serveur Web.

Yoctopuce met à disposition une application gratuite qui exploite au maximum les possibilités du Callback Yocto-API sur un serveur PHP: VirtualHub for Web. Cette application web permet d'interagir à distance avec les modules qui se connectent périodiquement via un Callback Yocto-API. De plus amples informations sur VirtualHub for Web sont disponibles sur le blog de Yoctopuce [1].

En mode Callback Yocto-API ou Yocto-API-JZON, il est possible de choisir entre les protocoles "HTTP" et "WebSocket".

### Callbacks Yocto-API en mode WebSocket

Lors d'une requête HTTP usuelle, le flux d'information est extrêmement simple: le client envoie une requête et écoute la réponse du serveur. Il ne s'agit pas à proprement parler d'une conversation, mais simplement d'une réponse à une question. Le client HTTP ne peut pas répondre à ce que le serveur lui a dit sans recommencer une nouvelle communication séparée.

Il y a néanmoins dans le standard HTTP 1.1 une porte ouverte vers une amélioration: le client peut demander d'upgrader le protocole de communication. Une méthode d'upgrade qui s'est standardisée s'appelle les WebSockets et est définie dans le RFC 6455. Cette upgrade transforme le simple canal question/réponse en un lien bidirectionnel permettant d'échanger des messages quelconques dans les deux directions.

Cette transformation de la connection HTTP exige que les deux parties en soient capables. YoctoHub-Ethernet et les librairies de programmation Yoctopuce le sont. Mais pour pouvoir transformer un callback HTTP en callback WebSocket, vous aurez aussi besoin d'un serveur Web basé sur une technologie qui permet l'upgrade de connection. C'est le cas par exemple de Java et Node.JS. Par contre, les implémentations de PHP sur Apache n'en sont à ce jour pas capables.

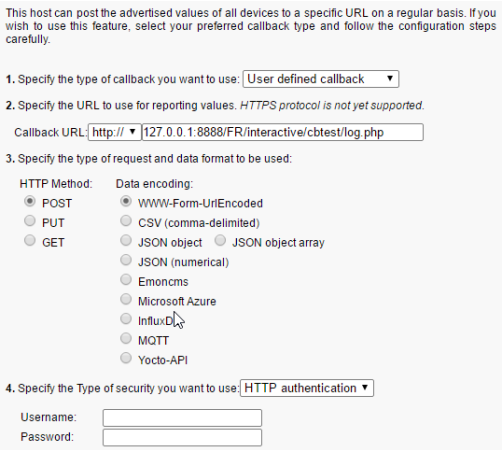
L'utilisation de WebSockets permet d'accéder à plusieurs fonctionnalités avancées des librairies Yoctopuce qui ne sont pas disponibles via un callback HTTP:

- un callback WebSocket peut énumérer et récupérer des données stockées dans l'enregistreur de données intégré dans chaque senseur Yoctopuce.
- un callback WebSocket peut utiliser les fonctions de communication bidirectionnelles des modules séries, par exemple pour exécuter des requêtes MODBUS avec un Yocto-RS485.
- un callback WebSocket peut profiter des notifications instantanées de changement de valeur par callback, et des notifications périodiques de valeurs moyennées des capteurs.
- un callback WebSocket peut garder une connection persistante entre le hub et le serveur, par exemple pour implémenter une interaction avec un utilisateur.
- un callback WebSocket peut même être utilisé pour effectuer une mise à jours de firmware.

## 7.5. Callbacks HTTP définis par l'utilisateur

Si aucune des autres options proposées pour la configuration de callback HTTP ne convient à vos besoins, vous pouvez essayer de spécifier vous-même la manière dont les données doivent être transmises. Les "*User defined callback*" vous permettent de personnaliser la manière dont YoctoHub-Ethernet envoie les informations au serveur. Notez que seul le protocole HTTP est supporté (pas de HTTPS).

[1] https://www.yoctopuce.com/FR/article/nouveau-un-virtualhub-qui-fonctionne-a-travers-le-web




*La fenêtre de configurations des callbacks*

Si vous désirez protéger votre script de callback, vous pouvez configurer un contrôle d'accès HTTP standard sur le serveur Web. YoctoHub-Ethernet sait comment gérer les méthodes standard d'identification de HTTP: indiquez simplement le nom d'utilisateur et le mot de passe nécessaires pour accéder à la page. Il est possible d'utiliser la méthode "Basic" aussi bien que la méthode "Digest", mais il est recommandé d'utiliser la méthode "Digest", car elle est basée sur un protocole de question-réponse qui évite la transmission du mot de passe sur le réseau et évite aussi les copies d'autorisation.

A titre d'exemple, voici un script PHP qui vous permettra de visualiser dans la fenêtre de debug le contenu des données postées par un callback HTTP défini par l'utilisateur en mode POST et WWW-Form-UrlEncoded.

```
<?php
  Print(Date('H:i:s')."\r\n");
  foreach ($_POST as $key => $value) {
      Print("$key=$value\r\n");
  }
?>
```

Il est possible de personaliser les noms associés aux mesures postées par un callback HTTP défini par l'utilisateur. Pour plus de détail, voir le paragraphe intitulé "Noms associés aux valeur postées" ci-dessous.

# 7.6. Noms associés aux valeur postées

A l'exception des callbacks de type Yocto-API qui donnent accès à la totalité des informations sur les modules Yoctopuce, les callbacks HTTP sont conçus pour ne transmettres que les informations les plus importantes au serveur, en associant chaque valeur avec un nom qui permette facilement de le rattacher à son origine.

## Comportement de base

Le comportement standard est de transmettre la valeur de l'attribut `advertisedValue` pour chaque fonction présente sur les modules Yoctopuce. Le nom associé automatiquement à chaque valeur suit la logique suivante:

1. Si un nom logique a été défini pour une fonction:

```
NOM_LOGIQUE_DE_LA_FONCTION = VALEUR
```

2. Si un nom logique a été défini pour le module, mais pas pour la fonction:

```
NOM_DU_MODULE.NOM_HARDWARE = VALUE
```

3. Si aucun nom logique n'a été attribué:

```
NUMERO_DE_SERIE.NOM_HARDWARE = VALEUR
```

La manière la plus simple pour personnaliser les noms associés aux valeurs consiste donc à configurer le nom désiré comme nom logique de la fonction, ou sinon comme nom logique du module lui-même.

Voici un exemple des données postées par un callback HTTP défini par l'utilisateur en mode POST et au format *JSON (numerical)* pour un système comportant un Yocto-Watt où chaque fonction a reçu un nom logique explicite (par exemple `VoltageDC`) et un Yocto-Meteo-V2 où c'est au module lui-même qu'on a donné le nom logique `Ambiant`:

```
{"timestamp":1678276738,
"CurrentAC":0
,"CurrentDC":0
,"VoltageAC":0
,"VoltageDC":0
,"Power":0
,"Ambiant.temperature":22.17
,"Ambiant.pressure":949.36
,"Ambiant.humidity":30
}
```

## Personnalisation avancée par un fichier

Si l'on désire une personnalisation plus poussée du format des données transmises, pour sélectionner spécifiquement quelle attribut de quel module doit être envoyé sous quel nom, ou pour y rajouter des informations contextuelles, c'est aussi possible, mais c'est un peu plus compliqué. Il faut à ce moment créer un fichier modèle définissant le format exact des données à envoyer. Le contenu et le nom de ce fichier est donc spécifique à chaque type de callback HTTP, et chaque cas sera expliqué individuellement ci-dessous.

Le point commun de tous les fichiers de modèle est que leur contenu sera envoyé tel quel au serveur, à l'exception des expressions englobées entre *accents graves* (le caractère `, code ASCII 96, appelé *backquote* ou *backtick* en anglais) qui seront évaluées par la passerelle REST de YoctoHub-Ethernet (voir la section "Passerelle REST" de ce manuel). Par exemple, si le fichier de modèle comporte le texte:

```
{ "origin": "`/api/module/productName`" }
```

alors le contenu effectivement posté sera

```
{ "origin": "YoctoHub-Ethernet" }
```

## Fichier de personnalisation pour Emoncms

Le format de base utilisé par YoctoHub-Ethernet pour Emoncms a la forme ci-dessous. Notez que les données sont transmise dans l'URL, donc en une seule ligne, mais elles ont été mises ici sur plusieurs ligne pour faciliter la lecture.

```
time=1678277614
&json={
  "CurrentAC":0,
  "CurrentDC":0,
  "VoltageAC":0,
  "VoltageDC":0,
  "Power":0,
  "Ambiant.temperature":22.28,
  "Ambiant.pressure":949.54,
  "Ambiant.humidity":29.7
}
```

Pour personnaliser le format des données envoyées à Emoncms, il faut créer sur YoctoHub-Ethernet un fichier modèle de format portant le nom `EMONCMS_cb.fmt`.

Ce fichier ne doit comporter qu'une seule ligne, sans aucun retour de chariot, et commencer par une chaîne du type `&json=`. Par exemple, pour ne poster que l'humidité absolue et relative, vous pourriez utiliser (sans retour de chariot!):

```
&json={
  "absoluteHumidity"=`/byName/Ambiant/api/humidity/absHum`,
  "relativeHumidity"=`/byName/Ambiant/api/humidity/relHum`
}
```

## Fichier de personnalisation pour InfluxDB

Le format de base utilisé par YoctoHub-Ethernet pour InfluxDB a la forme ci-dessous. Il associe toutes les valeurs à une base de mesures *yoctopuce* et ajoute un tag *name* avec le nom sur le réseau de YoctoHub-Ethernet et un tag *ip* avec son adresse IP. Ensuite, chaque valeur est postée dans un champ dont le nom suit la convention de base décrite précédemment. Les données sont transmises par un POST de type CSV, en une seule ligne, mais elles ont été mises ici sur plusieurs lignes pour faciliter la lecture.

```
yoctopuce,name=VIRTHUB0-12345678,ip=192.168.1.10
  CurrentAC=0,CurrentDC=0,VoltageAC=0,VoltageDC=0,Power=0,
  Ambiant_temperature=22.5,Ambiant_pressure=948.63,Ambiant_humidity=29.4
  1678281649
```

Pour personnaliser le format des données envoyées à InfluxDB, il faut créer sur YoctoHub-Ethernet un fichier modèle de format portant le nom `INFLUXDB_cb.fmt` (pour la version 1.0), ou `INFLUXDB_V2_cb.fmt` (pour la version 2.0).

Ce fichier peut comporter plusieurs lignes si vous le désirez, ce qui vous permettre d'utiliser des tags différents pour différents mesures, ou même de ventiler des mesures sur plusieurs bases de données.

Par exemple, pour poster l'humidité absolue et relative simultanément, mais avec tes tags différents, vous pourriez utiliser le fichier de format suivant:

```
humidity,type=relative,location=Library relHum=`/byName/Ambiant/api/humidity/relHum`
humidity,type=absolute,location=Library absHum=`/byName/Ambiant/api/humidity/absHum`
```

Attention: le serveur InfluxDB n'accepte que les retours de chariot au format UNIX (caractère `\n`, aussi appelé LF). Si vous éditez le fichier sur une machine Windows, prenez soin d'utiliser un éditeur de texte capable de ne pas ajouter le retour de chariot Windows (`\r\n`, aussi appelé CR LF).

## Fichier de personnalisation pour PRTG

Le format de base utilisé par YoctoHub-Ethernet pour PRTG a la forme ci-dessous. Il poste chaque valeur dans un canal dont le nom suit la convention de base décrite plus procédemment. Les données sont transmises par un POST de type CSV, en une seule ligne, mais elles ont été mises ici sur plusieurs lignes pour faciliter la lecture.

```
{"prtg":{"result":[
{"channel":"CurrentAC","value":"0","float":"1","DecimalMode":"All"}
,{"channel":"CurrentDC","value":"0","float":"1","DecimalMode":"All"}
,{"channel":"VoltageAC","value":"0","float":"1","DecimalMode":"All"}
,{"channel":"VoltageDC","value":"0","float":"1","DecimalMode":"All"}
,{"channel":"Power","value":"0","float":"1","DecimalMode":"All"}
,{"channel":"Ambiant.temperature","value":"22.48","float":"1","DecimalMode":"All"}
,{"channel":"Ambiant.pressure","value":"948.68","float":"1","DecimalMode":"All"}
,{"channel":"Ambiant.humidity","value":"29.7","float":"1","DecimalMode":"All"}
]}}
```

Pour personnaliser le format des données envoyées à PRTG, il faut créer sur YoctoHub-Ethernet un fichier modèle de format portant le nom `PRTG_cb.fmt`.

Ce fichier doit comporter au minimum la même première et dernière ligne que l'exemple ci-dessus. La description des canaux pourra par contre être entièrement personnalisée.

Par exemple, pour poster l'humidité absolue et relative simultanément, dans deux canaux séparés, vous pourriez utiliser le fichier de format suivant:

```
{"prtg":{"result":[
{"channel":"relHum","value":"`/byName/Ambiant/api/humidity/relHum`",
  "float":"1","DecimalMode":"All"},
{"channel":"absHum","value":"`/byName/Ambiant/api/humidity/absHum`",
  "float":"1","DecimalMode":"All"}
]}}
```

## 7.7. Planification des callbacks

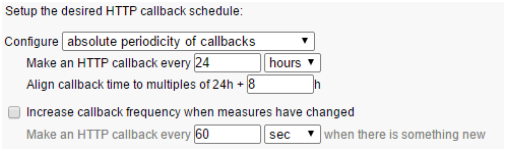
La section de plannification des callbacks est présente pour tous les types de callbacks. C'est la dernière section contenant des champs à remplir.

Un premier callback est toujours effectué quelques secondes après le lancement de YoctoHub-Ethernet. Pour les callbacks suivants, c'est le réglage de planification qui détermine la fréquence des callbacks.

Il est possible de choisir entre deux méthodes de planification: soit en configurant l'intervalle de temps entre deux callbacks consécutifs, soit en définissant une périodicité absolue, pour obtenir des callbacks à heure fixe. L'intervalle entre les callbacks peut être spécifié en secondes, en minutes ou en heures.

L'option `interval between subsequent callbacks` permet de spécifier le délais entre chaque callback. C'est-à-dire que si l'on configure un intervalle de 5 minutes, YoctoHub-Ethernet va attendre 5 minutes avant de déclencher le callback suivant. Si le premier callback est déclenché à 12h03, le suivant sera exécuté à 12h08, etc.

L'option `absolute periodicity of callbacks` permet de configurer des callbacks à heure fixe. C'est-à-dire que le callback est déclenché tous les multiples du délais configuré. Par exemple un délais de 5 minutes va déclencher un callback à 8h00, 8h05, 8h10, etc. Notez que dans ce mode il est aussi possible de spécifier un décalage par rapport au délais configuré. Par exemple avec un délais de 24h, il est possible d'utiliser un décalage de 8h pour déclencher le callback tous les jours à 8h du matin.




*Planification des callbacks*

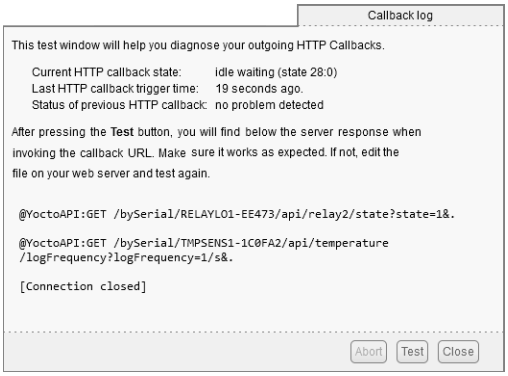
Vous pouvez choisir explicitement si vous désirez que la fréquence des callbacks varie lorsqu'aucune nouvelle mesure n'est détectée. Cela permet de choisir la fréquence minimale de transmission pour réduire la quantité de données transmises sur le réseau si rien ne se passe.

Attention, si vous configurez de nombreux hubs pour effectuer le callback à la même heure, vous allez générer un pic de charge sur votre serveur web. Il est donc souhaitable d'utiliser le paramètre décalage pour équilibrer la charge.

## 7.8. Tests

Afin de vous permettre de débugger le processus, YoctoHub-Ethernet vous permet de visualiser la réponse au callback envoyé par le serveur web. Dans la fenêtre de configuration des callbacks, cliquez sur le bouton **test** une fois que vous avez renseigné tous les champs pour ouvrir la fenêtre de tests.

La fenêtre vous montre l'état actuel du système de callback, pour vous permettre de voir par exemple si un callback est actuellement en cours sur le serveur web. Dans tous les cas, tant que cette fenêtre est ouverte, aucun callback HTTP ne sera déclenché automatiquement. C'est en pressant le bouton **Test** que vous pourrez déclancher manuellement un callback. Une fois déclanché, la fenêtre vous montre les informations retournées par le service web, comme dans l'exemple ci-dessous:




*Le résultat du test de callback avec un Yocto-PowerRelay et un Yocto-Temperature*

Si le résultat vous paraît satisfaisant, fermez la fenêtre de debug, et cliquez sur **Ok**.

# 7.9. Connexions spontanées

En plus de connexions liées aux callbacks définis décrites dans les sections précédentes, YoctoHub-Ethernet va occasionnellement tenter d'établir des connexions vers l'extérieur. Ces connexions sont les suivantes

- Connexion DHCP: Si l'adresse IP du hub est configurée pour être attribuée par DHCP, le YoctoHub-Ethernet effectuera des connections périodiques sur le serveur DHCP indiqué dans la configuration du hub. Si cette connexion ne peut pas être effectuée, une adresse IP par défaut sera affectée au hub.
- Connexion DNS: à chaque fois que le YoctoHub-Ethernet doit se connecter à un serveur externe, il effectue une connexion à un serveur DNS pour obtenir l'adresse IP qui correspond au nom du serveur qu'il est sur le point contacter. Si ces connexions DNS ne peuvent pas être effectuées, le serveur NTP par défaut ne pourra pas être contacté et les callbacks HTTP ne fonctionneront que s'ils sont définis par une adresse IP explicite.
- Connexion NTP: afin de maintenir la synchronisation de son horloge interne, le hub va se connecter régulièrement à un serveur NTP. Il est possible de changer l'adresse de ce serveur et modifiant l'option "*ntpServer*" de la fonction "*network*" . Si ces connexions NTP ne peuvent pas être effectuées, l'horloge interne du Hub va se mettre à dériver.
- SSDP et Multicast DNS: le YoctoHub-Ethernet effectue périodiquement des annonces SSDP et Multicast DNS pour notifier sa présence sur le réseau. Ces connexions peuvent être désactivées en modifiant l'attribut "*discoverable*" de la fonction "*network*".
- Installation de Yocto-Visualization (for web): Lorsque l'utilisateur déclenche l'installation de Yocto-Visualization (for web) depuis l'interface du YoctoHub-Ethernet, le hub va automatiquement télécharger le fichier d'installation le plus récent depuis www.yoctopuce.com
- Test de version: à chaque fois que la fenêtre de propriété ou de configuration d'un module est ouverte, le YoctoHub-Ethernet effectue une requête sur www.yoctopuce.com pour vérifier si le firmware du module est à jour. Cette requête ne contient que le numéro de série du module et sert à indiquer à l'utilisateur si un nouveau firmware est disponible.
- Téléchargement de firmware: A chaque fois que la fenêtre de mise à jour de firmware est ouverte, le YoctoHub-Ethernet effectue une requête sur www.yoctopuce.com pour y récupérer le firmware le plus plus récent. Cette connexion permet d'éviter à l'utilisateur d'avoir à télécharger manuellement le dernier firmware.

Notez que ces deux dernières connexions sont en fait établies par le navigateur web qui affiche l'interface utilisateur du YoctoHub-Ethernet. De plus, ces connexions sont purement optionnelles, si elles ne peuvent pas être établies, l'interface continuera à fonctionner normalement.

Voici un tableau résumant les paramètres exacts de ces connexions:

| Justification | Protocole Port | Adresse cible | Comment les désactiver |
|---|---|---|---|
| Adresse IP dynamique | DHCP UDP 67 | broadcast | configurer une adresse IP statique |
| Résolution de nom | DNS UDP 53 | serveur DNS configuré | configurer l'adresse IP du serveur NTP et définir les callbacks par adresse IP |
| Mise à jour de l'heure | NTP UDP 123 | 1.yoctopuce.pool.ntp.org ou serveur configuré | régler le serveur NTP à 255.255.255.255 |
| Discovery | SSDP UDP 1900 | multicast 239.255.255.250 | désactiver `discoverable` |
| Discovery | MDNS UDP 5353 | multicast 224.0.0.251 | désactiver `discoverable` |
| Installation de YV4web | HTTP TCP 80 | www.yoctopuce.com | ne pas utiliser le lien rapide sur l'interface Web |
| Test de version | HTTPS TCP 443 | www.yoctopuce.com | ne pas utiliser l'interface Web du YoctoHub-Ethernet |
| Téléchargement firmware | HTTPS TCP 443 | www.yoctopuce.com | ne pas faire la mise à jour par l'interface Web |

# 8. Personnalisation de l'interface Web

Votre YoctoHub-Ethernet dispose d'un petit système de fichiers embarqué, qui permet de stocker des fichiers personnalisés utilisables par le YoctoHub-Ethernet. Le système de fichiers se manipule grâce à la libraire *yocto_files*. Vous pourrez y stocker les fichiers de votre choix. Au besoin, vous pourrez y stocker une application Web permettant de gérer les modules connectés à votre YoctoHub-Ethernet.

## 8.1. Utilisation

### Utilisation interactive

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

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

### Utilisation programmée

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

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

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

que vous désirez, et ainsi le vérifier sans avoir à transférer le fichier. La fonction CRC utilisée par le système de fichiers Yoctopuce est la même que celle d'Ethernet, Gzip, PNG, etc. Sa valeur caractéristique pour la chaîne de neuf caractères "123456789" est 0xCBF43926.

### Utilisation par HTTP

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

### Interfaces utilisateur et optimisation

Puisque que pouvez sauver des fichiers directement sur la mémoire flash du module et les accéder depuis l'extérieur, il est très facile de construire une application WEB pour contrôler les modules connectés au hub et de la stocker directement sur le hub. C'est un moyen très pratique pour construire des systèmes télécommandables depuis un smart-phone ou une tablette. Cependant le YoctoHub-Ethernet est plus limité qu'un serveur WEB normal: il n'accepte qu'un nombre limité de connexions en parallèle. La plupart des browsers WEB actuels ayant tendance à ouvrir un maximum de connexions en parallèle pour charger tous les éléments d'une page WEB, cela peut mener à des temps de chargement très long. Pour éviter cela, essayez de garder vos pages WEB aussi compactes que possible en incluant le code javascript et CSS directement dans la page. Si vous le pouvez, incluez aussi les images en base64.

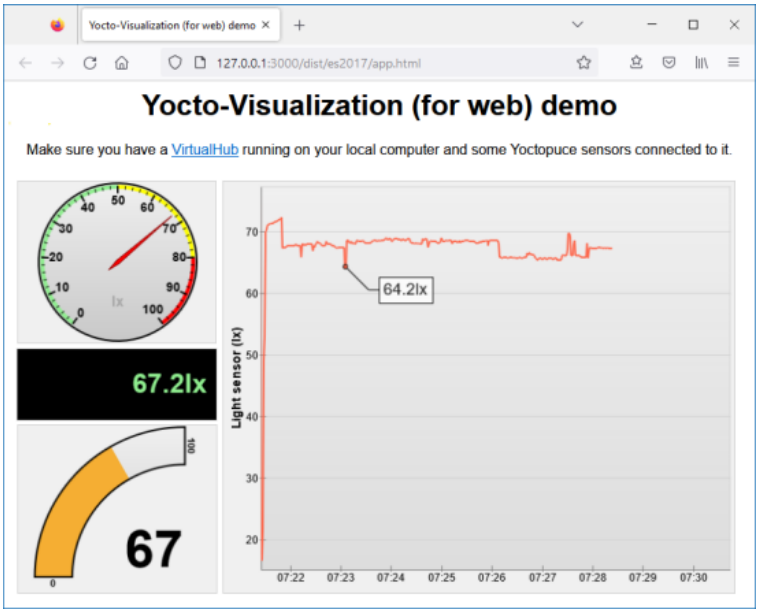
## 8.2. Limitations

Le filesystem embarqué sur votre YoctoHub-Ethernet a quelques limitations techniques:

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

# 9. Installation de Yocto-Visualization (for web)

Yocto-Visualization (for web) est une petite application Web qui permet de visualiser facilement les valeurs mesurées par des capteurs Yoctopuce.

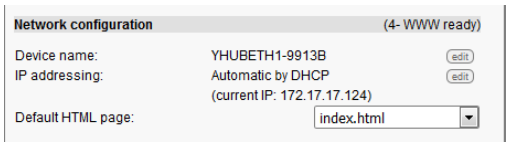



*L'interface de Yocto-Visualization (for web)*

Il est possible d'installer Yocto-Visualization (for web) comme plug-in de YoctoHub-Ethernet, directement via l'interface Web de configuration de YoctoHub-Ethernet.

Pour installer Yocto-Visualization (for web) depuis l'interface Web de YoctoHub-Ethernet

1. Dans l'interface de YoctoHub-Ethernet, ouvrez la fenêtre de configuration de YoctoHub-Ethernet en cliquant sur **configure**.
2. Dans la section **Network configuration**, en face de `Yocto-Visualization-4web`, cliquez sur **start installer**.
3. Il suffit ensuite de suivre les étapes en cliquant sur **Next** de manière répétée puis **OK**. Assurez-vous simplement que dans le champ **Hub address** le port ne soit pas présent deux fois.




*La section **Network configuration** de la fenêtre de configuration de YoctoHub-Ethernet*

Pour utiliser Yocto-Visualization (for web), référez-vous aux articles du blog du site de Yoctopuce.

# 10. Programmation

## 10.1. Accès aux modules connectés

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

```
YAPI.RegisterHub("usb",errmsg);
```

devient

```
YAPI.RegisterHub("192.168.0.10",errmsg); // l'adresse IP du hub est 192.168.0.10
```

## 10.2. Contrôle du YoctoHub-Ethernet

Du point de vue API de programmation, le YoctoHub-Ethernet est un module comme les autres. Il est parfaitement contrôlable depuis l'API Yoctopuce. Pour ce faire, vous aurez besoin des classes suivantes.

### Module

Cette classe, commune à tous les modules Yoctopuce permet de contrôler le module en temps que tel. Elle vous permettra de contrôler la Yocto-Led, de connaître la consommation sur USB du YoctoHub-Ethernet, etc.

### Network

Cette classe permet de contrôler la partie réseau du YoctoHub-Ethernet, vous pourrez contrôler l'état du link, lire l'adresse MAC, changer l'adresse IP du YoctoHub-Ethernet, connaître la consommation sur PoE, etc.

### Hub

Cette classe permet d'obtenir l'état de la connexion entre la librairie de programmation et les YoctoHubs ou VirtualHubs qui ont été enregistrés.

## HubPort

Cette classe permet de contrôler la partie hub. Vous pourrez activer ou désactiver les ports du YoctoHub-Ethernet, vous pourrez aussi savoir quel module est connecté à quel port.

## Files

Cette classe permet d'accéder aux fichiers stockées dans la mémoire flash du YoctoHub-Ethernet. Le YoctoHub-Ethernet dispose en effet d'un petit système de fichiers qui vous permet de stocker par exemple une Web App contrôlant les modules connectés au YoctoHub-Ethernet.

Vous trouverez quelques exemples de contrôle du YoctoHub-Ethernet par programmation dans les librairies Yoctopuce, disponibles gratuitement sur le site de Yoctopuce.

# 11. Référence de l'API de haut niveau

Ce chapitre résume les fonctions de l'API de haut niveau pour commander votre YoctoHub-Ethernet. 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 des modules Yoctopuce. Vous trouverez des explications plus détaillées dans la documentation des modules que vous souhaitez raccorder à votre YoctoHub-Ethernet.

# 11.1. La classe YModule

Interface de contrôle des paramètres généraux des modules Yoctopuce

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

| | |
|---|---|
| js | **<script type='text/javascript' src='yocto_api.js'></script>** |
| cpp | **#include "yocto_api.h"** |
| m | **#import "yocto_api.h"** |
| pas | **uses yocto_api;** |
| vb | **yocto_api.vb** |
| cs | **yocto_api.cs** |
| java | **import com.yoctopuce.YoctoAPI.YModule;** |
| uwp | **import com.yoctopuce.YoctoAPI.YModule;** |
| py | **from yocto_api import *** |
| php | **require_once('yocto_api.php');** |
| ts | **in HTML: import { YAPI, YErrorMsg, YModule, YSensor } from '../../dist/esm/yocto_api_browser.js';**<br>**in Node.js: import { YAPI, YErrorMsg, YModule, YSensor } from 'yoctolib-cjs/yocto_api_nodejs.js';** |
| es | **in HTML: <script src="../../lib/yocto_api.js"></script>**<br>**in node.js: require('yoctolib-es2017/yocto_api.js');** |
| dnp | **import YoctoProxyAPI.YModuleProxy** |
| cp | **#include "yocto_module_proxy.h"** |
| vi | **YModule.vi** |
| ml | **import YoctoProxyAPI.YModuleProxy"** |

## Fonction globales

**YModule.FindModule(func)**

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

cpp m pas vb cs java uwp py php ts es dnp

**YModule.FindModuleInContext(yctx, func)**

Permet de retrouver un module d'après un identifiant donné dans un Context YAPI.

java uwp ts es

**YModule.FirstModule()**

Commence l'énumération des modules accessibles par la librairie.

cpp m pas vb cs java uwp py php ts es

## Propriétés des objets `YModuleProxy`

**module→Beacon *[modifiable]***

état de la balise de localisation.

dnp

**module→FirmwareRelease *[lecture seule]***

Version du logiciel embarqué du module.

dnp

**module→FunctionId *[lecture seule]***

Identifiant matériel de la *n*ième fonction du module.

dnp

**module→HardwareId *[lecture seule]***

Identifiant unique du module.

dnp

**module→IsOnline *[lecture seule]***

Vérifie si le module est joignable.

dnp

**module→LogicalName *[modifiable]***

Nom logique du module.

dnp

**module→Luminosity *[modifiable]***

Luminosité des leds informatives du module (valeur entre 0 et 100).

dnp

**module→ProductId *[lecture seule]***

Identifiant USB du module, préprogrammé en usine.

dnp

**module→ProductName *[lecture seule]***

Nom commercial du module, préprogrammé en usine.

dnp

**module→ProductRelease *[lecture seule]***

Numéro uméro de révision du module hardware, préprogrammé en usine.

dnp

**module→SerialNumber *[lecture seule]***

Numéro de série du module, préprogrammé en usine.

dnp

## Méthodes des objets YModule

**module→addFileToHTTPCallback(filename)**

Ajoute un fichier aux données uploadées lors du prochain callback HTTP.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→checkFirmware(path, onlynew)**

Teste si le fichier byn est valide pour le module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→clearCache()**

Invalide le cache.

cpp m pas vb cs java py php ts es

**module→describe()**

Retourne un court texte décrivant le module.

cpp m pas vb cs java py php ts es

**module→download(pathname)**

Télécharge le fichier choisi du module et retourne son contenu.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→functionBaseType(functionIndex)**

Retourne le type de base de la *n*ième fonction du module.

cpp pas vb cs java py php ts es

**module→functionCount()**

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

cpp m pas vb cs java py php ts es

**module→functionId(functionIndex)**

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

cpp m pas vb cs java py php ts es

**module→functionName(functionIndex)**

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

cpp m pas vb cs java py php ts es

**module→functionType(functionIndex)**

Retourne le type de la *n*ième fonction du module.

cpp pas vb cs java py php ts es

**module→functionValue(functionIndex)**

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

cpp m pas vb cs java py php ts es

**module→get_allSettings()**

Retourne tous les paramètres de configuration du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_beacon()**

Retourne l'état de la balise de localisation.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_errorMessage()**

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

cpp m pas vb cs java py php ts es

**module→get_errorType()**

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

cpp m pas vb cs java py php ts es

**module→get_firmwareRelease()**

Retourne la version du logiciel embarqué du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_functionIds(funType)**

Retourne les identifiants matériels des fonctions correspondant au type passé en argument.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_hardwareId()**

Retourne l'identifiant unique du module.

cpp m vb cs java py php ts es dnp pas uwp cmd

**module→get_icon2d()**

Retourne l'icône du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_lastLogs()**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_logicalName()**

Retourne le nom logique du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_luminosity()**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_parentHub()**

Retourne le numéro de série du YoctoHub sur lequel est connecté le module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_persistentSettings()**

Retourne l'état courant des réglages persistents du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_productId()**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_productName()**

Retourne le nom commercial du module, préprogrammé en usine.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_productRelease()**

Retourne le numéro uméro de révision du module hardware, préprogrammé en usine.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_rebootCountdown()**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_serialNumber()**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_subDevices()**

Retourne la liste des modules branchés au module courant.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_upTime()**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_url()**

Retourne l'URL utilisée pour accéder au module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_usbCurrent()**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→get_userData()**

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

cpp m pas vb cs java py php ts es

**module→get_userVar()**

Retourne la valeur entière précédemment stockée dans cet attribut.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→hasFunction(funcId)**

Teste la présence d'une fonction pour le module courant.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→isOnline()**

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

cpp m pas vb cs java py php ts es dnp

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

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

**module→load(msValidity)**

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

cpp m pas vb cs java py php ts es

**module→load_async(msValidity, callback, context)**

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

**module→log(text)**

Ajoute un message arbitraire dans les logs du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→nextModule()**

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

cpp m pas vb cs java uwp py php ts es

**module→reboot(secBeforeReboot)**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→registerBeaconCallback(callback)**

Enregistre une fonction de callback qui sera appelée à chaque changement d'état de la balise de localisation du module.

cpp m pas vb cs java uwp py php ts es

**module→registerConfigChangeCallback(callback)**

Enregistre une fonction de callback qui sera appelée à chaque fois qu'un réglage persistant d'un module est modifié (par exemple changement d'unité de mesure, etc.)

cpp m pas vb cs java uwp py php ts es

**module→registerLogCallback(callback)**

Enregistre une fonction de callback qui sera appelée à chaque fois le module émet un message de log.

cpp m pas vb cs java uwp py php ts es

**module→revertFromFlash()**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→saveToFlash()**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→set_allSettings(settings)**

Rétablit tous les paramètres du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→set_allSettingsAndFiles(settings)**

Rétablit tous les paramètres de configuration et fichiers sur un module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→set_beacon(newval)**

Allume ou éteint la balise de localisation du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→set_logicalName(newval)**

Change le nom logique du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→set_luminosity(newval)**

Modifie la luminosité des leds informatives du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

cpp m pas vb cs java py php ts es

**module→set_userVar(newval)**

Stocke une valeur 32 bits dans la mémoire volatile du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→triggerConfigChangeCallback()**

Force le déclenchement d'un callback de changement de configuration, afin de vérifier s'ils sont disponibles ou pas.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→triggerFirmwareUpdate(secBeforeReboot)**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→updateFirmware(path)**

Prepare une mise à jour de firmware du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**module→updateFirmwareEx(path, force)**

Prepare une mise à jour de firmware du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

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

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

ts es

# 11.2. La classe YNetwork

Interface pour intéragir avec les interfaces réseau, disponibles par exemple dans le YoctoHub-Ethernet, le YoctoHub-GSM-4G, le YoctoHub-Wireless-SR et le YoctoHub-Wireless-n

La classe `YNetwork` permet de contrôler les paramètres TCP/IP des modules Yoctopuce dotés d'une interface réseau.

Pour utiliser les fonctions décrites ici, vous devez inclure:

| | |
|---|---|
| es | **in HTML: <script src="../../lib/yocto_network.js"></script>**<br>**in node.js: require('yoctolib-es2017/yocto_network.js');** |
| js | **<script type='text/javascript' src='yocto_network.js'></script>** |
| cpp | **#include "yocto_network.h"** |
| m | **#import "yocto_network.h"** |
| pas | **uses yocto_network;** |
| vb | **yocto_network.vb** |
| cs | **yocto_network.cs** |
| java | **import com.yoctopuce.YoctoAPI.YNetwork;** |
| uwp | **import com.yoctopuce.YoctoAPI.YNetwork;** |
| py | **from yocto_network import *** |
| php | **require_once('yocto_network.php');** |
| ts | **in HTML: import { YNetwork } from '../../dist/esm/yocto_network.js';**<br>**in Node.js: import { YNetwork } from 'yoctolib-cjs/yocto_network.js';** |
| dnp | **import YoctoProxyAPI.YNetworkProxy** |
| cp | **#include "yocto_network_proxy.h"** |
| vi | **YNetwork.vi** |
| ml | **import YoctoProxyAPI.YNetworkProxy** |

## Fonction globales

**YNetwork.FindNetwork(func)**

Permet de retrouver une interface réseau d'après un identifiant donné.

cpp m pas vb cs java uwp py php ts es dnp

**YNetwork.FindNetworkInContext(yctx, func)**

Permet de retrouver une interface réseau d'après un identifiant donné dans un Context YAPI.

java uwp ts es

**YNetwork.FirstNetwork()**

Commence l'énumération des interfaces réseau accessibles par la librairie.

cpp m pas vb cs java uwp py php ts es

**YNetwork.FirstNetworkInContext(yctx)**

Commence l'énumération des interfaces réseau accessibles par la librairie.

java uwp ts es

**YNetwork.GetSimilarFunctions()**

Enumère toutes les fonctions de type Network disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

dnp

## Propriétés des objets `YNetworkProxy`

**network→AdminPassword *[modifiable]***

Chaîne de hash si un mot de passe a été configuré pour l'utilisateur "admin", ou sinon une chaîne vide.

dnp

**network→AdvertisedValue *[lecture seule]***

Courte chaîne de caractères représentant l'état courant de la fonction.

dnp

**network→CallbackCredentials *[modifiable]***

Version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide.

dnp

**network→CallbackEncoding *[modifiable]***

Encodage à utiliser pour représenter les valeurs notifiées par callback.

dnp

**network→CallbackInitialDelay *[modifiable]***

Attente initiale avant la première notification par callback, en secondes.

dnp

**network→CallbackMaxDelay *[modifiable]***

Attente entre deux callback HTTP lorsque rien n'est à signaler, en secondes.

dnp

**network→CallbackMethod *[modifiable]***

Méthode HTTP à utiliser pour signaler les changements d'état par callback.

dnp

**network→CallbackMinDelay *[modifiable]***

Attente minimale entre deux callbacks HTTP, en secondes.

dnp

**network→CallbackSchedule *[modifiable]***

Planification des callbacks HTTP, sous forme de chaîne de caractères.

dnp

**network→CallbackTemplate *[modifiable]***

état d'activation du fichier modèle pour personnaliser le format des callbacks.

dnp

**network→CallbackUrl *[modifiable]***

Adresse (URL) de callback à notifier lors de changement d'état significatifs.

dnp

**network→DefaultPage *[modifiable]***

Page HTML à envoyer pour l'URL "/" **Modifiable**.

dnp

**network→Discoverable *[modifiable]***

état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

dnp

**network→FriendlyName *[lecture seule]***

Identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

dnp

**network→FunctionId *[lecture seule]***

Identifiant matériel de l'interface réseau, sans référence au module.

dnp

**network→HardwareId *[lecture seule]***

Identifiant matériel unique de la fonction au format `SERIAL.FUNCTIONID`.

dnp

**network→HttpPort *[modifiable]***

Port TCP utilisé pour l'interface Web du hub.

dnp

**network→IpAddress *[lecture seule]***

Adresse IP utilisée par le module Yoctopuce.

dnp

**network→IsOnline *[lecture seule]***

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

dnp

**network→LogicalName *[modifiable]***

Nom logique de la fonction.

dnp

**network→MacAddress *[lecture seule]***

Adresse MAC de l'interface réseau, unique pour chaque module.

dnp

**network→NtpServer *[modifiable]***

Adresse IP du serveur de NTP à utiliser pour maintenir le module à l'heure.

dnp

**network→PrimaryDNS *[modifiable]***

Adresse IP du serveur de noms primaire que le module doit utiliser.

dnp

**network→Readiness *[lecture seule]***

état de fonctionnement atteint par l'interface réseau.

dnp

**network→SecondaryDNS *[modifiable]***

Adresse IP du serveur de noms secondaire que le module doit utiliser.

dnp

**network→SerialNumber *[lecture seule]***

Numéro de série du module, préprogrammé en usine.

dnp

**network→UserPassword *[modifiable]***

Chaîne de hash si un mot de passe a été configuré pour l'utilisateur "user", ou sinon une chaîne vide.

dnp

**network→WwwWatchdogDelay *[modifiable]***

Durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

dnp

## Méthodes des objets YNetwork

### network→callbackLogin(username, password)

Contacte le callback de notification et sauvegarde un laisser-passer pour s'y connecter.

cpp m pas vb cs java py php ts es dnp cmd

### network→clearCache()

Invalide le cache.

cpp m pas vb cs java py php ts es

### network→describe()

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

cpp m pas vb cs java py php ts es

### network→get_adminPassword()

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

cpp m pas vb cs java uwp py php ts es dnp cmd

### network→get_advertisedValue()

Retourne la valeur courante de l'interface réseau (pas plus de 6 caractères).

cpp m pas vb cs java uwp py php ts es dnp cmd

### network→get_callbackCredentials()

Retourne une version hashée du laisser-passer pour le callback de notification s'il a été configuré, ou sinon une chaîne vide.

cpp m pas vb cs java uwp py php ts es dnp cmd

### network→get_callbackEncoding()

Retourne l'encodage à utiliser pour représenter les valeurs notifiées par callback.

cpp m pas vb cs java uwp py php ts es dnp cmd

### network→get_callbackInitialDelay()

Retourne l'attente initiale avant la première notification par callback, en secondes.

cpp m pas vb cs java uwp py php ts es dnp cmd

### network→get_callbackMaxDelay()

Retourne l'attente entre deux callback HTTP lorsque rien n'est à signaler, en secondes.

cpp m pas vb cs java uwp py php ts es dnp cmd

### network→get_callbackMethod()

Retourne la méthode HTTP à utiliser pour signaler les changements d'état par callback.

cpp m pas vb cs java uwp py php ts es dnp cmd

### network→get_callbackMinDelay()

Retourne l'attente minimale entre deux callbacks HTTP, en secondes.

cpp m pas vb cs java uwp py php ts es dnp cmd

### network→get_callbackSchedule()

Retourne la planification des callbacks HTTP, sous forme de chaîne de caractères.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_callbackTemplate()**

Retourne l'état d'activation du fichier modèle pour personnaliser le format des callbacks.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_callbackUrl()**

Retourne l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_currentDNS()**

Retourne l'adresse IP du serveur DNS actuellement utilisé par le module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_defaultPage()**

Retourne la page HTML à envoyer pour l'URL "/"

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_discoverable()**

Retourne l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_errorMessage()**

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

cpp m pas vb cs java py php ts es

**network→get_errorType()**

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

cpp m pas vb cs java py php ts es

**network→get_friendlyName()**

Retourne un identifiant global de l'interface réseau au format NOM_MODULE.NOM_FONCTION.

cpp m cs java py php ts es dnp

**network→get_functionDescriptor()**

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

cpp m pas vb cs java py php ts es

**network→get_functionId()**

Retourne l'identifiant matériel de l'interface réseau, sans référence au module.

cpp m vb cs java py php ts es dnp

**network→get_hardwareId()**

Retourne l'identifiant matériel unique de l'interface réseau au format SERIAL.FUNCTIONID.

cpp m vb cs java py php ts es dnp

**network→get_httpPort()**

Retourne le port TCP utilisé pour l'interface Web du hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_ipAddress()**

Retourne l'adresse IP utilisée par le module Yoctopuce.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_ipConfig()**

Retourne la configuration IP de l'interface réseau.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_logicalName()**

Retourne le nom logique de l'interface réseau.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_macAddress()**

Retourne l'adresse MAC de l'interface réseau, unique pour chaque module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_module()**

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

cpp m pas vb cs java py php ts es dnp

**network→get_module_async(callback, context)**

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

**network→get_ntpServer()**

Retourne l'adresse IP du serveur de NTP à utiliser pour maintenir le module à l'heure.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_poeCurrent()**

Retourne le courant consommé par le module depuis Power-over-Ethernet (PoE), en milliampères.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_primaryDNS()**

Retourne l'adresse IP du serveur de noms primaire que le module doit utiliser.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_readiness()**

Retourne l'état de fonctionnement atteint par l'interface réseau.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_router()**

Retourne l'adresse IP du routeur (passerelle) utilisé par le module (*default gateway*).

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_secondaryDNS()**

Retourne l'adresse IP du serveur de noms secondaire que le module doit utiliser.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_serialNumber()**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_subnetMask()**

Retourne le masque de sous-réseau utilisé par le module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_userData()**

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

cpp m pas vb cs java py php ts es

**network→get_userPassword()**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→get_wwwWatchdogDelay()**

Retourne la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→isOnline()**

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

cpp m pas vb cs java py php ts es dnp

**network→isOnline_async(callback, context)**

Vérifie si le module hébergeant l'interface réseau est joignable, sans déclencher d'erreur.

**network→isReadOnly()**

Indique si la fonction est en lecture seule.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→load(msValidity)**

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

cpp m pas vb cs java py php ts es

**network→loadAttribute(attrName)**

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

cpp m pas vb cs java uwp py php ts es dnp

**network→load_async(msValidity, callback, context)**

Met en cache les valeurs courantes de l'interface réseau, avec une durée de validité spécifiée.

**network→muteValueCallbacks()**

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→nextNetwork()**

Continue l'énumération des interfaces réseau commencée à l'aide de `yFirstNetwork()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les interfaces réseau sont retournés.

cpp m pas vb cs java uwp py php ts es

**network→ping(host)**

Ping l'adresse choisie pour vérifier la connexion réseau.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→registerValueCallback(callback)**

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

cpp m pas vb cs java uwp py php ts es

**network→set_adminPassword(newval)**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_callbackCredentials(newval)**

Modifie le laisser-passer pour se connecter à l'adresse de callback.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_callbackEncoding(newval)**

Modifie l'encodage à utiliser pour représenter les valeurs notifiées par callback.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_callbackInitialDelay(newval)**

Modifie l'attente initiale avant la première notification par callback, en secondes.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_callbackMaxDelay(newval)**

Modifie l'attente entre deux callback HTTP lorsque rien n'est à signaler.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_callbackMethod(newval)**

Modifie la méthode HTTP à utiliser pour signaler les changements d'état par callback.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_callbackMinDelay(newval)**

Modifie l'attente minimale entre deux callbacks HTTP, en secondes.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_callbackSchedule(newval)**

Modifie la planification des callbacks HTTP, sous forme de chaîne de caractères.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_callbackTemplate(newval)**

Modifie l'état d'activation du fichier modèle pour personnaliser les callbacks.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_callbackUrl(newval)**

Modifie l'adresse (URL) de callback à notifier lors de changement d'état significatifs.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_defaultPage(newval)**

Modifie la page HTML par défaut du hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_discoverable(newval)**

Modifie l'état d'activation du protocole d'annonce sur le réseau permettant de retrouver facilement le module (protocols uPnP/Bonjour).

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_httpPort(newval)**

Modifie le port TCP utilisé pour l'interface Web du hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_logicalName(newval)**

Modifie le nom logique de l'interface réseau.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_ntpServer(newval)**

Modifie l'adresse IP du serveur NTP que le module doit utiliser.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_periodicCallbackSchedule(interval, offset)**

Configure la planification de callbacks HTTP périodiques (fonction simplifiée).

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_primaryDNS(newval)**

Modifie l'adresse IP du serveur de noms primaire que le module doit utiliser.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_secondaryDNS(newval)**

Modifie l'adresse IP du serveur de nom secondaire que le module doit utiliser.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

cpp m pas vb cs java py php ts es

**network→set_userPassword(newval)**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→set_wwwWatchdogDelay(newval)**

Modifie la durée de perte de connection WWW tolérée (en secondes) avant de déclencher un redémarrage automatique pour tenter de récupérer la connectivité Internet.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→triggerCallback()**

Déclenche un callback HTTP rapidement.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→unmuteValueCallbacks()**

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→useDHCP(fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)**

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→useDHCPauto()**

Modifie la configuration de l'interface réseau pour utiliser une adresse assignée automatiquement par le serveur DHCP.

cpp m pas vb cs java uwp py php ts es dnp cmd

**network→useStaticIP(ipAddress, subnetMaskLen, router)**

Modifie la configuration de l'interface réseau pour utiliser une adresse IP assignée manuellement (adresse IP statique).

cpp m pas vb cs java uwp py php ts es dnp cmd

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

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

ts es

# 11.3. La classe YHub

Pour utiliser les fonctions décrites ici, vous devez inclure:

| | |
|---|---|
| js | **<script type='text/javascript' src='yocto_module.js'></script>** |
| cpp | **#include "yocto_module.h"** |
| m | **#import "yocto_module.h"** |
| pas | **uses yocto_module;** |
| vb | **yocto_module.vb** |
| cs | **yocto_module.cs** |
| java | **import com.yoctopuce.YoctoAPI.YHub;** |
| uwp | **import com.yoctopuce.YoctoAPI.YHub;** |
| py | **from yocto_module import *** |
| php | **require_once('yocto_module.php');** |
| ts | **in HTML: import { YHub } from '../../dist/esm/yocto_module.js';**<br>**in Node.js: import { YHub } from 'yoctolib-cjs/yocto_module.js';** |
| es | **in HTML: <script src="../../lib/yocto_module.js"></script>**<br>**in node.js: require('yoctolib-es2017/yocto_module.js');** |
| dnp | **import YoctoProxyAPI.YHubProxy** |
| cp | **#include "yocto_module_proxy.h"** |
| ml | **import YoctoProxyAPI.YHubProxy** |

## Fonction globales

### YHub.FirstHubInUse()

Commence l'énumération des hubs en utilisation par la librairie.

cpp m vb cs java uwp py php ts es dnp

### YHub.FirstHubInUseInContext(yctx)

Commence l'énumération des hubs en utilisation par la librairie dans un contexte YAPI donné.

java uwp ts es

## Méthodes des objets YHub

### hub→get_connectionUrl()

Retourne l'URL actuellement utilisée pour communiquer avec ce hub.

cpp m pas vb cs java uwp py php ts es dnp

### hub→get_knownUrls()

Retourne toutes les URLs sous lesquelles ce hub a été enregistré.

cpp m pas vb cs java uwp py php ts es dnp

### hub→get_networkTimeout()

Retourne le délai de connexion réseau pour ce hub.

cpp m pas vb cs java uwp py php ts es dnp

### hub→get_registeredUrl()

Retourne l'URL sous laquelle ce hub a été initialement enregistré.

cpp m pas vb cs java uwp py php ts es dnp

### hub→get_serialNumber()

Retourne le numéro de série du hub, si la connexion a déjà été établie.

cpp m pas vb cs java uwp py php ts es dnp

**hub→get_userData()**

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

cpp m pas vb cs java uwp py php ts es

**hub→isInUse()**

Indique si ce hub est actuellement enregistré dans l'API.

cpp m pas vb cs java uwp py php ts es dnp

**hub→isOnline()**

Indique si la communication avec ce hub est actuellement active.

cpp m pas vb cs java uwp py php ts es dnp

**hub→isReadOnly()**

Indique si l'accès à ce hub est protégé contre l'écriture.

cpp m pas vb cs java uwp py php ts es dnp

**hub→nextHubInUse()**

Continue l'énumération des hubs commencée à l'aide de `YHub.FirstHubInUse()`.

cpp m pas vb cs java uwp py php ts es dnp

**hub→set_networkTimeout(networkMsTimeout)**

Change le délai de connexion réseau pour ce hub.

cpp m pas vb cs java uwp py php ts es dnp

**hub→set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

cpp m pas vb cs java uwp py php ts es

# 11.4. La classe YHubPort

Interface pour intéragir avec les ports de YoctoHub, disponibles par exemple dans le YoctoHub-Ethernet, le YoctoHub-GSM-4G, le YoctoHub-Shield et le YoctoHub-Wireless-n

La classe `YHubPort` permet de contrôler l'alimentation des ports descendants d'un YoctoHub. Il permet de détecter si un module y est raccordé et lequel. Un YHubPort reçoit toujours automatiquement comme nom logique le numéro de série unique du module Yoctopuce qui y est connecté.

Pour utiliser les fonctions décrites ici, vous devez inclure:

| | |
|---|---|
| es | **in HTML: <script src="../../lib/yocto_hubport.js"></script>**<br>**in node.js: require('yoctolib-es2017/yocto_hubport.js');** |
| js | **<script type='text/javascript' src='yocto_hubport.js'></script>** |
| cpp | **#include "yocto_hubport.h"** |
| m | **#import "yocto_hubport.h"** |
| pas | **uses yocto_hubport;** |
| vb | **yocto_hubport.vb** |
| cs | **yocto_hubport.cs** |
| java | **import com.yoctopuce.YoctoAPI.YHubPort;** |
| uwp | **import com.yoctopuce.YoctoAPI.YHubPort;** |
| py | **from yocto_hubport import *** |
| php | **require_once('yocto_hubport.php');** |
| ts | **in HTML: import { YHubPort } from '../../dist/esm/yocto_hubport.js';**<br>**in Node.js: import { YHubPort } from 'yoctolib-cjs/yocto_hubport.js';** |
| dnp | **import YoctoProxyAPI.YHubPortProxy** |
| cp | **#include "yocto_hubport_proxy.h"** |
| vi | **YHubPort.vi** |
| ml | **import YoctoProxyAPI.YHubPortProxy** |

## Fonction globales

### YHubPort.FindHubPort(func)

Permet de retrouver un port de YoctoHub d'après un identifiant donné.

cpp m pas vb cs java uwp py php ts es dnp

### YHubPort.FindHubPortInContext(yctx, func)

Permet de retrouver un port de YoctoHub d'après un identifiant donné dans un Context YAPI.

java uwp ts es

### YHubPort.FirstHubPort()

Commence l'énumération des ports de YoctoHub accessibles par la librairie.

cpp m pas vb cs java uwp py php ts es

### YHubPort.FirstHubPortInContext(yctx)

Commence l'énumération des ports de YoctoHub accessibles par la librairie.

java uwp ts es

### YHubPort.GetSimilarFunctions()

Enumère toutes les fonctions de type HubPort disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

dnp

## Propriétés des objets YHubPortProxy

**hubport→AdvertisedValue *[lecture seule]***

Courte chaîne de caractères représentant l'état courant de la fonction.

dnp

**hubport→Enabled *[modifiable]***

Vrai si le port du YoctoHub est alimenté, faux sinon.

dnp

**hubport→FriendlyName *[lecture seule]***

Identifiant global de la fonction au format NOM_MODULE.NOM_FONCTION.

dnp

**hubport→FunctionId *[lecture seule]***

Identifiant matériel du port de YoctoHub, sans référence au module.

dnp

**hubport→HardwareId *[lecture seule]***

Identifiant matériel unique de la fonction au format SERIAL.FUNCTIONID.

dnp

**hubport→IsOnline *[lecture seule]***

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

dnp

**hubport→LogicalName *[modifiable]***

Nom logique de la fonction.

dnp

**hubport→PortState *[lecture seule]***

état actuel du port de YoctoHub.

dnp

**hubport→SerialNumber *[lecture seule]***

Numéro de série du module, préprogrammé en usine.

dnp

## Méthodes des objets YHubPort

**hubport→clearCache()**

Invalide le cache.

cpp m pas vb cs java py php ts es

**hubport→describe()**

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

cpp m pas vb cs java py php ts es

**hubport→get_advertisedValue()**

Retourne la valeur courante du port de YoctoHub (pas plus de 6 caractères).

cpp m pas vb cs java uwp py php ts es dnp cmd

**hubport→get_baudRate()**

Retourne la vitesse de transfert utilisée par le port de YoctoHub, en kbps.

cpp m pas vb cs java uwp py php ts es dnp cmd

**hubport→get_enabled()**

Retourne vrai si le port du YoctoHub est alimenté, faux sinon.

cpp m pas vb cs java uwp py php ts es dnp cmd

**hubport→get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du port de YoctoHub.

cpp m pas vb cs java py php ts es

**hubport→get_errorType()**

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

cpp m pas vb cs java py php ts es

**hubport→get_friendlyName()**

Retourne un identifiant global du port de YoctoHub au format `NOM_MODULE.NOM_FONCTION`.

cpp m cs java py php ts es dnp

**hubport→get_functionDescriptor()**

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

cpp m pas vb cs java py php ts es

**hubport→get_functionId()**

Retourne l'identifiant matériel du port de YoctoHub, sans référence au module.

cpp m vb cs java py php ts es dnp

**hubport→get_hardwareId()**

Retourne l'identifiant matériel unique du port de YoctoHub au format `SERIAL.FUNCTIONID`.

cpp m vb cs java py php ts es dnp

**hubport→get_logicalName()**

Retourne le nom logique du port de YoctoHub.

cpp m pas vb cs java uwp py php ts es dnp cmd

**hubport→get_module()**

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

cpp m pas vb cs java py php ts es dnp

**hubport→get_module_async(callback, context)**

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

**hubport→get_portState()**

Retourne l'état actuel du port de YoctoHub.

cpp m pas vb cs java uwp py php ts es dnp cmd

**hubport→get_serialNumber()**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**hubport→get_userData()**

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

cpp m pas vb cs java py php ts es

**hubport→isOnline()**

Vérifie si le module hébergeant le port de YoctoHub est joignable, sans déclencher d'erreur.

cpp m pas vb cs java py php ts es dnp

**hubport→isOnline_async(callback, context)**

Vérifie si le module hébergeant le port de YoctoHub est joignable, sans déclencher d'erreur.

**hubport→isReadOnly()**

Indique si la fonction est en lecture seule.

cpp m pas vb cs java uwp py php ts es dnp cmd

**hubport→load(msValidity)**

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

cpp m pas vb cs java py php ts es

**hubport→loadAttribute(attrName)**

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

cpp m pas vb cs java uwp py php ts es dnp

**hubport→load_async(msValidity, callback, context)**

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

**hubport→muteValueCallbacks()**

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

cpp m pas vb cs java uwp py php ts es dnp cmd

**hubport→nextHubPort()**

Continue l'énumération des ports de YoctoHub commencée à l'aide de `yFirstHubPort()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les ports de YoctoHub sont retournés.

cpp m pas vb cs java uwp py php ts es

**hubport→registerValueCallback(callback)**

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

cpp m pas vb cs java uwp py php ts es

**hubport→set_enabled(newval)**

Modifie le mode d'activation du port du YoctoHub.

cpp m pas vb cs java uwp py php ts es dnp cmd

**hubport→set_logicalName(newval)**

Modifie le nom logique du port de YoctoHub.

cpp m pas vb cs java uwp py php ts es dnp cmd

**hubport→set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

cpp m pas vb cs java py php ts es

**hubport→unmuteValueCallbacks()**

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

cpp m pas vb cs java uwp py php ts es dnp cmd

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

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

ts es

# 11.5. La classe YFiles

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

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

Pour utiliser les fonctions décrites ici, vous devez inclure:

| | |
|---|---|
| js | **<script type='text/javascript' src='yocto_files.js'></script>** |
| cpp | **#include "yocto_files.h"** |
| m | **#import "yocto_files.h"** |
| pas | **uses yocto_files;** |
| vb | **yocto_files.vb** |
| cs | **yocto_files.cs** |
| java | **import com.yoctopuce.YoctoAPI.YFiles;** |
| uwp | **import com.yoctopuce.YoctoAPI.YFiles;** |
| py | **from yocto_files import *** |
| php | **require_once('yocto_files.php');** |
| ts | **in HTML: import { YFiles } from '../../dist/esm/yocto_files.js';**<br>**in Node.js: import { YFiles } from 'yoctolib-cjs/yocto_files.js';** |
| es | **in HTML: <script src="../../lib/yocto_files.js"></script>**<br>**in node.js: require('yoctolib-es2017/yocto_files.js');** |
| dnp | **import YoctoProxyAPI.YFilesProxy** |
| cp | **#include "yocto_files_proxy.h"** |
| vi | **YFiles.vi** |
| ml | **import YoctoProxyAPI.YFilesProxy** |

## Fonction globales

### YFiles.FindFiles(func)

Permet de retrouver un système de fichier d'après un identifiant donné.

cpp m pas vb cs java uwp py php ts es dnp

### YFiles.FindFilesInContext(yctx, func)

Permet de retrouver un système de fichier d'après un identifiant donné dans un Context YAPI.

java uwp ts es

### YFiles.FirstFiles()

Commence l'énumération des systèmes de fichier accessibles par la librairie.

cpp m pas vb cs java uwp py php ts es

### YFiles.FirstFilesInContext(yctx)

Commence l'énumération des systèmes de fichier accessibles par la librairie.

java uwp ts es

### YFiles.GetSimilarFunctions()

Enumère toutes les fonctions de type Files disponibles sur les modules actuellement joignables par la librairie, et retourne leurs identifiants matériels uniques (hardwareId).

dnp

## Propriétés des objets YFilesProxy

**files→AdvertisedValue *[lecture seule]***

Courte chaîne de caractères représentant l'état courant de la fonction.

dnp

**files→FilesCount *[lecture seule]***

Nombre de fichiers présents dans le système de fichier.

dnp

**files→FriendlyName *[lecture seule]***

Identifiant global de la fonction au format `NOM_MODULE.NOM_FONCTION`.

dnp

**files→FunctionId *[lecture seule]***

Identifiant matériel du système de fichier, sans référence au module.

dnp

**files→HardwareId *[lecture seule]***

Identifiant matériel unique de la fonction au format `SERIAL.FUNCTIONID`.

dnp

**files→IsOnline *[lecture seule]***

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

dnp

**files→LogicalName *[modifiable]***

Nom logique de la fonction.

dnp

**files→SerialNumber *[lecture seule]***

Numéro de série du module, préprogrammé en usine.

dnp

## Méthodes des objets `YFiles`

**files→clearCache()**

Invalide le cache.

cpp m pas vb cs java py php ts es

**files→describe()**

Retourne un court texte décrivant de manière non-ambigüe l'instance du système de fichier au format `TYPE(NAME)=SERIAL.FUNCTIONID`.

cpp m pas vb cs java py php ts es

**files→download(pathname)**

Télécharge le fichier choisi du filesystème et retourne son contenu.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→download_async(pathname, callback, context)**

Procède au chargement du bloc suivant de mesures depuis l'enregistreur de données du module, de manière asynchrone.

**files→fileExist(filename)**

Test si un fichier esit dans le système de fichier du module.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→format_fs()**

Rétabli le système de fichier dans on état original, défragmenté.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→get_advertisedValue()**

Retourne la valeur courante du système de fichier (pas plus de 6 caractères).

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→get_errorMessage()**

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation du système de fichier.

cpp m pas vb cs java py php ts es

**files→get_errorType()**

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

cpp m pas vb cs java py php ts es

**files→get_filesCount()**

Retourne le nombre de fichiers présents dans le système de fichier.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→get_freeSpace()**

Retourne l'espace disponible dans le système de fichier pour charger des nouveaux fichiers, en octets.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→get_friendlyName()**

Retourne un identifiant global du système de fichier au format `NOM_MODULE.NOM_FONCTION`.

cpp m cs java py php ts es dnp

**files→get_functionDescriptor()**

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

cpp m pas vb cs java py php ts es

**files→get_functionId()**

Retourne l'identifiant matériel du système de fichier, sans référence au module.

cpp m vb cs java py php ts es dnp

**files→get_hardwareId()**

Retourne l'identifiant matériel unique du système de fichier au format `SERIAL.FUNCTIONID`.

cpp m vb cs java py php ts es dnp

**files→get_list(pattern)**

Retourne une liste d'objets objet YFileRecord qui décrivent les fichiers présents dans le système de fichier.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→get_logicalName()**

Retourne le nom logique du système de fichier.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→get_module()**

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

cpp m pas vb cs java py php ts es dnp

**files→get_module_async(callback, context)**

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

**files→get_serialNumber()**

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

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→get_userData()**

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

cpp m pas vb cs java py php ts es

**files→isOnline()**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

cpp m pas vb cs java py php ts es dnp

**files→isOnline_async(callback, context)**

Vérifie si le module hébergeant le système de fichier est joignable, sans déclencher d'erreur.

**files→isReadOnly()**

Indique si la fonction est en lecture seule.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→load(msValidity)**

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

cpp m pas vb cs java py php ts es

**files→loadAttribute(attrName)**

Retourne la valeur actuelle d'un attribut spécifique de la fonction, sous forme de texte, le plus rapidement possible mais sans passer par le cache.

cpp m pas vb cs java uwp py php ts es dnp

**files→load_async(msValidity, callback, context)**

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

**files→muteValueCallbacks()**

Désactive l'envoi de chaque changement de la valeur publiée au hub parent.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→nextFiles()**

Continue l'énumération des systèmes de fichier commencée à l'aide de `yFirstFiles()` Attention, vous ne pouvez faire aucune supposition sur l'ordre dans lequel les systèmes de fichier sont retournés.

cpp m pas vb cs java uwp py php ts es

**files→registerValueCallback(callback)**

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

cpp m pas vb cs java uwp py php ts es

**files→remove(pathname)**

Efface un fichier, spécifié par son path complet, du système de fichier.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→set_logicalName(newval)**

Modifie le nom logique du système de fichier.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→set_userData(data)**

Enregistre un contexte libre dans l'attribut userData de la fonction, afin de le retrouver plus tard à l'aide de la méthode `get_userData`.

cpp m pas vb cs java py php ts es

**files→unmuteValueCallbacks()**

Réactive l'envoi de chaque changement de la valeur publiée au hub parent.

cpp m pas vb cs java uwp py php ts es dnp cmd

**files→upload(pathname, content)**

Télécharge un contenu vers le système de fichier, au chemin d'accès spécifié.

cpp m pas vb cs java uwp py php ts es dnp cmd

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

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

ts es

# 12. Problèmes courants

## 12.1. Par où commencer ?

Si c'est la première fois que vous utilisez un module Yoctopuce et ne savez pas trop par où commencer, allez donc jeter un coup d'œil sur le blog de Yoctopuce. Il y a une section dédiée aux débutants [1].

## 12.2. Linux et USB

Pour fonctionner correctement sous Linux, la librairie a besoin d'avoir accès en écriture à tous les périphériques USB Yoctopuce. Or, par défaut, sous Linux les droits d'accès des utilisateurs non-root à USB sont limités à la lecture. Afin d'éviter de devoir lancer les exécutables en tant que root, il faut créer une nouvelle règle *udev* pour autoriser un ou plusieurs utilisateurs à accéder en écriture aux périphériques Yoctopuce.

Pour ajouter une règle *udev* à votre installation, il faut ajouter un fichier avec un nom au format "`##-nomArbitraire.rules`" dans le répertoire "`/etc/udev/rules.d`". Lors du démarrage du système, *udev* va lire tous les fichiers avec l'extension "`.rules`" de ce répertoire en respectant l'ordre alphabétique (par exemple, le fichier "`51-custom.rules`" sera interprété APRES le fichier "`50-udev-default.rules`").

Le fichier "`50-udev-default`" contient les règles *udev* par défaut du système. Pour modifier le comportement par défaut du système, il faut donc créer un fichier qui commence par un nombre plus grand que 50, qui définira un comportement plus spécifique que le défaut du système. Notez que pour ajouter une règle vous aurez besoin d'avoir un accès root sur le système.

Dans le répertoire `udev_conf` de l'archive de VirtualHub[2] pour Linux, vous trouverez deux exemples de règles qui vous éviteront de devoir partir de rien.

### Exemple 1: 51-yoctopuce.rules

Cette règle va autoriser tous les utilisateurs à accéder en lecture et en écriture aux périphériques Yoctopuce USB. Les droits d'accès pour tous les autres périphériques ne seront pas modifiés. Si ce scénario vous convient, il suffit de copier le fichier "`51-yoctopuce_all.rules`" dans le répertoire "`/etc/udev/rules.d`" et de redémarrer votre système.

[1] voir: http://www.yoctopuce.com/FR/blog_by_categories/pour-les-debutants
[2] http://www.yoctopuce.com/EN/virtualhub.php

```
# udev rules to allow write access to all users
# for Yoctopuce USB devices
SUBSYSTEM=="usb", ATTR{idVendor}=="24e0", MODE="0666"
```

### Exemple 2: 51-yoctopuce_group.rules

Cette règle va autoriser le groupe "yoctogroup" à accéder en lecture et écriture aux périphériques Yoctopuce USB. Les droits d'accès pour tous les autres périphériques ne seront pas modifiés. Si ce scénario vous convient, il suffit de copier le fichier "`51-yoctopuce_group.rules`" dans le répertoire "`/etc/udev/rules.d`" et de redémarrer votre système.

```
# udev rules to allow write access to all users of "yoctogroup"
# for Yoctopuce USB devices
SUBSYSTEM=="usb", ATTR{idVendor}=="24e0", MODE="0664", GROUP="yoctogroup"
```

## 12.3. Plateformes ARM: HF et EL

Sur ARM il existe deux grandes familles d'executables: HF (Hard Float) et EL (EABI Little Endian). Ces deux familles ne sont absolument pas compatibles entre elles. La capacité d'une machine ARM à faire tourner des exécutables de l'une ou l'autre de ces familles dépend du hardware et du système d'exploitation. Les problèmes de compatibilité entre ArmHL et ArmEL sont assez difficiles à diagnostiquer, souvent même l'OS se révèle incapable de distinguer un exécutable HF d'un exécutable EL.

Tous les binaires Yoctopuce pour ARM sont fournis pré-compilée pour ArmHF et ArmEL, si vous ne savez à quelle famille votre machine ARM apartient, essayez simplement de lancer un exécutable de chaque famille.

## 12.4. Les exemples de programmation n'ont pas l'air de marcher

La plupart des exemples de programmation de l'API Yoctopuce sont des programmes en ligne de commande et ont besoin de quelques paramètres pour fonctionner. Vous devez les lancer depuis l'invite de commande de votre système d'exploitation ou configurer votre IDE pour qu'il passe les paramètres corrects au programme [3].

## 12.5. Module alimenté mais invisible pour l'OS

Si votre YoctoHub-Ethernet est branché par USB et que sa LED bleue s'allume, mais que le module n'est pas vu par le système d'exploitation, vérifiez que vous utilisez bien un vrai câble USB avec les fils pour les données, et non pas un câble de charge. Les câbles de charge n'ont que les fils d'alimentation.

## 12.6. Another process named xxx is already using yAPI

Si lors de l'initialisation de l'API Yoctopuce, vous obtenez le message d'erreur "*Another process named xxx is already using yAPI*", cela signifie qu'une autre application est déjà en train d'utiliser les modules Yoctopuce USB. Sur une même machine, un seul processus à la fois peut accéder aux modules Yoctopuce par USB. Cette limitation peut facilement être contournée en utilisant un VirtualHub et le mode réseau [4].

[3] voir: http://www.yoctopuce.com/FR/article/a-propos-des-programmes-d-exemples
[4] voir: http://www.yoctopuce.com/FR/article/message-d-erreur-another-process-is-already-using-yapi

## 12.7. Déconnexions, comportement erratique

Si votre YoctoHub-Ethernet se comporte de manière erratique et/ou se déconnecte du bus USB sans raison apparente, vérifiez qu'il est alimenté correctement. Evitez les câbles d'une longueur supérieure à 2 mètres. Au besoin, intercalez un hub USB alimenté [5] [6].

## 12.8. Le module ne marche plus après une mise à jour ratée

Si une mise à jour du firmware de votre YoctoHub-Ethernet échoue, il est possible que le module ne soit plus en état de fonctionner. Si c'est le cas, branchez votre module en maintenant sont Yocto-Bouton pressé. La Yocto-LED devrait s'allumer en haute intensité et rester fixe. Relâchez le bouton. Votre YoctoHub-Ethernet devrait alors apparaître dans le bas de l'interface du virtualHub comme un module attendant une mise à jour de firmware. Cette mise à jour aura aussi pour effet de réinitialiser le module à sa configuration d'usine.

## 12.9. RegisterHub d'une instance de VirtualHub déconnecte la précédente

Si lorsque vous faire un YAPI.RegisterHub de VirtualHub et que la connexion avec un autre VirtualHub précédement enregistré tombe, vérifiez que les machines qui hébergent ces VirtualHubs ont bien un *hostname* différent. Ce cas de figure est très courant avec les machines dont le système d'exploitation est installé avec une image monolithique, comme les Raspberry Pi par exemple. L'API Yoctopuce utilise les numéros de série Yoctopuce pour communiquer et le numéro de série d'un VirtualHub est créé à la volée à partir du *hostname* de la machine qui l'héberge.

## 12.10. Commandes ignorées

Si vous avez l'impression que des commandes envoyées à un module Yoctopuce sont ignorées, typiquement lorsque vous avez écrit un programme qui sert à configurer ce modules Yoctopuce et qui envoie donc beaucoup de commandes, vérifiez que vous avez bien mis un YAPI.FreeAPI() à la fin du programme. Les commandes sont envoyées aux modules de manière asynchrone grâce à un processus qui tourne en arrière plan. Lorsque le programme se termine, ce processus est tué, même s'il n'a pas eu le temps de tout envoyer. En revanche API.FreeAPI() attend que la file d'attente des commandes à envoyer soit vide avant de libérer les ressources utilisées par l'API et rendre la main.

## 12.11. Certains câbles réseaux ne fonctionnent pas..

..avec le YoctoHub-Ethernet alors qu'ils marchent très bien avec d'autres équipements. Contrairement à la plupart des équipements réseaux modernes, le YoctoHub-Ethernet n'a pas de fonction auto-MDIX. Vous devez impérativement utiliser le type de câble correspondant à votre utilisation: un câble droit (les plus courants) pour connecter le YoctoHub-Ethernet à un switch réseau et un câble croisé pour le connecter directement à la prise réseau d'un ordinateur.

## 12.12. Impossible de contacter les sous-modules par USB

Le but du YoctoHub-Ethernet est de fournir une connectivité réseau aux sous-modules qui lui sont connectés, il ne se comporte pas comme un hub USB. Le port USB du YoctoHub-Ethernet ne sert qu'à l'alimenter et le configurer. Pour accéder aux modules connectés au hub, vous devez impérativement passer par une connexion réseau.

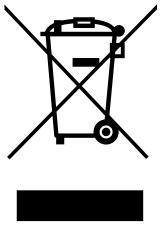
[5] voir: http://www.yoctopuce.com/FR/article/cables-usb-la-taille-compte
[6] voir: http://www.yoctopuce.com/FR/article/combien-de-capteurs-usb-peut-on-connecter

## 12.13. Network Readiness coincé à 3- LAN ready

Vérifiez que votre connexion Internet sortante fonctionne et que vous n'avez pas un callback invalide défini dans la configuration to YoctoHub-Ethernet.

## 12.14. Module endommagé

Yoctopuce s'efforce de réduire la production de déchets électroniques. Si vous avez l'impression que votre YoctoHub-Ethernet ne fonctionne plus, commencez par contacter le support Yoctopuce par e-mail pour poser un diagnostic. Même si c'est suite à une mauvaise manipulation que le module a été endommagé, il se peut que Yoctopuce puisse le réparer, et ainsi éviter de créer un déchet électronique.



**Déchets d'équipements électriques et électroniques (DEEE)** Si voulez vraiment vous débarasser de votre YoctoHub-Ethernet, ne le jetez pas à la poubelle, mais ramenez-le à l'un des points de collecte proposé dans votre région afin qu'il soit envoyé à un centre de recyclage ou de traitement spécialisé.

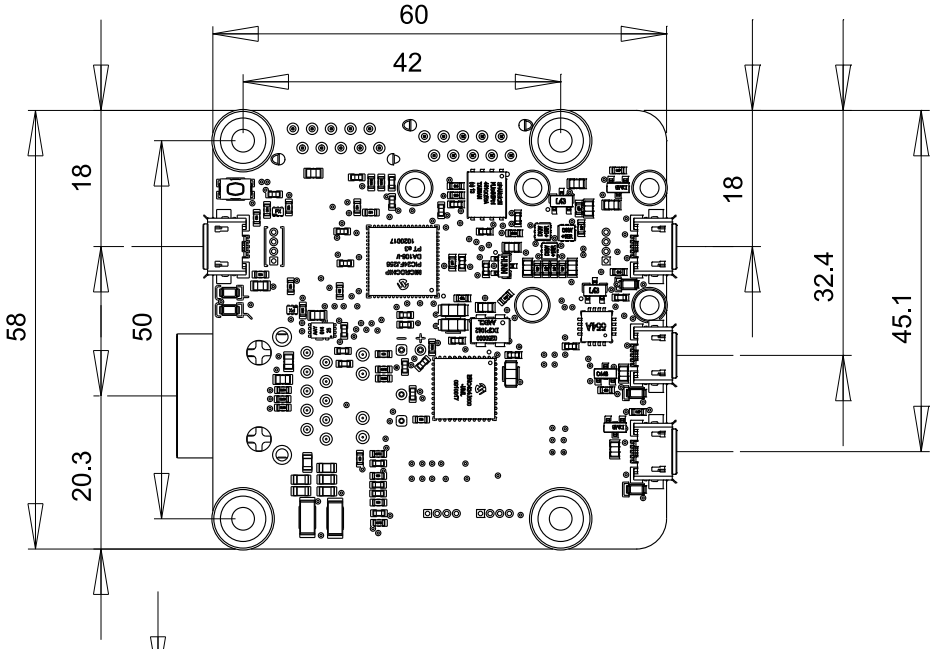
# 13. Caractéristiques

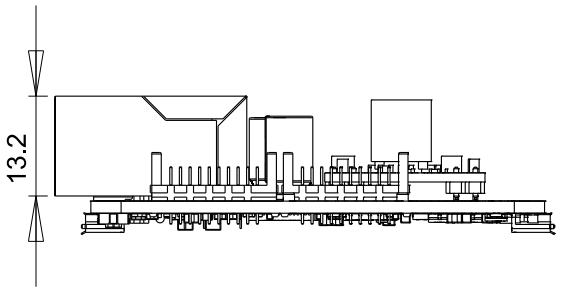
Vous trouverez résumées ci-dessous les principales caractéristiques techniques de votre module YoctoHub-Ethernet

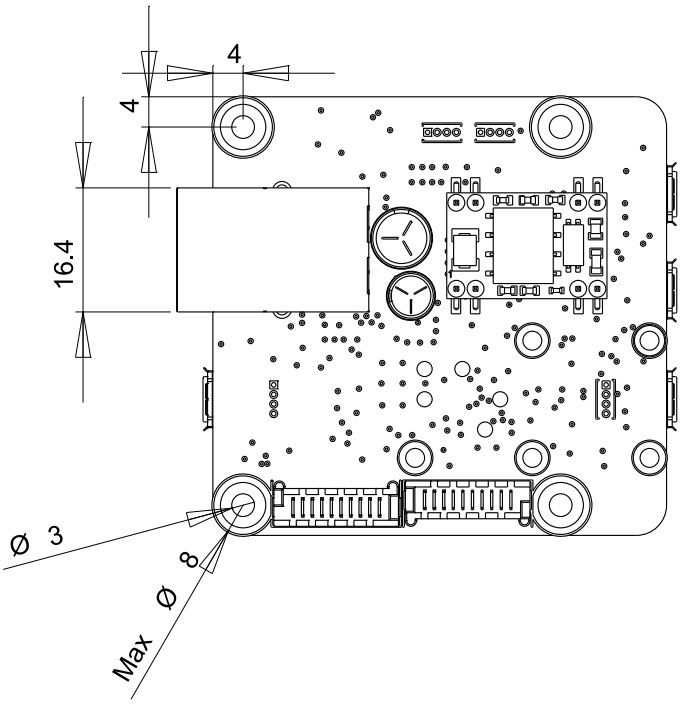
| | |
|---|---|
| Identifiant produit | YHUBETH1 |
| Révision matérielle† | Rev. E |
| Connecteur USB | micro-B |
| Epaisseur | 22 mm |
| Largeur | 58 mm |
| Longueur | 60 mm |
| Poids | 34 g |
| Canaux | 3 ports |
| Courant Max (continu) | 2 A |
| Classe de protection selon IEC 61140 | classe III |
| Temp. de fonctionnement normale | 5...40 °C |
| Temp. de fonctionnement étendue‡ | -30...85 °C |
| Consommation USB | 130 mA |
| Connection réseau | Fast-Ethernet |
| Conformité RoHS | RoHS III (2011/65/UE+2015/863) |
| USB Vendor ID | 0x24E0 |
| USB Device ID | 0x000E |
| Boîter recommandé | YoctoBox-HubEth-Transp |
| Code tarifaire harmonisé | 9032.9000 |
| Fabriqué en | Suisse |

† Ces spécifications correspondent à la révision matérielle actuelle du produit. Les spécifications des versions antérieures peuvent être inférieures.

‡ La plage de température étendue est définie d'après les spécifications des composants et testée sur une durée limitée (1h). En cas d'utilisation prolongée hors de la plage de température standard, il est recommandé procéder à des tests extensifs avant la mise en production.




All dimensions are in mm
Toutes les dimensions sont en mm

# YoctoHub-Ethernet

A4

Scale
1:1
Echelle