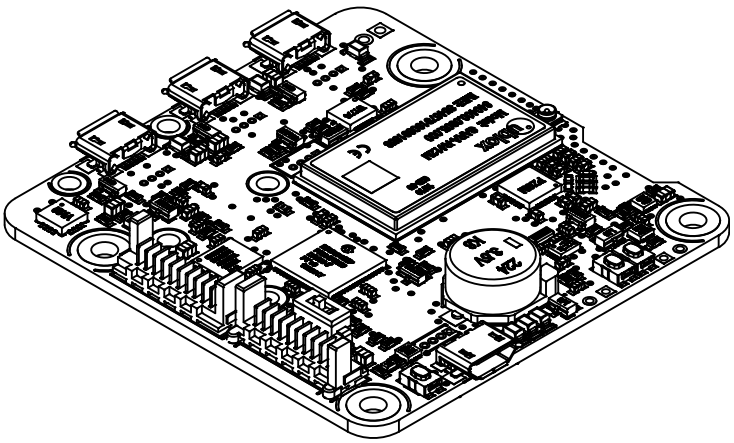
YoctoHub-GSM-4G

Mode d'emploi

Table des matières

1. Introduction

Le YoctoHub-GSM-4G est un module électronique de 60x58mm qui permet de contrôler d'autres modules Yoctopuce à travers une connection cellulaire de type 4G (LTE-M ou NB-IoT) ou 2G. En 4G, le YoctoHub-GSM-4G supporte les bandes LTE 2, 3, 4, 5, 8, 12, 13 et 20 en LTE-M (LTE Cat M1) et NB-IoT (LTE Cat NB1). En 2G, le YoctoHub-GSM-4G supporte les bandes GSM 850 MHz, E-GSM 900 MHz, DCS 1800 MHz et PCS 1900 MHz (quad band) en GPRS et EGPRS (EDGE)[1]



Le YoctoHub-GSM-4G

Le YoctoHub-GSM-4G 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. Les 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-GSM-4G 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-GSM-4G.

Yoctopuce vous remercie d'avoir fait l'acquisition de ce YoctoHub-GSM-4G et espère sincèrement qu'il vous donnera entière satisfaction. Les ingénieurs Yoctopuce se sont donnés beaucoup de mal pour que votre YoctoHub-GSM-4G soit facile à installer n'importe où et soit facile utiliser en toutes

[1] Pour une liste détaillée des bandes de fréquence supportées par pays, consultez les pages Wikipedia https://en.wikipedia.org/wiki/LTE_frequency_bands et https://en.wikipedia.org/wiki/GSM_frequency_bands . Notez que certains pays n'ont pas encore mis en place le support LTE-M et/ou NB-IoT.

circonstances. Néanmoins, si ce module venait à vous décevoir, n'hésitez pas à contacter le support Yoctopuce[2].

1.1. Accessoires optionnels

Il existe un certain nombre d'accessoires qui vous aideront à tirer le meilleur parti de votre YoctoHub-GSM-4G.

Vis et entretoises

Pour fixer le module YoctoHub-GSM-4G à un support, vous pouvez placer des petites vis de 3mm avec une tête de 8mm au maximum dans les trous prévus ad-hoc. Il est conseillé de les visser dans des entretoises filetées, que vous pourrez fixer sur le support. Vous trouverez plus de détail à ce sujet dans le chapitre concernant le montage et la connectique. Micro-hub USB

Câble USB MicroB-MicroB

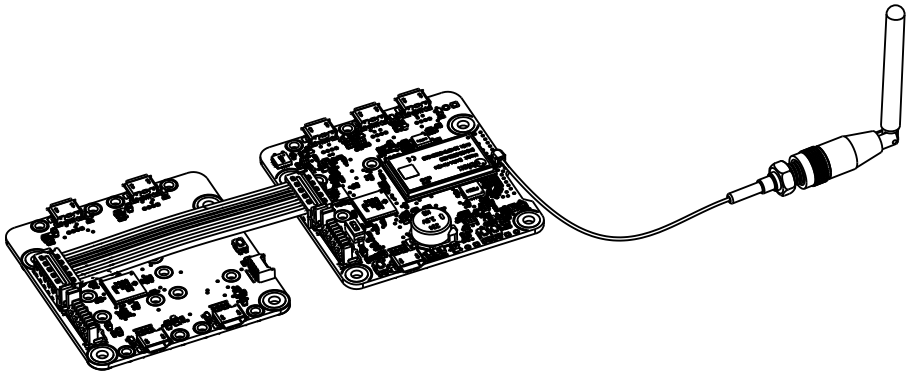
Les ports descendant du YoctoHub-GSM-4G sont au format micro-B, tous les produits Yoctopuce sont équipés de connecteur micro-B. Cela signifie que vous aurez besoin d'un câble se terminaux par une prise Micro-B à chaque extrémité. Ces câbles ne sont pas très répandus dans le commerce, mais vous pourrez en trouver sur le magasin en ligne de Yoctopuce[3].

Connecteur au pas 1.27mm

L'utilisation de câbles USB pour connecter les sous modules est très pratique, mais cela reste une solution assez volumineuse. Sur le PCB du YoctoHub-GSM-4G, vous trouverez a proximité de chaque connecteur USB une empreinte permettant de souder un connecteur au pas 1.27 ou 1.25mm. Ce qui permet d'interconnecter les modules avec un câblage beaucoup compact. Ce genre de connecteur est très standard, et peut être acheté dans à peu près n'importe que magasin d'électronique. Yoctopuce commercialise une ensemble connecteur + câble de 11cm sous la référence *1.27-1.27-11*.

YoctoHub-Shield

Le YoctoHub-GSM-4G dispose de trois ports permettant de brancher trois sous-modules. Il est possible d'augmenter significativement cette capacité en utiliser des extension nommées *YoctoHub-Shield*. Chacun de ces shields ajoute quatre ports supplémentaires, et il est possible d'en chainer jusqu'à dix. Consulter la documentation du YoctoHub-Shield pour plus de détails.

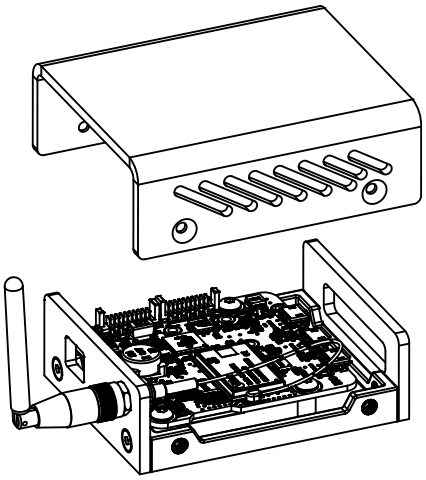


Le YoctoHub-Shield ajoute des ports à votre YoctoHub-GSM-4G.

Boîtier

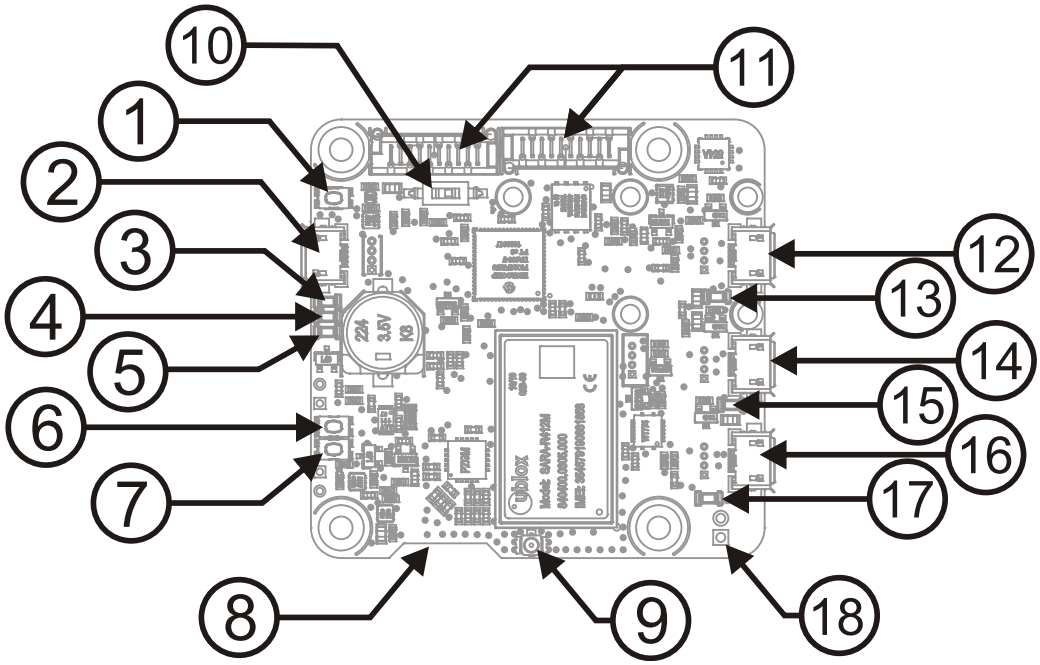
Votre YoctoHub-GSM-4G a été conçu pour pouvoir être installé tel quel dans votre projet. Néanmoins Yoctopuce commercialise des boîtiers spécialement conçus pour les modules Yoctopuce. Vous trouverez plus d'informations à propos de ces boîtiers sur le site de Yoctopuce. Le boîtier recommandé pour votre YoctoHub-GSM-4G est le modèle YoctoBox-HubWlan-Transp.

[2] support@yoctopuce.com
[3] *USB-OTG-MicroB-MicroB-20* et *USB-OTG-MicroB-MicroB-100*



Votre YoctoHub-GSM-4G peut être installé dans un boitier.

2. Présentation



1: Yocto-bouton
2: Port USB de contrôle + alimentation
3: Yocto-LED
4: Indicateur de sur-consommation
5: Indicateur de transfert réseau
6: Touche réveil
7: Touche mise en sommeil
8: Support pour carte SIM (dessous)
9: Connecteur d'antenne
10: Neutralisation de la mise en sommeil
11: Connexion dorsale
12: Port descendant 1
13: Indicateur port descendant 1
14: Port descendant 2
15: Indicateur port descendant 2
16: Port descendant 3
17: Indicateur port descendant 3
18: Contacts d'activation du mode avion

2.1. Les éléments du YoctoHub-GSM-4G

Le numéro de série

Chaque Yocto-module a un numéro de série unique attribué en usine, pour les modules YoctoHub-GSM-4G ce numéro commence par YHUBGSM5. 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-GSM-4G 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-GSM-4G 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-GSM-4G 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-GSM-4G et le module Yoctopuce n'est pas de l'USB mais un protocole propriétaire plus léger. De ce fait le YoctoHub-GSM-4G 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-GSM-4G 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

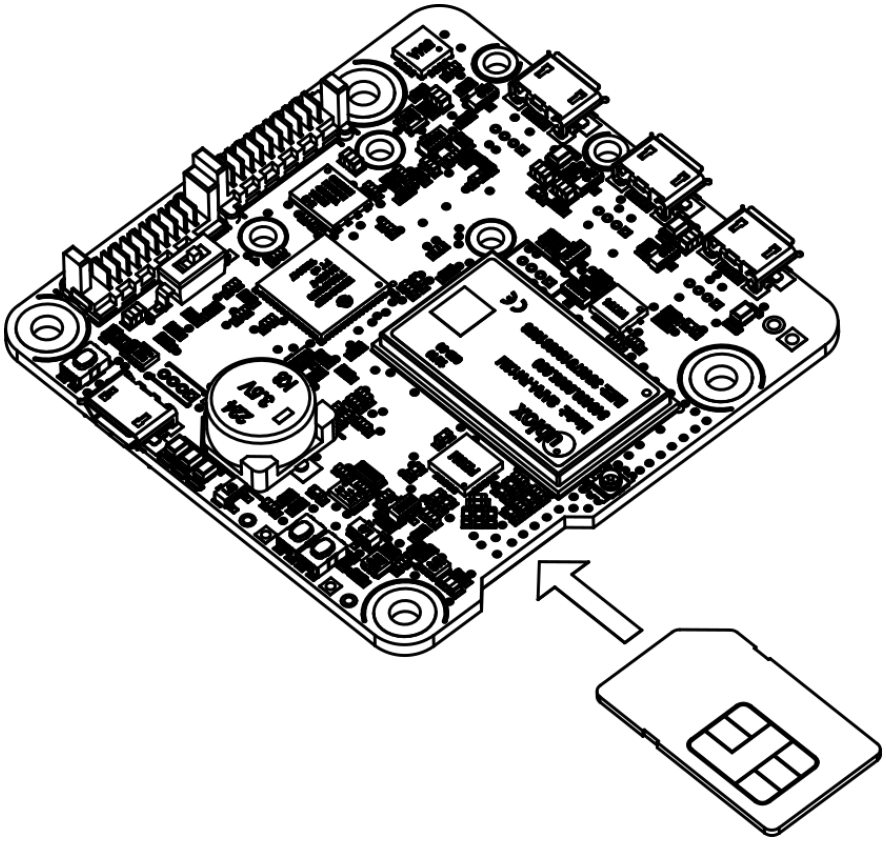
[1] court-court-court long-long-long court-court-court
[2] support@yoctopuce.com
[3] Le Micro-USB-Hub fabriqué par Yoctopuce est un hub USB standard et ne fonctionnera pas avec le YoctoHub-GSM-4G.
[4] www.yoctopuce.com/FR/products/yoctohub-shield

souder un fil USB directement dans les trous espacés de 1.27mm prévus à cet effet, près du connecteur.

Le support pour la carte SIM

Pour vous connecter à un réseau cellulaire GSM, vous devrez insérer dans votre YoctoHub-GSM-4G une carte SIM autorisant la connexion au réseau cellulaire, associée à un abonnement permettant le transfert de données. Le support à carte SIM est prévu pour le format mini-SIM le plus standard,aussi appelé 2FF. Il existe des adaptateurs permettant l'utilisation des Micro-SIM ou Nano-SIM, ces adaptateur peuvent être achetés dans n'importe quel magasin de téléphone portable. Le support SIM est de type *push-push*: pressez pour insérer la SIM jusqu'à-ce qu'elle soit en position et produise un petit clic. Pressez une deuxième fois pour éjecter la SIM de son support.

Vous devez insérer la carte SIM avec les contacts métalliques contre le circuit imprimé.

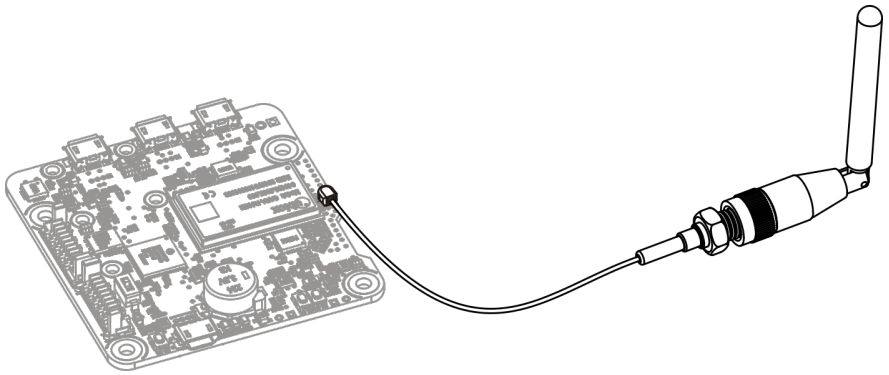


Sens d'insertion de la carte SIM dans le YoctoHub-GSM-4G.

Le connecteur d'antenne

Le YoctoHub-GSM-4G dispose d'un connecteur d'antenne coaxial ultra miniature (UFL). Prenez grand soin du connecteur UFL, il est fragile et n'est pas conçu pour supporter beaucoup de cycles de connexion/déconnexion. Le YoctoHub-GSM-4G est livré en standard avec un petit câble UFL vers SMA femelle[5] et une antenne correspondante SMA mâle[6]. Vous pouvez utiliser une autre antenne de votre choix, pour autant qu'elle soit conçue pour la gamme de fréquence utilisée dans votre pays pour le réseau cellulaire et qu'elle ait le bon connecteur. Prenez garde aussi au fait que l'utilisation d'antennes à fort gain peut vous amener à émettre un signal supérieur à la norme autorisée dans votre pays.

[5] filetage extérieur et tube femelle au centre
[6] filetage intérieur et pin mâle au centre



Connexion de l'antenne.

Indicateur de sur-consommation

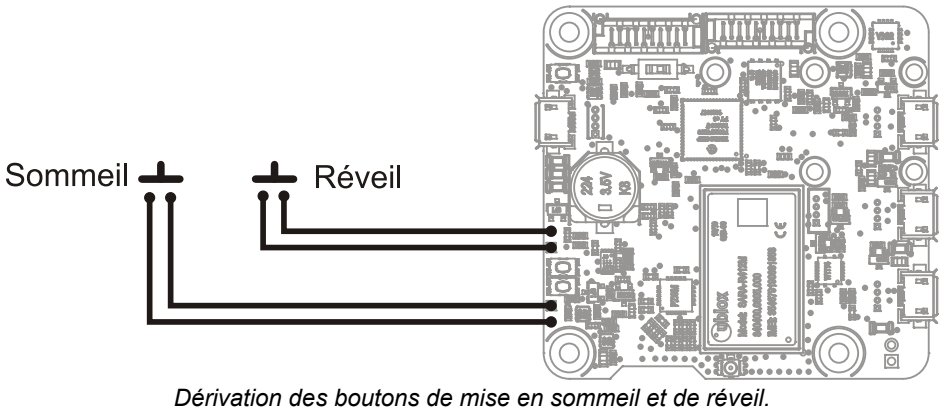
Le YoctoHub-GSM-4G analyse en permanence sa consommation. S'il détecte une consommation globale de plus de 2A suite à une surcharge sur un des ports descendants par exemple, il va automatiquement désactiver tous les ports descendants et allumer l'indicateur de sur-consommation. Pour isoler la source 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-GSM-4G 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.

Mise en sommeil

En moyenne, le YoctoHub-GSM-4G consomme environ 0,5 Watt (50mA), auquel il faut ajouter la consommation des modules qui lui sont connectés. Mais il est capable de se mettre en sommeil pour réduire sa consommation d'énergie au strict minimum, et de se réveiller à une heure précise ou lorsqu'un contact extérieur est fermé. Cette fonctionnalité est très utile pour construire des installations de mesure fonctionnant sur batterie. Lorsque que le YoctoHub-GSM-4G est en sommeil, la quasi totalité de l'électronique du module ainsi que les modules Yoctopuce connectés sont hors tension, ce qui réduit sa consommation totale à 75 µW (15 µA).

La mise en sommeil et le réveil peuvent être soit programmés sur base horaire, soit contrôlés par logiciel, soit contrôlés manuellement à l'aide de deux boutons poussoirs présents sur le circuit du YoctoHub-GSM-4G. Vous y trouverez aussi deux paires de contacts qui permettent de dériver ces deux boutons.



Dérivation des boutons de mise en sommeil et de réveil.

Le YoctoHub-GSM-4G dispose d'un interrupteur qui permet de désactiver au niveau hardware la fonctionnalité de mise en sommeil. Cette fonctionnalité est utile en particulier durant les phases de développement/déverminage de votre projet, ainsi que pour effectuer les mises à jour du firmware.

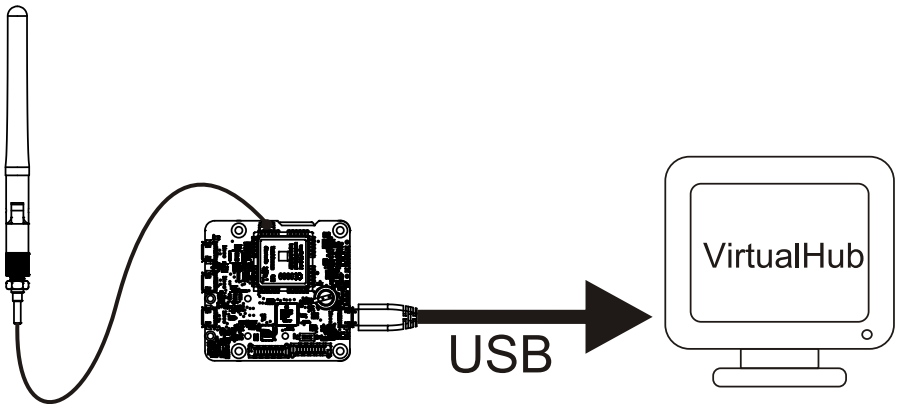
3. Premiers pas

Ce chapitre a pour but de vous aider à connecter et configurer votre YoctoHub-GSM-4G pour la première fois.

3.1. Configuration manuelle

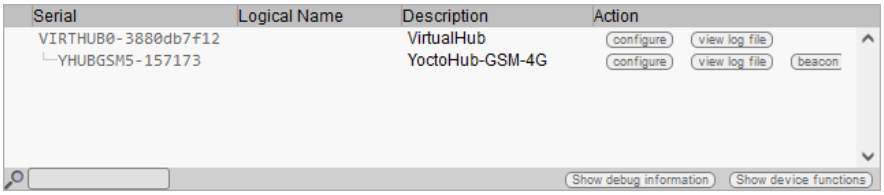
Vous pouvez configurer votre YoctoHub-GSM-4G via son port de contrôle USB, en utilisant VirtualHub[1].

Lancez VirtualHub sur votre ordinateur favori et raccordez votre ordinateur au port *power / control port* du YoctoHub-GSM-4G. Vous aurez besoin d'un câble USB A-MicroB.



Configuration: raccordez par USB votre YoctoHub-GSM-4G à un ordinateur

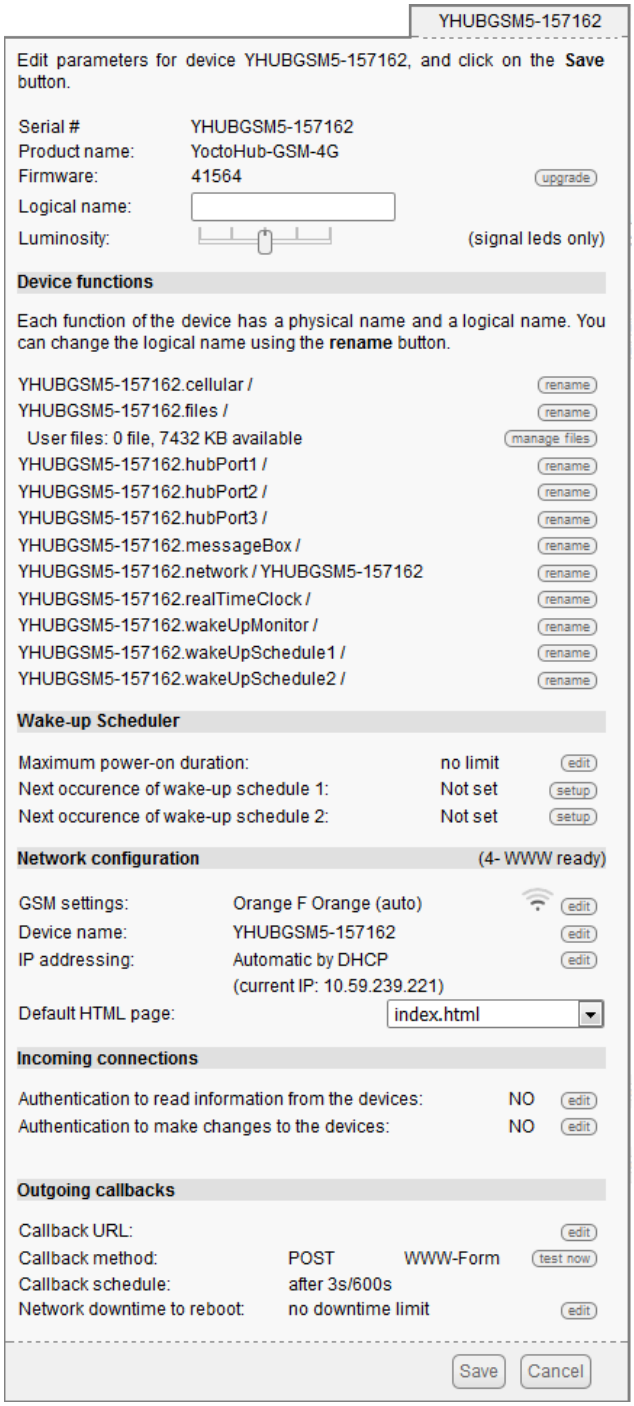
Lancez alors votre browser favori sur l'URL de votre VirtualHub. Il s'agit généralement de *http://127.0.0.1:4444*. Vous obtiendrez la liste des modules Yoctopuce connectés par USB, dont votre YoctoHub-GSM-4G.



Liste des modules Yoctopuce raccordés par USB à votre ordinateur, dont votre YoctoHub-GSM-4G

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

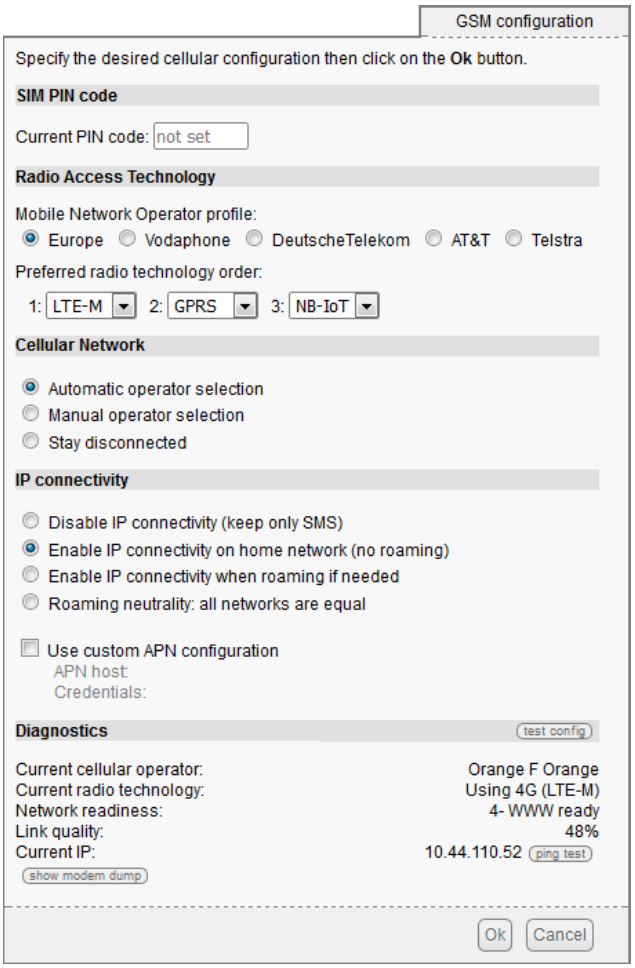
Cliquez sur le bouton **configure** correspondant à votre YoctoHub-GSM-4G, votre obtiendrez la fenêtre de configuration du module. Cette fenêtre comporte une section **Network configuration**.



Fenêtre de configuration du module YoctoHub-GSM-4G

Connexion au réseau cellulaire

La première chose à faire consiste à configurer votre YoctoHub-GSM-4G pour qu'il se connecte à votre réseau cellulaire. Pour cela cliquez sur le bouton **edit** correspondant à **GSM settings** dans la section **Network configuration**, et la fenêtre de configuration du réseau cellulaire apparaît:



Fenêtre de configuration du réseau cellulaire.

Si nécessaire, vous pouvez y entrer le code PIN correspondant à la carte SIM introduite dans le YoctoHub-GSM-4G. Attention, vous n'avez en général que trois essais pour configurer le bon code PIN. Après ces trois essais, vous devrez réinitialiser la SIM à l'aide de son code PUK.

Comme le YoctoHub-GSM-4G est capable de se connecter à plusieurs types de réseau différents sur différentes bandes de fréquence, il vous faut choisir la technologie de connection souhaitée et le profile d'opérateur, définissant les bandes de fréquences qui seront utilisées ainsi que certains paramètres de communication plus spécifiques. Les profiles supportés sont les suivants:

AT&T
: LTE-M (bandes 2, 4, 5 et 12) et (E)GPRS

AT&T 2-4-12
: LTE-M (bandes 2, 4 et 12) et (E)GPRS

Europe
: LTE-M (bandes 3, 8, 20), NB-IoT (bandes 3, 8, 20) et (E)GPRS

DT (Deutsche Telekom)
: LTE-M (bandes 3, 8, 20), NB-IoT (bandes 3, 8, 20) et (E)GPRS

Vodafone
: LTE-M (bandes 3, 8, 20), NB-IoT (bandes 3, 8, 20) et (E)GPRS

Orange France
: LTE-M (bandes 3, 8, 20), NB-IoT (bandes 3, 8, 20) et (E)GPRS

VIVO
: LTE-M (bandes 3, 28) et NB-IoT (bande 28)

TIM Brasil
: NB-IoT (bande 3 et 28)

Claro Brasil
: LTE-M (bandes 3, 28) et NB-IoT (bandes 3, 5 et 28)

En 2G, le YoctoHub-GSM-4G supporte les bandes GSM 850 MHz, E-GSM 900 MHz, DCS 1800 MHz et PCS 1900 MHz (quad band) en GPRS et EGPRS (EDGE).

Si vous prévoyez d'utiliser l'un des opérateurs nommé dans les profiles, choisissez le profile correspondant. Pour le reste de l'Europe, choisissez le profile générique Europe qui couvre au mieux tous les autres pays d'Europe. En plus du choix du profile, vous pouvez choisir la ou les technologies radio préférentielles à utiliser, pour accélérer la recherche de connexion ou maximiser le débit de données disponible. Si vous n'activez qu'une seule technologie, elle seule sera utilisée. Attention donc à ne pas forcer l'utilisation d'une technologie qui n'est pas disponible dans votre environnement. Si vous laissez plusieurs technologies, elles seront recherchées dans l'ordre choisi mais il est néanmoins possible que le second choix soit retenu même s'il aurait pu être possible de se connecter au premier choix.

En plus du choix de la technologie radio, vous pouvez spécifier quel opérateur spécifique utiliser. Souvent, la carte SIM "sait" pour quel opérateur elle est prévue, et il est possible de garder simplement la sélection automatique de l'opérateur. Toutefois, en zone frontalière, il se peut que la carte soit tentée d'utiliser un émetteur étranger plus puissant, mais aussi plus cher à l'utilisation. Pour éviter cela, vous pouvez effectuer un choix d'opérateur manuel de votre réseau domestique. Pour que le module puisse se connecter à un opérateur sur les réseau LTE-M et NB-IoT, il faut non seulement que la carte SIM l'y autorise, mais aussi que l'APN soit configuré correctement (comme expliqué ci-dessous). Sinon, le module n'arrivera probablement même pas à s'attacher à l'opérateur, même juste pour la fonction SMS.

Vous pouvez aussi spécifier dans votre YoctoHub-GSM-4G le contexte dans lequel vous désirez activer la connection IP (transfert de données). Vous pouvez soit la désactiver complètement si vous n'êtes intéressé qu'à l'utilisation de SMS, soit l'activer sur le réseau propre de la carte SIM, soit autoriser l'utilisation des données sur les réseaux tiers (roaming). Prenez garde si vous activez cette dernière option, car elle peut engendrer rapidement des coûts importants ! Notez toutefois que dans certains cas, les SMS peuvent n'être fonctionnels que si la connexion IP est fonctionnelle. De plus, avec certaines cartes SIM, les SMS peuvent ne pas être disponibles du tout, selon le contrat fourni par l'opérateur qui a émis la cartes SIM.

Selon votre SIM et votre opérateur cellulaire, il est possible que vous deviez configurer un APN (*Access Point Name*), qui correspond à la passerelle vers internet sur votre réseau cellulaire. Il s'agit d'un nom arbitraire, décidé par votre opérateur, et auquel s'ajoute parfois un nom d'utilisateur et un mot de passe. Il est impossible de lister dans ce mode d'emploi le nom de l'APN à utiliser avec chaque opérateur, mais si vous faites une recherche sur internet avec le nom de votre opérateur cellulaire et le mot "APN", vous trouverez très facilement le nom de l'APN à utiliser ainsi que le nom d'utilisateur et mot de passe éventuel. Le site apnchanger.org contient ces informations pour les principaux opérateurs de nombreux pays.

Après avoir entré les paramètres de réseau sans fil et éventuellement les avoir testés, vous pouvez cliquer sur le bouton **Ok** pour fermer cette fenêtre de configuration et retourner à la fenêtre de configuration générale.

Différence entre un réseau cellulaire et un réseau usuel

Important: Le réseau cellulaire auquel est connecté votre YoctoHub-GSM-4G n'est pas strictement équivalent à une connexion internet usuelle. En effet, l'adresse IP qui est attribuée aux modems cellulaires est quasiment toujours une adresse IP *non routée*. Le YoctoHub-GSM-4G voit tout le réseau Internet, mais Internet ne dispose pas d'adresse publique pour le contacter. Ceci signifie que vous ne pourrez pas vous connecter à distance sur votre YoctoHub-GSM-4G depuis n'importe quel ordinateur, juste en tapant son adresse IP dans un navigateur internet.

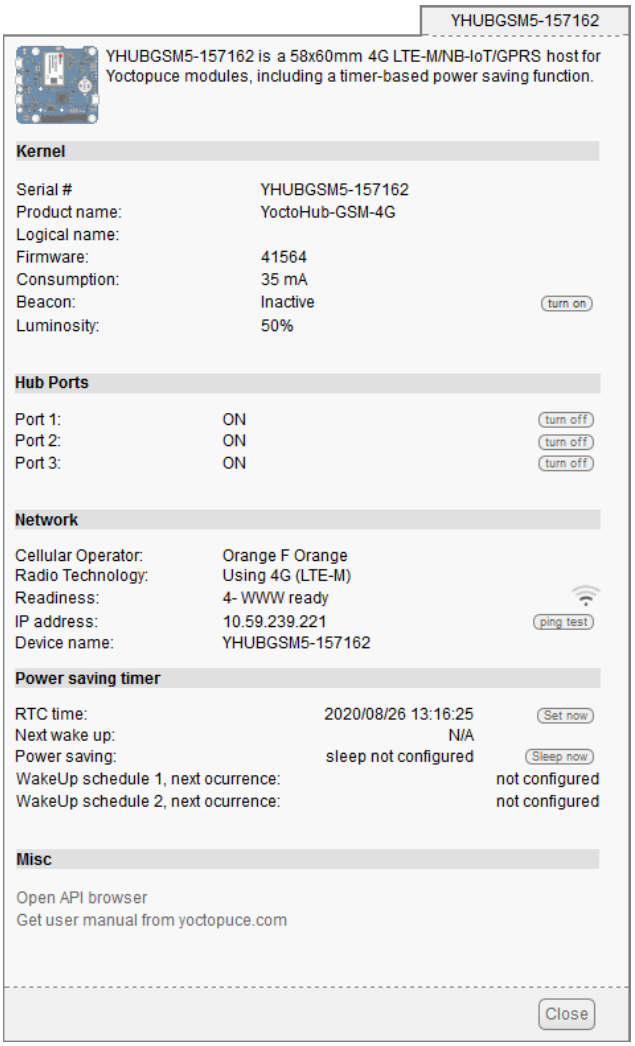
Une des solutions qui permet de palier à ce problème consiste à obtenir de votre opérateur cellulaire une carte SIM permettant une connection directe à travers un réseau privé virtuel. C'est souvent une option qui implique une surcharge conséquente du prix des communications.

Une autre solution consiste à utiliser l'outil gratuit *VirtualHub (for web)* disponible sur le site de Yoctopuce. De plus amples informations sur cette manière de procéder sont disponibles dans le chapitre Utilisation.

3.2. Fenêtre d'état du hub

Il est possible de connaitre l'état général du hub, c'est-à-dire son nom logique, l'état du réseau, l'état des ports, etc. Pour cela, retournez simplement à la liste des modules.

Cliquez sur le numéro de série correspondant à votre YoctoHub-GSM-4G. Cela ouvrira la fenêtre des détails de votre module:



Les propriétés du YoctoHub-GSM-4G

Cette fenêtre comporte une section qui relate l'état de la partie réseau du YoctoHub-GSM-4G. 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- Le module ne trouve pas de réseau cellulaire. Dans ce cas, vérifiez que:
  - vous êtes à un endroit où l'on peut recevoir du réseau 4G (LTE-M ou NB-IoT) ou 2G
  - vous avez introduit une carte SIM autorisée à se connecter au réseau choisi
  - vous avez configuré le code PIN de la carte SIM dans le YoctoHub-GSM-4G
  - vous n'avez pas désactivé la radio (mode avion)
  - vous avez bien configuré la technologie radio du YoctoHub-GSM-4G pour le service désiré
  - la bande de fréquences utilisée par l'opérateur pour ce service est supportée par le profil de connection choisi sur le YoctoHub-GSM-4G
  - vous avez bien connecté une antenne compatible avec cette bande de fréquence

- 1- network exists: un réseau cellulaire a été détecté, mais le module n'y est pas encore accepté. Si cet état persiste, vérifiez que votre SIM est valable et que l'opérateur cellulaire choisi correspond.
- 2- network linked: le YoctoHub-GSM-4G a pu se connecter au réseau GSM, mais n'a pas encore établi de connection IP. Si cet état persiste, vérifiez vos réglages APN.
- 3- LAN ready: le réseau local est opérationnel (connection IP avec l'opérateur de téléphonie)
- 4- WWW ready: le module a pu vérifier la connectivité à Internet en se connectant à un serveur de temps (NTP), ou en établissant un callback HTTP si vous l'avez configuré.

Si votre YoctoHub-GSM-4G passe bien à l'état *WWW Ready*, cela signifie que votre connection internet cellulaire fonctionne correctement. Vous pouvez alors passer aux étapes suivantes: connecter les modules Yoctopuce souhaités (capteurs et/ou actuateurs) et configurer les interactions avec l'extérieur.

3.3. Configuration automatisée

Il est possible d'industrialiser la configuration réseau du YoctoHub-GSM-4G. 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

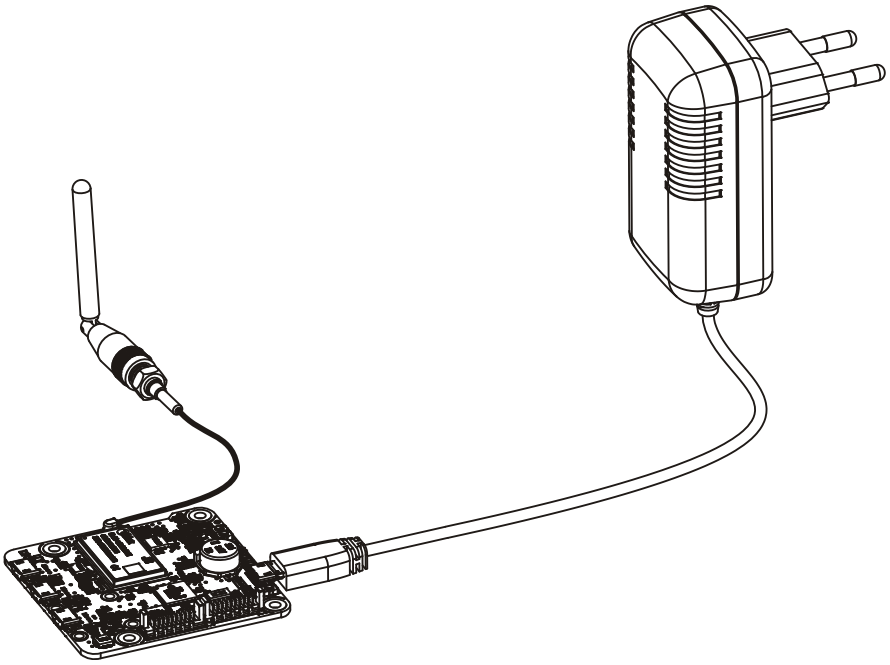
Alimentation

Le YoctoHub-GSM-4G doit être alimenté par la prise USB de contrôle.

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-GSM-4G consomme environ 50mA, auxquels il faudra ajouter la consommation de chaque sous-module. Le YoctoHub-GSM-4G 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.

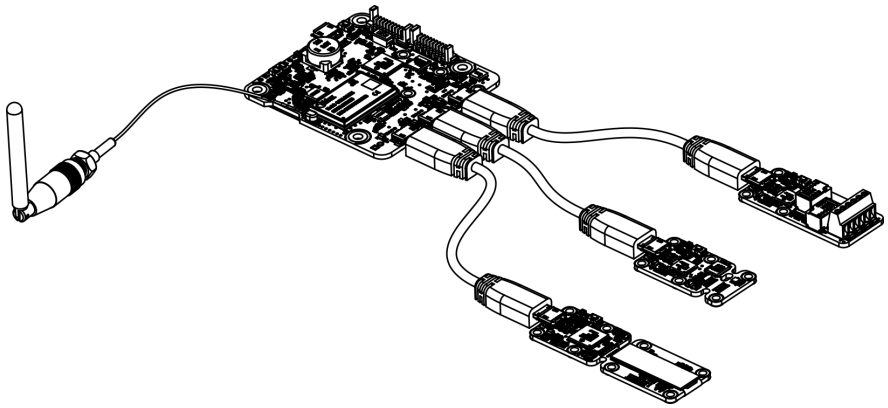
[2] http://www.yoctopuce.com/FR/libraries.php



Le YoctoHub-GSM-4G peut être alimenté par un chargeur USB

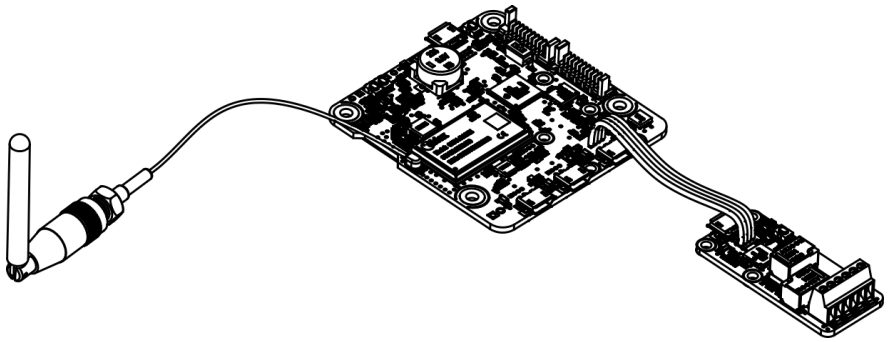
Sous-modules

Le YoctoHub-GSM-4G 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.



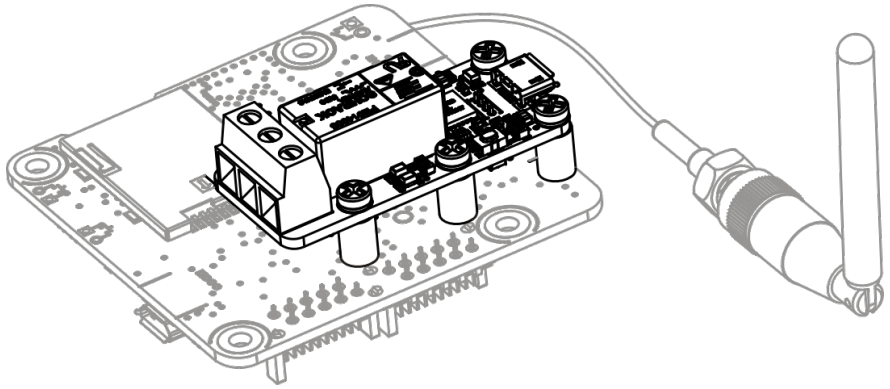
Connexion des sous-modules à l'aide de câbles USB

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-GSM-4G, 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-GSM-4G est conçu pour que vous puissiez fixer un module simple largeur directement dessus. Vous aurez besoin de vis, d'entretoises[3] et d'un connecteur au pas 1.27mm[4]. Vous pouvez ainsi transformer un module Yoctopuce USB en en module réseau tout en gardant un format très compact.



Fixation d'un module directement sur le hub

Attention, le YoctoHub-GSM-4G est conçu pour piloter des modules Yoctopuce uniquement. En effet le protocole utilisé entre le YoctoHub-GSM-4G 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-GSM-4G, le port en question sera automatiquement désactivé pour éviter d'endommager le périphérique.

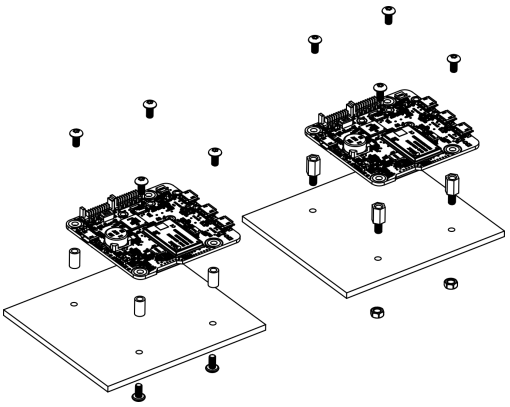
[3] http://www.yoctopuce.com/FR/products/accessoires-et-connectique/fix-2-5mm
[4] http://www.yoctopuce.com/FR/products/accessoires-et-connectique/board2board-127

4. Montage

Ce chapitre fournit des explications importantes pour utiliser votre module YoctoHub-GSM-4G 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.



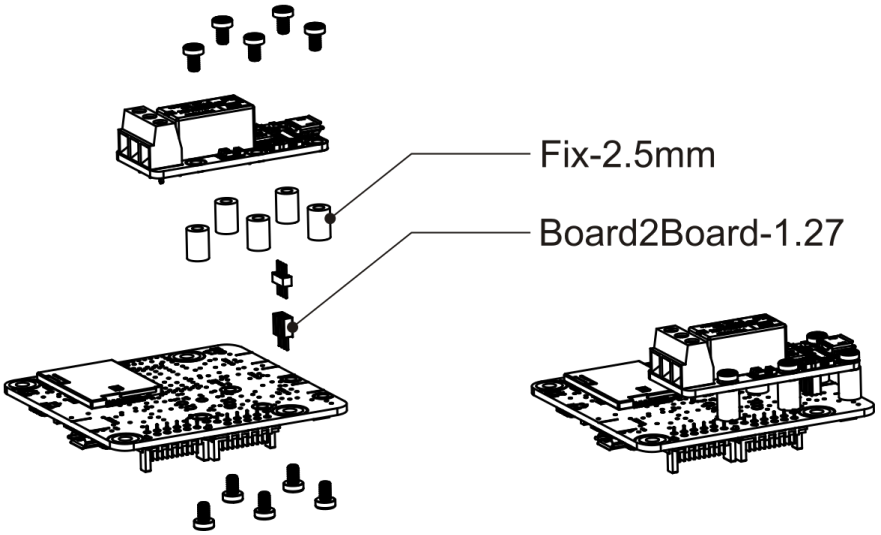
Exemples de montage sur un support.

Le module YoctoHub-GSM-4G 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-GSM-4G, 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-GSM-4G 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-GSM-4G 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.



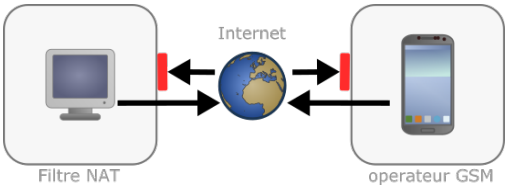
Fixation d'un module directement sur le hub

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-GSM-4G, et le contact carré sur sur le module doit être connecté au contact carré sur le port descendant du YoctoHub-GSM-4G. 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-GSM-4G 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.

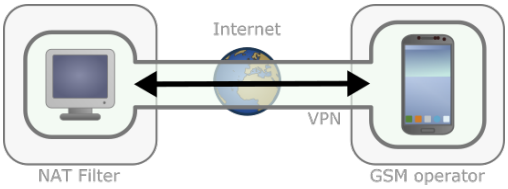
5. Utilisation

Le YoctoHub-GSM-4G est l'équivalent technique d'un YoctoHub-Wireless pour le GSM, mais un réseau GSM est loin d'être l'équivalent d'un réseau WiFi. La grosse différence se situe au niveau de l'adressage. Les opérateurs téléphoniques n'affectent pour ainsi dire jamais d'adresses IP publiques aux terminaux qu'ils gèrent. Tous les terminaux se trouvent chacun derrière une espèce de NAT qui empêche qu'ils soient contactés directement depuis l'extérieur. En clair, un smart-phone peut accéder à internet, mais il ne peut pas être contacté directement depuis internet, en tous cas pas avec un abonnement normal. Le mécanisme est similaire au filtre NAT qui empêche n'importe qui d'accéder directement aux machines qui se trouvent chez vous, sauf que là, vous ne pouvez pas ajouter une redirection de port sur le routeur. Cette limitation s'applique aussi au YoctoHub-GSM: avec un contrat de téléphonie mobile normal, il y a peu de chance que vous puissiez contacter directement un YoctoHub-GSM-4G depuis votre ordinateur à la maison.



En temps normal, les opérateurs limitent l'accès direct à leurs terminaux depuis internet

En revanche, certains opérateurs de téléphonie mobile proposent des contrats spéciaux incluant un VPN avec un accès direct à un parc de terminaux. Avec ce genre de contrat, il est possible de contacter directement le YoctoHub-GSM-4G comme si c'était un simple YoctoHub-Ethernet ou YoctoHub-Wireless connecté à votre réseau local. Mais sachez que ce genre de contrat est généralement réservé aux grandes infrastructures, ce qui les mets hors de portée des simples particuliers. Si vous souhaitez demander plus de précisions à votre opérateur favori, sachez que le mot magique à prononcer pour obtenir le bon interlocuteur est: M2M pour machine-to-machine.



Les opérateurs proposent parfois un service VPN, permettant l'accès direct aux terminaux

5.1. Utilisation courante, sans VPN

Principes généraux

Si vous ne pouvez pas vous offrir un accès direct à votre YoctoHub-GSM-4G, cela ne vous empêchera pas de l'utiliser. Le YoctoHub-GSM-4G dispose du même système de callbacks que les autres hubs Yoctopuce. Ainsi, il est capable de poster automatiquement les valeurs de ses capteurs sur divers services de cloud tels que Emoncms, Valarm, etc. Mais surtout, l'API callback, qui permet à un serveur PHP, Java ou node.js de prendre le contrôle du hub à distance, est aussi disponible.

Pour configurer les callbacks de votre YoctoHub-GSM-4G, vous devez le brancher par USB sur une machine qui tourne VirtualHub et accéder à sa page de configuration. Dans la section **Outgoing callbacks**, cliquez sur **Edit** à droite de la ligne **Callback URL**.

Les détails des différents types de callbacks sont décrits au chapitre 7.

Cas particulier d'un callback HTTP vers VirtualHub (for Web)

VirtualHub (for Web) fournit un accès distant à vos modules Yoctopuce, à travers Internet. C'est un logiciel qui doit être installé sur un serveur Web, avant naturellement de créer un callback pointant vers lui. L'installation de VirtualHub (for Web) n'étant pas le sujet actuel, vous trouverez procédure d'installation de ce logiciel sur notre blog[1].

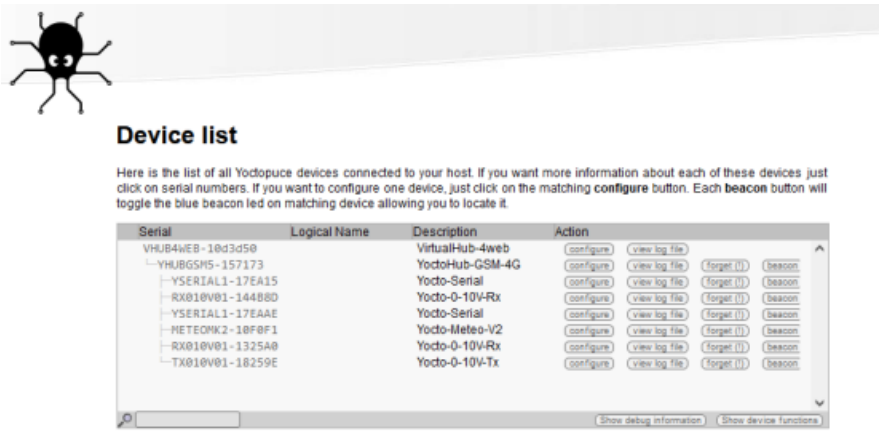
Pour connecter un système Yoctopuce à VirtualHub (for Web), il suffit de configurer sur le YoctoHub-GSM-4G qui le contrôle un callback HTTP pointant sur ce serveur Web. Lors de chaque callback, VirtualHub (for Web) se chargera de télécharger automatiquement toutes les informations nécessaires du YoctoHub-GSM-4G, notamment la configuration des modules, les dernières données des capteurs, les messages de logs des modules, et même les fichiers nécessaires à l'interface utilisateur.

Pour configurer des callbacks HTTP sur le YoctoHub-GSM-4G, commencez par vous assurer que le firmware de votre YoctoHub-GSM-4G est au moins à la version 51900, et que les modules qui lui sont connectés sont à la version 50730 au minimum.

Vous pourrez ensuite configurer les *outgoing callbacks* de votre YoctoHub-GSM-4G de la manière suivante:

1. Type de callback: Yocto-API callback
2. Callback URL: *_path_de_l'instance_VirtualHub-4web_*/HTTPCallback
3. Type de sécurité: MD5 signature
4. Password: *le même que vous avez mis sur votre serveur pour les incoming callbacks*

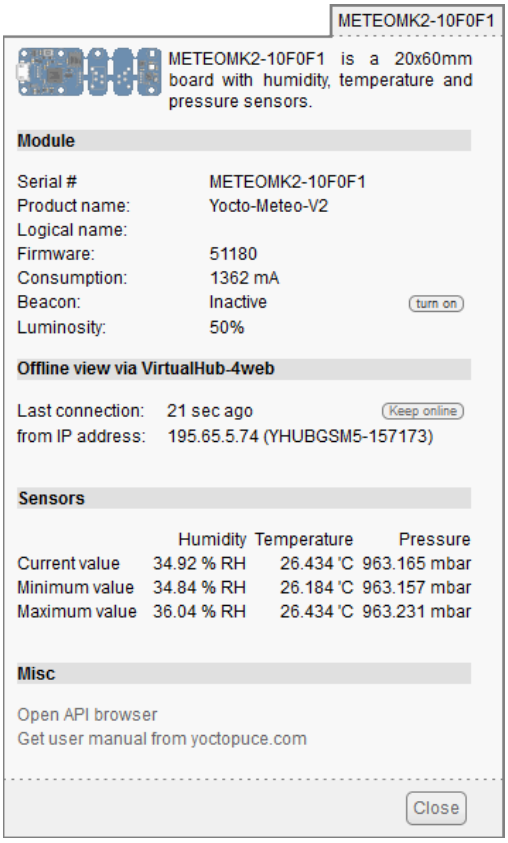
Lorsque vous vous connectez avec un navigateur Web sur le serveur VirtualHub (for Web), vous obtenez une interface très similaire à celle d'un YoctoHub ou d'un VirtualHub habituel.



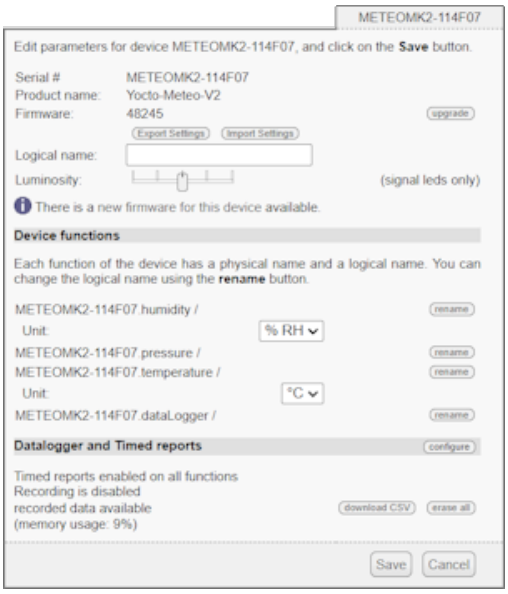
L'interface de VirtualHub (for Web)

[1] https://www.yoctopuce.com/FR/blog.php

Vous pouvez aller consulter l'état de chaque module, tel qu'il était la dernière fois que le module s'est connecté au serveur. Vous pouvez même changer les réglages du module: les changements seront appliqués lors de la prochaine connexion du module au serveur.



La fenêtre des détails du Yocto-Meteo



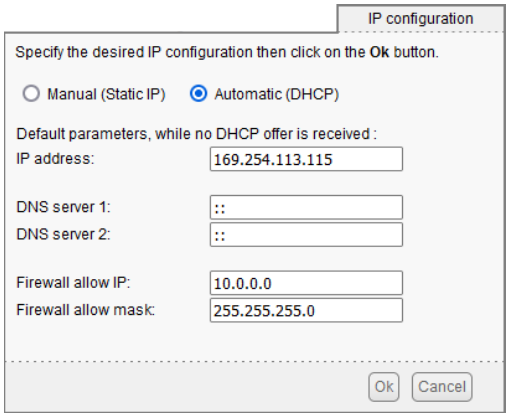
La fenêtre de configuration du Yocto-Meteo

Quand vous utilisez VirtualHub (for Web), n'oubliez pas que l'application des réglages est différée au prochain callback HTTP, ce qui implique forcément un comportement un peu différent d'une connexion directe: par exemple, si vous changez un attribut et que vous relisez directement sa valeur, vous obtiendrez la valeur précédente jusqu'au moment où le réglage sera véritablement appliqué au module et retransmis à VirtualHub (for Web).

5.2. Utilisation avec une SIM permettant un accès VPN

Certains opérateurs cellulaires peuvent fournir sous condition une connection cellulaire avec un lien internet sur une plage d'adresses privées, qui vous est dédiée. Ce type de connexion, dédié aux services *machine-to-machine*, est généralement réservé aux entreprises. Il permet de se connecter à distance (via un réseau privé virtuel) sur votre YoctoHub-GSM-4G, ce qui n'est autrement pas possible car tout téléphone cellulaire est normalement implicitement isolé derrière un filtre NAT.

Dans le cas ou vous disposez d'une telle connection, vous pouvez configurer quelle adresse IP doit être attribuée au YoctoHub-GSM-4G, et quelle adresse IP est autorisée à le contacter (fonction *firewall*). Pour ce faire, cliquez sur le bouton **edit** en face de la ligne **IP adressing**. Il est indispensable de configurer correctement ces paramètres pour pouvoir contacter votre module par son adresse IP, car sinon le firewall bloquera toute connection entrante.

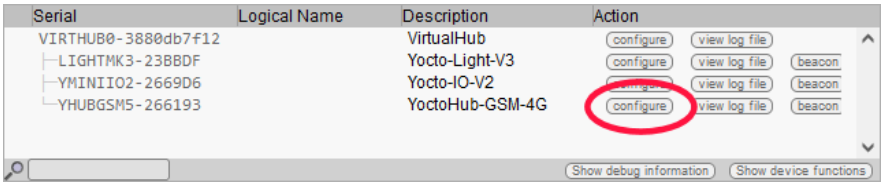


Modification des paramères d'adressage IP pour permettre un accès direct au YoctoHub-GSM-4G

Une fois que ces parametres sont correctement configurés, vous pouvez contacter votre YoctoHub-GSM-4G comme n'importe quel autre YoctoHub.

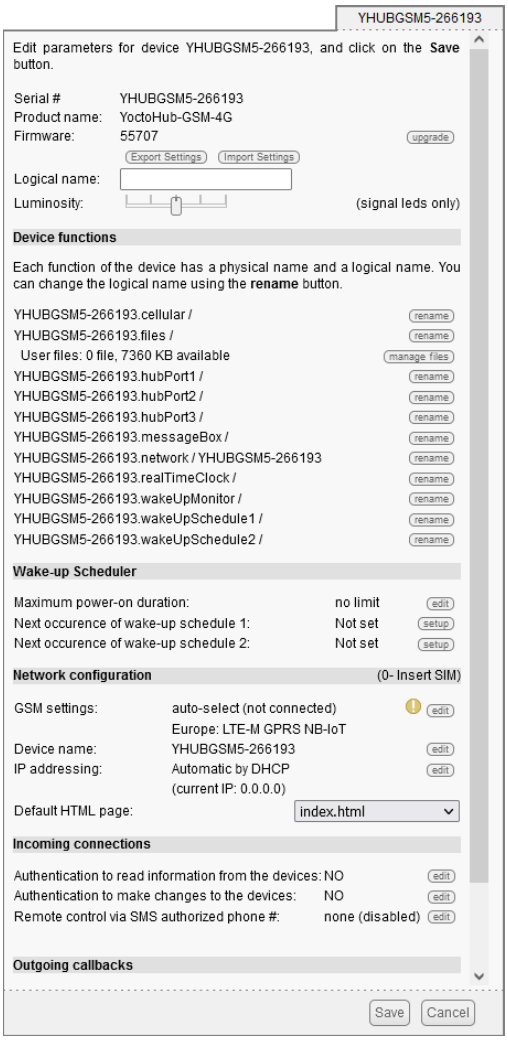
Contrôle d'accès

YoctoHub-GSM-4G 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-GSM-4G 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-GSM-4G.



La fenêtre de configuration de YoctoHub-GSM-4G

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

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-GSM-4G comme ils le souhaitent.

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-GSM-4G. 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.

Influence sur les API
Attention, le contrôle d'accès agira aussi sur les API Yoctopuce qui tenteront de se connecter à YoctoHub-GSM-4G. 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-GSM-4G sous la forme *login:password@adresse:port*, par exemple:

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

5.3. Consommation de données

Une grande part du coût d'une installation de mesures distante basée sur le réseau cellulaire est liée aux frais de communication facturés par l'opérateur. Il est donc important d'estimer correctement la quantité de données qui sera transmise afin de budgéter le système, choisir l'abonnement le plus approprié et faire les bons choix techniques. Voici quelques éléments qui vous aideront pour cette étape.

Pour que cette discussion reste concrète, nous allons prendre en exemple une petit système composé d'un YoctoHub-GSM-4G, assurant la connectivité à trois modules Yoctopuce:

- Un Yocto-PT100 pour mesurer la température
- Un Yocto-4-20mA-Rx pour mesurer deux grandeurs physiques spécifiques
- Un Yocto-GPS pour localiser le système (latitude, longitude, altitude, vitesse)

Au total, il s'agit donc de sept mesures à transmettre.

Si l'on désire surveiller ces 7 mesures chaque minute 24h/24, on pourrait imaginer qu'avec un système idéal cela coûte par exemple 4 octets par mesure, soit grosso-modo 30 octets par minute.

Hélas, Internet n'est pas si efficace que cela pour les petites quantités de données, et il faut y ajouter un bon nombre de données auxiliaires nécessaires au fonctionnement du système.

Les données doivent être au minimum encapsulées dans des paquets réseau TCP/IP afin d'être acheminées au bon serveur, ce qui rajoute au minimum 50 octets à chaque transmission. Le protocole d'échange utilisé par le serveur peut ajouter quelques centaines d'octets par transmission. C'est le cas par exemple avec le protocole HTTP, qui utilise des headers en texte très verbeux. Les données elles-mêmes sont souvent encodées de manière plus explicite qu'une transmission binaire, par exemple en JSON. Cela multiplie facilement la quantité de données à transmettre par 3 ou 4. La vérification de la continuité de la connexion GSM afin de pouvoir poster les données sans retard peut aussi coûter quelques dizaines d'octets par minute, selon le degré de qualité désiré.

Tout cela est un peu vague, mais heureusement, pour préciser les choses, le YoctoHub-GSM-4G dispose d'un compteur d'octets transmis et reçus. Il n'est pas garanti que le chiffre facturé par l'opérateur soit exactement le même puisque des arrondis peuvent s'appliquer, mais au moins cela permet d'avoir une idée plus précise de la consommation de données. Pour relever ce compteur, il vous suffit d'appeler les méthodes:

```
cellular.get_dataSent()
cellular.get_dataReceived()
```

Si votre YoctoHub-GSM-4G est connecté par un câble USB à un PC, vous pouvez donc faire un petit script qui relève ces deux valeurs chaque minute et fait des statistiques. C'est ce que nous avons fait ici, pour documenter la consommation de quelques scénarios classiques.

Consommation de base

Lorsqu'un YoctoHub-GSM-4G est connecté au réseau TCP/IP, le comportement par défaut est de vérifier sa connectivité chaque minute, afin d'initier une nouvelle connexion en cas de problème. Ce comportement coûte 60 octets par minute, en upload et en download. Si nécessaire, il est possible d'espacer les paquets de vérification à l'aide de la méthode

```
cellular.set_pingInterval(n_sec)
```

Par ailleurs, le hub effectue un échange avec un serveur NTP toutes les 10 minutes pour faire un éventuel ajustement de l'horloge en temps réel, utilisée pour l'horodatage des mesures. Cela représente 150 octets par échange, dans les deux sens.

Multiplié par 60 minutes, 24 heures et 31 jours, on se retrouve avec une consommation de 6.7 MB/mois pour un maintient de la connectivité et de l'horodatage 24 heures sur 24, indépendamment du nombre de capteurs connectés. Il est bien entendu possible de réduire proportionnellement ce volume en endormant le hub à l'aide de l'horloge interne pendant certaines heures de la nuit, ou en ne le réveillant que périodiquement.

Envoi de mesures par callback HTTP simple

Si l'on rajoute une transmission minimaliste de la valeur courante de chaque capteur chaque minute par une simple requête HTTP GET avec un encodage CSV, on monte à 0.6KB en upload et 0.5KB en download par minute, soit 50 MB/mois. En choisissant l'alternative plus luxueuse d'une transmission par HTTP POST en formulaire WWW-urlencodé, on arrive même à 70 MB/mois.

Ces chiffres correspondent à notre petit système de test avec sept valeurs mesurées. Si vous avez plus de capteurs, ils grandiront un peu, mais pas proportionnellement puisqu'une partie significative des données transmises est due à la pénalité des protocoles, indépendante du nombre de capteurs.

Connexion par callback HTTP Yocto-API

L'utilisation du protocole de callback Yocto-API coûte plus cher. En effet, ce n'est pas qu'une valeur par capteur qui est transmise chaque minute, mais l'état courant de chaque capteur, y compris les valeurs minimales et maximales rencontrées, l'unité de mesure, la précision, les réglages, l'uptime, etc. Et bien sûr, l'intérêt de ce protocole est qu'il permet aussi d'agir sur les modules pour changer leur configuration à distance, ou actionner un actuateur.

Dans sa version de base, le protocole Yocto-API en http coûte pour notre système 12.1KB/minute en upload, et 0.7KB/minute en download, soit un total de 580 MB/mois. Hereusement, si on active l'optimisation introduite pour la librairie PHP, on peut ramener le trafic à 300 MB/mois.

Connexion par callback WebSocket

L'utilisation d'un canal WebSocket persistant permet d'éviter la répétition des entêtes HTTP à chaque requête. Par contre, le maintien de ce canal ouvert en permanence, avec la publication en continu de toutes les mesures a aussi un coût non négligeable.

Sans autre précaution, la connexion permanente en WebSocket de notre système coûte entre 3 et 5 KB/minute en upload, et entre 2 et 3 KB en download, soit un total de 350 MB/mois environ. Comme chaque nouvelle mesure est transmise instantanément, le volume transmis variera selon la variabilité des mesures. Par contre cela permet de générer autant de callbacks HTTP simples vers des services tiers que désiré, sans trafic GSM supplémentaire.

De même, si l'on désire forwarder un callback WebSocket vers un callback HTTP Yocto-API avec VirtualHub (for web), il faut compter avec le chargement à travers le canal websocket de l'état complet de chaque capteur, ce qui représente une charge supplémentaire. Pour un callback HTTP Yocto-API par minute, on arrive à 500 MB/mois. C'est presque le double de ce que coûte un callback HTTP Yocto-API optimisé fait directement par le hub, mais cela offre l'avantage de permettre de prendre entièrement le contrôle à distance du hub à tout moment.

Heureusement, on peut faire mieux que cela. En effet, les 350 MB/mois utilisés pour transmettre instantanément en permanence la valeur courante de chaque capteur ne sont pas forcément utiles, surtout si on ne veut que traiter les données qu'une fois par minute. Il suffit donc de désactiver sur tous les senseurs l'envoi des valeurs instantanées, et d'utiliser à la place les mesures par intervalle de temps.

```
sensor.muteValueCallbacks()
sensor.set_reportFrequency("1/m")
```

Grâce à l'utilisation des mesures par intervalle de temps, on ramène le traffic à 1.3KB/min en download et 0.9KB/min en upload, soit 100 MB/mois seulement avec une connexion WebSocket permanente. De plus, on reçoit pour chaque intervalle de temps non seulement la valeur moyenne, mais aussi les valeurs min/max sur l'intervalle. Et en cas d'interruption temporaire de connectivité GSM, si vous avez activé l'enregistreur de données sur les modules, vous pourrez retrouver précisément les mesures manquantes, puisqu'elles seront enregistrées simultanément et avec la même granularité.

Cette solution exige toutefois que vous codiez votre solution en exploitant la librairie Yoctopuce et son API callback WebSocket. Si vous cherchez quelquechose de plus facile à mettre en place, en utilisant VirtualHub (for web) et un callback HTTP simple vers un service tiers comme EmonCMS, vous pouvez aussi simplement reconfigurer les capteurs pour propager la valeur moyenne par minute plutôt que la valeur instantanée.

```
sensor.set_advMode(YSensor.ADVMODE_PERIOD_AVG);
```

Cela vous permettra probablement de bénéficier de la possibilité d'une connection à distance sur votre hub, tout en consommant moins de 100 MB/mois.

Planifiez soigneusement votre consommation de données

Les différences de consommation de donnée GSM entre les différentes options sont vraiment significatives. Cela vaut donc la peine de les prendre en compte dès la conception de la solution logicielle que vous mettrez en place pour lire vos capteurs.

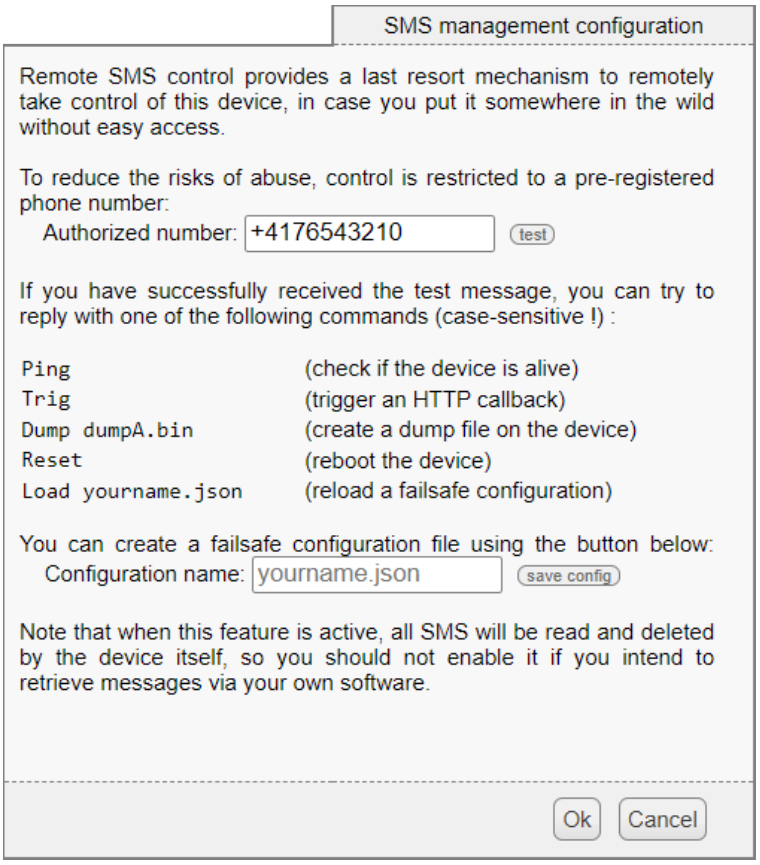
N'oubliez pas non plus que vous pouvez aussi largement réduire la consommation en espaçant les mesures et/ou en déconnectant le hub du réseau par une mise en sommeil.

5.4. Accès par SMS

Le YoctoHub-GSM-4G peut également recevoir des SMS de contrôle. Ainsi, s'il ne parvient plus à se connecter à votre serveur mais qu'il capte encore un réseau GSM, vous avez une chance de le récupérer.

Sécurité

Par mesure de sécurité, seuls les SMS provenant d'un numéro préalablement configuré sont pris en compte. Les autres ne produisent qu'un message de log et sont automatiquement effacés. Ce réglage se trouve dans la section **Incoming connections** de la configuration, en cliquant sur le bouton **edit**:



L'interface de configuration du contrôle par SMS

Cette fenêtre contient un bouton permettant d'envoyer un message de test au numéro choisi, ce qui est doublement intéressant. D'une part, cela vous permettra de vérifier que la carte SIM supporte l'envoi de SMS - ce n'est pas toujours le cas avec les forfaits IoT. D'autre part, c'est la manière la plus simple de connaître le numéro de téléphone de votre YoctoHub-GSM-4G, car le YoctoHub-GSM-4G lui-même n'a aucun moyen de vous en informer: dans la grande majorité des cas le numéro n'est pas stocké sur la carte SIM, et le standard 4G n'a pas prévu de commande pour le demander à l'opérateur.

Grâce à ce message de test, vous pourrez donc simplement répondre au message sur votre téléphone avec une des commandes ci-dessous pour vérifier que votre YoctoHub-GSM-4G reçoit bien vos messages.

Messages reconnus

Le contrôle par SMS permet uniquement quelques opérations importantes suivantes. Voici les commandes reconnues:

Ping demande simplement au hub de vous confirmer par SMS la bonne réception du message. Si le YoctoHub-GSM-4G est configuré pour se réveiller périodiquement, il vous répondra au prochain réveil - les SMS ne sont pas reçus lors du sommeil profond. Vous pourrez donc savoir si le YoctoHub-GSM-4G est encore vivant et capable de se connecter au réseau cellulaire.

Trig demande au YoctoHub-GSM-4G de déclencher un callback HTTP dès que possible. Dans le cas où vous auriez configuré un intervalle entre les callbacks trop grand pour permettre les callbacks, ou programmé une mise en sommeil trop rapide pour permettre au YoctoHub-GSM-4G de faire son callback, cela peut vous permettre de récupérer le contrôle de votre YoctoHub-GSM-4G.

Dump xxx.bin demande au YoctoHub-GSM-4G faire un *dump* de son état actuel dans le fichier `xxx.bin` sur la mémoire flash, pour investigation ultérieure. Dans les cas où le YoctoHub-GSM-4G ne parvient plus à établir la connectivité IP et que vous devez le redémarrer ou même intervenir sur place, ce dump permettra éventuellement au support Yoctopuce de diagnostiquer la cause du problème.

Reset provoque un redémarrage du YoctoHub-GSM-4G.

Load nomfichier.json demande au YoctoHub-GSM-4G de recharger d'un seul coup toute sa configuration depuis le fichier local `nomfichier.json`. Pour utiliser cette fonction, vous devez naturellement avoir préalablement créé ce fichier de sauvegarde de configuration sur le YoctoHub-GSM-4G, avec le bouton disponible sur cette même fenêtre, ou avec la fonction *get_allSettings* de la librairie Yoctopuce par exemple. Vous pouvez donc basculer votre YoctoHub-GSM-4G sur une *configuration de secours* prédéfinie en cas de problème.

Notez que si vous êtes amené à devoir changer à distance la configuration des callbacks ou la configuration cellulaire de votre YoctoHub-GSM-4G, la manière la plus sûre est de

1. prédéfinir la nouvelle configuration sur un autre YoctoHub-GSM-4G, de manière à pouvoir tester préalablement
2. puis d'uploader le fichier de configuration sur le YoctoHub-GSM-4G distant par le gestionnaire de fichier (fonction *manage files* sur VirtualHub for Web)
3. et enfin d'activer la nouvelle configuration d'un coup par la commande SMS **Load maconfig.json**

En effet, si vous essayez de changer ce genre de paramètres directement via l'interface de VirtualHub (for Web), le YoctoHub-GSM-4G risque de couper le callback dès le premier changement de réglage, et de ne pas terminer la reconfiguration, L'utilisation de la commande **Load** garantit une application de tous les réglages d'un seul coup.

SMS et portails IoT

Comme mentionné précédemment, certaines SIM IoT n'offrent pas un accès direct des SMS vers les réseaux cellulaires publiques. Il peut néanmoins être possible d'envoyer et de recevoir des SMS par l'utilisation du portail Web du fournisseur de la SIM IoT, ce qui offre au final les mêmes possibilités d'utilisation du contrôle par SMS. Renseignez-vous après de l'opérateur IoT pour les détails spécifiques à votre SIM.

Si vous n'êtes pas sûr du numéro d'expéditeur utilisé par le portail que vous devez autoriser dans l'interface de configuration de votre YoctoHub-GSM-4G, faites simplement un essai en autorisant un numéro "au hasard". Lorsque le portail enverra votre premier SMS *Ping*, vous verrez dans les logs du YoctoHub le numéro de téléphone à l'origine du message "rejeté".

6. Envoi de données vers l'extérieur

YoctoHub-GSM-4G est capable de se connecter à des services externes pour communiquer l'état des modules qui lui sont raccordés.

YoctoHub-GSM-4G sait comment poster ses données au format accepté par quelques services Cloud tiers, tels que

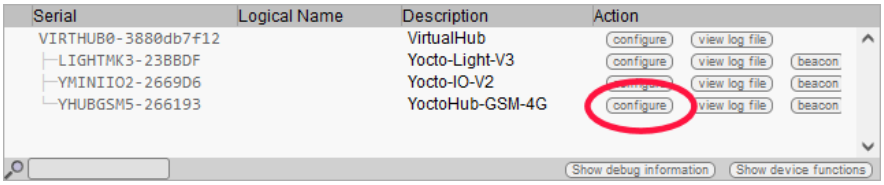
- Emoncms
- InfluxDB (versions 1.0 et 2.0)
- PRTG
- Valarm.net

YoctoHub-GSM-4G 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:

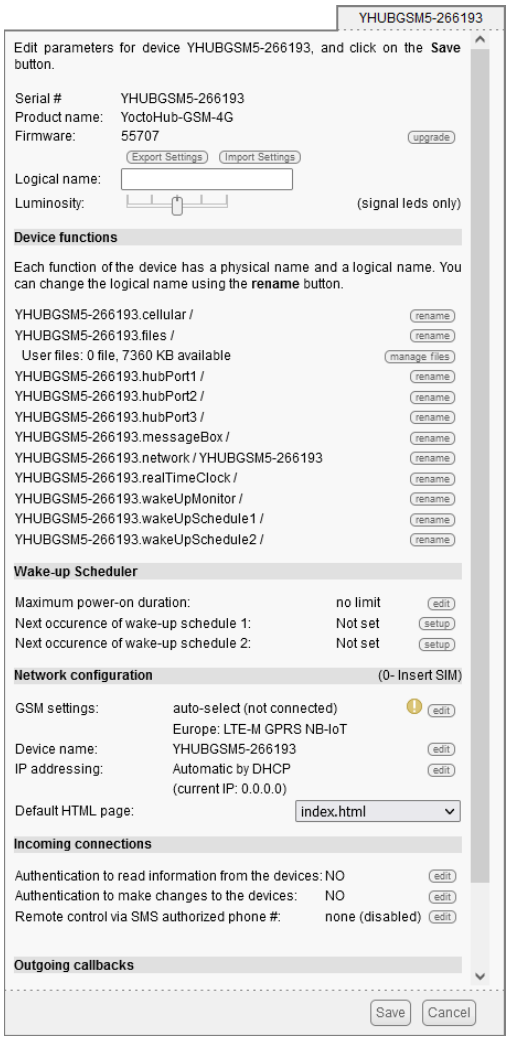
- MQTT
- Yocto-API

6.1. Configuration

Pour utiliser cette fonctionnalité, cliquez simplement sur le bouton **configure** de la ligne correspondant à YoctoHub-GSM-4G 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-GSM-4G peut interagir avec un serveur web externe. Vous avez plusieurs type d'interactions à votre disposition.

6.2. Callbacks HTTP vers des services tiers

YoctoHub-GSM-4G 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-GSM-4G.

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-GSM-4G.

6.3. 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-GSM-4G 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-GSM-4G (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.

6.4. 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-GSM-4G 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.

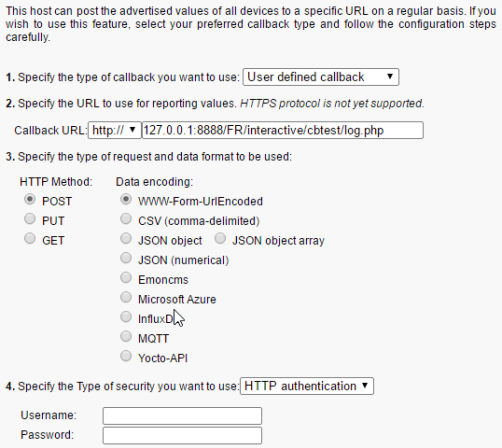
L'utilisation de WebSockets permet d'accéder à plusieurs fonctionnalités avancées des librairies Yoctopuce qui ne sont pas disponibles via un callback HTTP:

- 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.

6.5. 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-GSM-4G 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-GSM-4G 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.

6.6. 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-GSM-4G (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-GSM-4G" }
```

Fichier de personnalisation pour Emoncms

Le format de base utilisé par YoctoHub-GSM-4G 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-GSM-4G 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-GSM-4G 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-GSM-4G 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-GSM-4G 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-GSM-4G 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-GSM-4G 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"}
]}}
```

6.7. Planification des callbacks

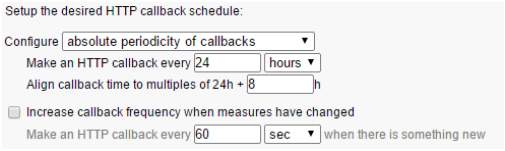
La section de plannification des callbacks est présente pour tous les types de callbacks. C'est la dernière section contenant des champs à remplir.

Un premier callback est toujours effectué quelques secondes après le lancement de YoctoHub-GSM-4G. 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-GSM-4G 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

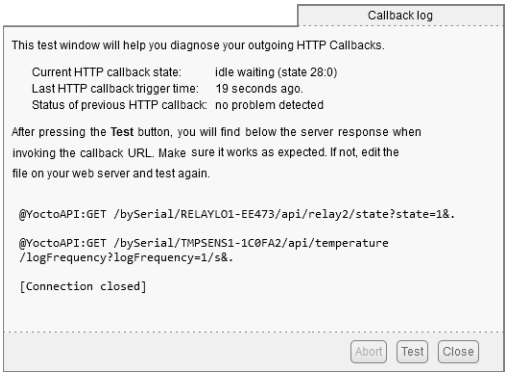
Vous pouvez choisir explicitement si vous désirez que la fréquence des callbacks varie lorsqu'aucune nouvelle mesure n'est détectée. Cela permet de choisir la fréquence minimale de transmission pour réduire la quantité de données transmises sur le réseau si rien ne se passe.

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.

6.8. Tests

Afin de vous permettre de débugger le processus, YoctoHub-GSM-4G 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**.

6.9. Connexions spontanées

En plus de connexions liées aux callbacks définis décrites dans les sections précédentes, YoctoHub-GSM-4G va occasionnellement tenter d'établir des connexions vers l'extérieur. Ces connexions sont les suivantes

- Connexion DNS: à chaque fois que le YoctoHub-GSM-4G 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.
- Installation de Yocto-Visualization (for web): Lorsque l'utilisateur déclenche l'installation de Yocto-Visualization (for web) depuis l'interface du YoctoHub-GSM-4G, 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-GSM-4G 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-GSM-4G 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-GSM-4G. 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
Résolution de nom	DNS UDP 53	serveur DNS configuré	configurer l'adresse IP du serveur NTP et définir les callbacks par adresse IP

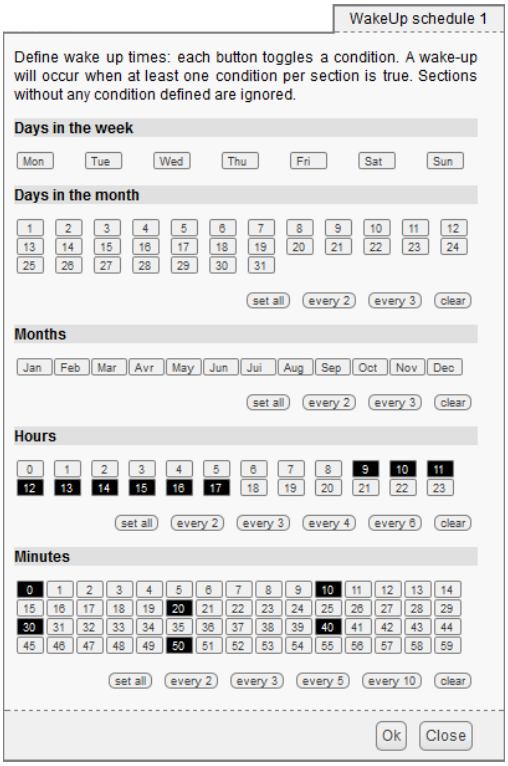
Mise à jour de l'heure	NTP UDP 123	1.yoctopuce.pool.ntp.org ou serveur configuré	régler le serveur NTP à 255.255.255.255
Installation de YV4web	HTTP TCP 80	www.yoctopuce.com	ne pas utiliser le lien rapide sur l'interface Web
Test de version	HTTPS TCP 443	www.yoctopuce.com	ne pas utiliser l'interface Web du YoctoHub-GSM-4G
Téléchargement firmware	HTTPS TCP 443	www.yoctopuce.com	ne pas faire la mise à jour par l'interface Web

7. Mise en sommeil

Le YoctoHub-GSM-4G dispose d'une horloge en temp réel (RTC) alimentée par un super condensateur, qui se recharge automatiquement lorsque le module est sous tension mais permet de maintenir l'heure sans aucune alimentation pendant plusieurs jours. Ce RTC est utilisé pour piloter un système de mise en sommeil afin d'économiser l'énergie. Le système de mise en sommeil peut être configuré manuellement à l'aide d'une interface, ou piloté par logiciel.

7.1. Configuration manuelle du système de réveil

Les conditions de réveil peuvent être configurées manuellement en vous connectant sur l'interface du YoctoHub-GSM-4G. Dans la section **Wake-up scheduler** de la fenêtre de configuration générale, cliquez sur le bouton setup correspondant à l'un des "wakeup-schedule". Une fenêtre qui permet d'agender des réveils plus ou moins réguliers s'ouvre. Il suffit de sélectionner les cases correspondant aux occurrences désirées. Les sections laissées vides seront ignorée.



Fenêtre de configuration des réveils, ici toutes les 10 minutes entre 9h et 17h

De même, vous pouvez configurer directement sur l'interface du YoctoHub-GSM-4G le temps d'éveil maximal désiré, après lequel le module retournera automatiquement en sommeil profond. Si vous utilisez votre YoctoHub-GSM-4G sur batteries, ceci vous assurera de ne pas vider les batteries même si aucun ordre de mise en sommeil explicite n'est reçu.

7.2. Paramétrage du système de réveil par logiciel

Au niveau de l'interface de programmation, le système de réveil est implémenté a l'aide de deux types de fonction : La fonction *wakeUpMonitor* et la fonction *WakeUpSchedule*.

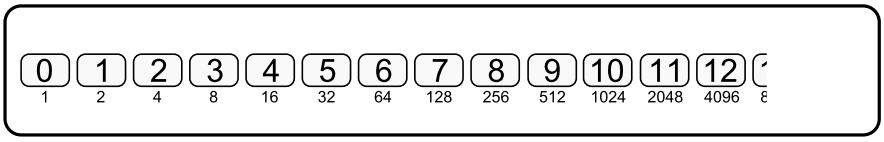
wakeUpMonitor

La fonction *wakeUpMonitor* gère l'éveil et la mise en sommeil proprement dits. Elle met à disposition toutes les fonctionnalités de contrôle immédiate: éveil immédiat, mise en sommeil immédiate, calcul de la date du prochain réveil etc... Le fonction *wakeUpMonitor* permet aussi de définir le temps maximum pendant lequel le YoctoHub-GSM-4G peut rester éveil avant de de se mettre automatiquement en sommeil.

wakeUpSchedule

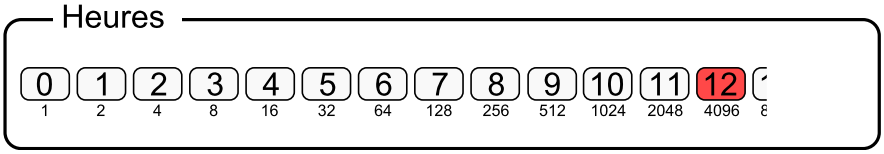
La fonction *wakeUpSchedule* permet de programmer une condition de réveil. Elle dispose de cinq variables qui permet de définir des correspondance sur les minutes, heure, jour de la semaine, jour dans le mois, et mois. Ces variables sont des variables entières dont chaque bit défini une correspondance. Schématiquement, chaque ensemble de minutes, jours, heures est représenté sous la forme d'un ensemble de case avec chacune un coefficient qui est une puissante de deux, exactement comme dans l'interface correspondante du YoctoHub-GSM-4G.

Par exemple le bit 0 des heures correspond à l'heure zéro, le bit 1 correspond à l'heure une, le bit 2 correspond à l'heure 2 etc.



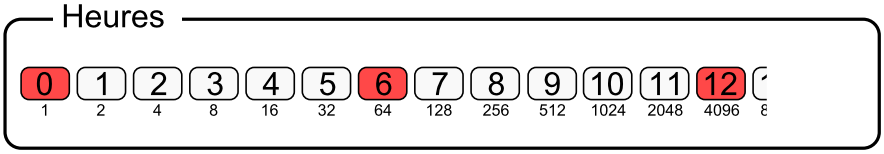
Chaque case se voit affecter une puissante de deux

Ainsi pour programmer le YoctoHub-GSM-4G pour qu'il se réveille tout les jours a midi, il mettre le bit 12 à 1, ce qui correspond à la valeur 2^12 = 4096.



Exemple de réveil à 12 H

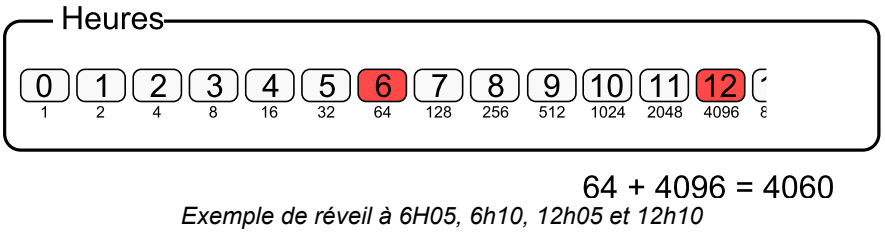
Pour que le module se reveille à 0 heure, 6 heures et 12 heures, il faut mettre les bit 0,6,et 12 à un, ce qui correspondant à la valeur 2^0+ 2^6 + 2^12 = 1 + 64 + 4096 = 4161



1 + 64 + 4096 = 4151

Exemple de réveil à 0, 6 et 12 H

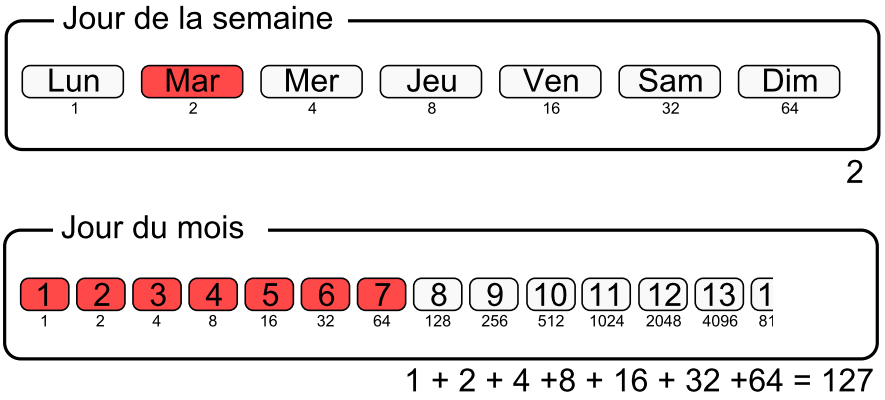
Les variables peuvent être combinées, pour qu'un réveil ait lieu tous les jours à 6H05, 6h10, 12h05 et 12h10 il suffit de mettre les heures à 2^6 + 2^12 = 4060 et les minutes à 2^5 + 2^10= 1056. Les variables laissée à zéro sont ignorées.



Exemple de réveil à 6H05, 6h10, 12h05 et 12h10

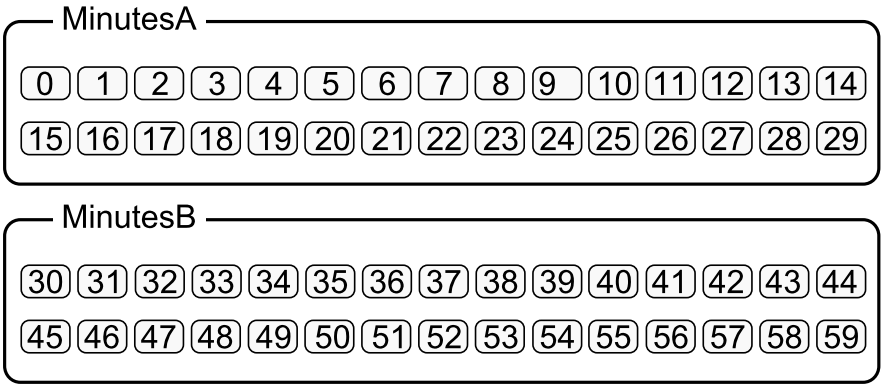
Notez que si vous désirez programmer un réveil à 6H05 et 12h10, mais pas 6h10 et 12h10, vous aurez besoin d'utiliser deux fonctions *wakeUpSchedule* différentes.

Ce paradigme permet de programmer des réveils assez complexe. Ainsi pour programmer un réveil tous les premiers mardis du mois, faut mettre à un le deuxième bit des jours de la semaine et les sept premiers bit des jours du mois.



Exemple de réveil tous les premiers mardi du mois

Certains languages de programmation, dont Javascript et Python, ne supportent pas les entiers 64 bits, ce qui pose un problème pour encoder les minutes. C'est pourquoi les minutes sont à la fois accessibles via un entier 64 bits *minutes* et deux entiers 32 bits, *minutesA* et *minutesB*, qui eux sont disponibles dans tous les languages actuels.



Les minutes sont aussi disponibles sous forme de deux entiers 32 bits.

La fonction *wakeUpSchedule* dispose d'une variable supplémentaire qui permet de définir le temps, en secondes, durant lequel le module restera éveillé après un réveil. Si cette variable est mise à zéro, le module restera éveillé.

Le YoctoHub-GSM-4G dispose de deux fonctions *wakeUpSchedule* ce qui permet de programmer jusqu'a deux types de réveils indépendants.

8. Programmation

8.1. Accès aux modules connectés

Sans VPN, on ne peut accéder par programmation aux fonctions du YoctoHub-GSM-4G que via le port USB, ou par l'API par callback HTTP en utilisant par exemple en PHP:

```
YAPI::RegisterHub("callback",errmsg);
```

Dans les rares cas où vous pouvez accéder à votre YoctoHub-GSM-4G à travers son adresse IP, il 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-GSM-4G se situe au niveau de l'appel à RegisterHub. Pour utiliser des modules connectés à un YoctoHub-GSM-4G, vous devez utilisez l'adresse IP du YoctoHub-GSM-4G lors de l'appel à RegisterHub. Ce qui donne par exemples en PHP:

```
YAPI::RegisterHub("169.254.214.162",errmsg); // l'adresse IP du hub est
169.254.214.162
```

8.2. Contrôle du YoctoHub-GSM-4G

Du point de vue API de programmation, le YoctoHub-GSM-4G est un module comme les autres quand il est connecté par USB. 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-GSM-4G, etc.

Cellular

Cette classe permet de contrôler la configuration du réseau cellulaire du YoctoHub-GSM-4G, en particulier le choix de la technologie radio, le nom de l'opérateur cellulaire, le code PIN pour utiliser la carte SIM et les paramètres de l'APN.

Network

Cette classe permet de contrôler la partie réseau du YoctoHub-GSM-4G, vous pourrez contrôler l'état du link, lire l'adresse MAC, changer l'adresse IP du YoctoHub-GSM-4G, 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-GSM-4G, 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-GSM-4G. Le YoctoHub-GSM-4G 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-GSM-4G.

WakeUpMonitor

Cette classe permet de contrôler la mise en sommeil du YoctoHub-GSM-4G.

WakeUpSchedule

Cette classe permet d'agender un ou plusieurs réveils du YoctoHub-GSM-4G.

Vous trouverez quelques exemples de contrôle du YoctoHub-GSM-4G par programmation dans les librairies Yoctopuce, disponibles gratuitement sur le site de Yoctopuce.

9. Référence de l'API de haut niveau

Ce chapitre résume les fonctions de l'API de haut niveau pour commander votre YoctoHub-GSM-4G. 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-GSM-4G.

9.1. 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';** **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>** **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

9.2. La classe YCellular

Interface pour intéragir avec les interfaces réseau cellulaire, disponibles par exemple dans le YoctoHub-GSM-2G, le YoctoHub-GSM-3G-EU, le YoctoHub-GSM-3G-NA et le YoctoHub-GSM-4G

La classe `YCellular` permet de configurer et de contrôler la configuration du réseau cellulaire sur les modules Yoctopuce qui en sont dotés. Notez que les paramètres TCP/IP sont configurés séparément, à l'aide de la classe `YNetwork`.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

YCellular.FindCellular(func)

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

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

YCellular.FindCellularInContext(yctx, func)

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

java uwp ts es

YCellular.FirstCellular()

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

cpp m pas vb cs java uwp py php ts es

YCellular.FirstCellularInContext(yctx)

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

java uwp ts es

YCellular.GetSimilarFunctions()

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

dnp

Propriétés des objets `YCellularProxy`

cellular→AdvertisedValue *[lecture seule]*

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

dnp

cellular→Apn *[modifiable]*

Nom du point d'accès (APN) à utiliser, si nécessaire.

dnp

cellular→CellOperator *[lecture seule]*

Nom de l'opérateur de réseau cellulaire actuellement utilisé.

dnp

cellular→EnableData *[modifiable]*

Condition dans laquelle le service de données IP (GRPS) doit être activé.

dnp

cellular→FriendlyName *[lecture seule]*

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

dnp

cellular→FunctionId *[lecture seule]*

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

dnp

cellular→HardwareId *[lecture seule]*

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

dnp

cellular→IsOnline *[lecture seule]*

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

dnp

cellular→LinkQuality *[lecture seule]*

Qualité de la connection, exprimée en pourcents.

dnp

cellular→LockedOperator *[modifiable]*

Nom de l'opérateur de réseau cellulaire à utiliser exclusivement, si le choix automatique est désactivé, ou une chaîne vide si la carte SIM sélectionne automatiquement l'opérateur selon ceux disponibles.

dnp

cellular→LogicalName *[modifiable]*

Nom logique de la fonction.

dnp

cellular→Pin *[modifiable]*

String opaque si un code PIN a été configuré dans le module pour accéder à la carte SIM, ou une chaîne vide il n'a pas été configuré ou si la SIM a rejeté le code indiqué.

dnp

cellular→PingInterval *[modifiable]*

Intervalle entre les tests de connectivité spontanés, en secondes.

dnp

cellular→RadioConfig *[modifiable]*

Type de protocole utilisé sur la communication série, sous forme d'une chaîne de caractères.

dnp

cellular→SerialNumber *[lecture seule]*

Numéro de série du module, préprogrammé en usine.

dnp

Méthodes des objets YCellular

cellular→_AT(cmd)

Envoie une commande AT au module GSM, et retourne le résultat.

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

cellular→clearCache()

Invalide le cache.

cpp m pas vb cs java py php ts es

cellular→clearDataCounters()

Réinitialise les compteurs de données transmises et reçues.

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

cellular→decodePLMN(mccmnc)

Retourne le nom usuel de l'opérateur cellulaire pour une paire MCC/MNC donnée.

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

cellular→describe()

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

cpp m pas vb cs java py php ts es

cellular→get_advertisedValue()

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

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

cellular→get_airplaneMode()

Retourne vrai si le mode avion est activé (radio désactivée).

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

cellular→get_apn()

Retourne le nom du point d'accès (APN) à utiliser, si nécessaire.

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

cellular→get_apnSecret()

Retourne une string opaque si des paramètres d'identification sur l'APN ont été configurés dans le module, ou une chaîne vide autrement.

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

cellular→get_availableOperators()

Retourne la liste des opérateurs GSM disponibles à proximité.

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

cellular→get_cellIdentifier()

Retourne l'identifiant unique de la station de base utilisée: MCC, MNC, LAC et Cell ID.

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

cellular→get_cellOperator()

Retourne le nom de l'opérateur de réseau cellulaire actuellement utilisé.

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

cellular→get_cellType()

Type de connection cellulaire active.

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

cellular→get_communicationProfiles()

Retourne la liste des profiles de communication radio, sous forme d'un tableau de strings (YoctoHub-GSM-4G seulement).

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

cellular→get_dataReceived()

Retourne le nombre d'octets reçus jusqu'à présent.

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

cellular→get_dataSent()

Retourne le nombre d'octets envoyés jusqu'à présent.

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

cellular→get_enableData()

Retourne la condition dans laquelle le service de données IP (GRPS) doit être activé.

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

cellular→get_errorMessage()

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

cpp m pas vb cs java py php ts es

cellular→get_errorType()

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

cpp m pas vb cs java py php ts es

cellular→get_friendlyName()

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

cpp m cs java py php ts es dnp

cellular→get_functionDescriptor()

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

cpp m pas vb cs java py php ts es

cellular→get_functionId()

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

cpp m vb cs java py php ts es dnp

cellular→get_hardwareId()

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

cpp m vb cs java py php ts es dnp

cellular→get_imsi()

Retourne le "International Mobile Subscriber Identity" (MSI) qui identifie de manière unique la carte SIM.

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

cellular→get_linkQuality()

Retourne la qualité de la connection, exprimée en pourcents.

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

cellular→get_lockedOperator()

Retourne le nom de l'opérateur de réseau cellulaire à utiliser exclusivement, si le choix automatique est désactivé, ou une chaîne vide si la carte SIM sélectionne automatiquement l'opérateur selon ceux disponibles.

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

cellular→get_logicalName()

Retourne le nom logique de l'interface cellulaire.

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

cellular→get_message()

Retourne le dernier message de diagnostique de l'interface au réseau sans fil.

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

cellular→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

cellular→get_module_async(callback, context)

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

cellular→get_pin()

Retourne une string opaque si un code PIN a été configuré dans le module pour accéder à la carte SIM, ou une chaîne vide il n'a pas été configuré ou si la SIM a rejeté le code indiqué.

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

cellular→get_pingInterval()

Retourne l'intervalle entre les tests de connectivité spontanés, en secondes.

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

cellular→get_radioConfig()

Retourne le type de protocole utilisé sur la communication série, sous forme d'une chaîne de caractères.

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

cellular→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

cellular→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

cellular→isOnline()

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

cpp m pas vb cs java py php ts es dnp

cellular→isOnline_async(callback, context)

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

cellular→isReadOnly()

Indique si la fonction est en lecture seule.

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

cellular→load(msValidity)

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

cpp m pas vb cs java py php ts es

cellular→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

cellular→load_async(msValidity, callback, context)

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

cellular→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

cellular→nextCellular()

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

cpp m pas vb cs java uwp py php ts es

cellular→quickCellSurvey()

Retourne la liste d'identifiants pour les antennes GSM à proximité, telle que requise pour géolocaliser rapidement le module.

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

cellular→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

cellular→sendPUK(puk, newPin)

Envoie le code PUK à la carte SIM pour la débloquer après trois échecs consécutifs de code PIN, et établit un nouveau code PIN dans la SIM.

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

cellular→set_airplaneMode(newval)

Modifie l'état du mode avion (radio désactivée).

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

cellular→set_apn(newval)

Retourne le nom du point d'accès (APN) à utiliser, si nécessaire.

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

cellular→set_apnAuth(username, password)

Configure les paramètres d'identification pour se connecter à l'APN.

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

cellular→set_dataReceived(newval)

Modifie la valeur du compteur d'octets reçus.

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

cellular→set_dataSent(newval)

Modifie la valeur du compteur d'octets envoyés.

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

cellular→set_enableData(newval)

Modifie la condition dans laquelle le service de données IP (GRPS) doit être activé.

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

cellular→set_lockedOperator(newval)

Modifie le nom de l'opérateur de réseau cellulaire à utiliser.

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

cellular→set_logicalName(newval)

Modifie le nom logique de l'interface cellulaire.

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

cellular→set_pin(newval)

Modifie le code PIN utilisé par le module pour accéder à la carte SIM.

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

cellular→set_pingInterval(newval)

Modifie l'intervalle entre les tests de connectivité spontanés, en secondes.

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

cellular→set_radioConfig(newval)

Modifie le type de protocol utilisé sur la communication série.

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

cellular→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

cellular→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

cellular→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

9.3. 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>** **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';** **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

9.4. 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';** **in Node.js: import { YHub } from 'yoctolib-cjs/yocto_module.js';**
es	**in HTML: <script src="../../lib/yocto_module.js"></script>** **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

9.5. 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>** **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';** **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

9.6. 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';** **in Node.js: import { YFiles } from 'yoctolib-cjs/yocto_files.js';**
es	**in HTML: <script src="../../lib/yocto_files.js"></script>** **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

9.7. La classe YRealTimeClock

Interface pour intéragir avec les horloges à temps réel, disponibles par exemple dans le YoctoHub-GSM-4G, le YoctoHub-Wireless-SR, le YoctoHub-Wireless-g et le YoctoHub-Wireless-n

La classe `YRealTimeClock` permet d'accéder à l'horloge embarquée sur certains modules Yoctopuce. Elle fournit la date et l'heure courante de manière persistante, même en cas de coupure de courant de plusieurs jours. Elle est le fondement des fonctions de réveil automatique implémentées par le WakeUpScheduler. L'heure courante peut représenter aussi bien une heure locale qu'une heure UTC, mais aucune adaptation automatique n'est fait au changement d'heure été/hiver.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

YRealTimeClock.FindRealTimeClock(func)

Permet de retrouver une horloge à temps réel d'après un identifiant donné.

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

YRealTimeClock.FindRealTimeClockInContext(yctx, func)

Permet de retrouver une horloge à temps réel d'après un identifiant donné dans un Context YAPI.

java uwp ts es

YRealTimeClock.FirstRealTimeClock()

Commence l'énumération des horloges à temps réel accessibles par la librairie.

cpp m pas vb cs java uwp py php ts es

YRealTimeClock.FirstRealTimeClockInContext(yctx)

Commence l'énumération des horloges à temps réel accessibles par la librairie.

java uwp ts es

YRealTimeClock.GetSimilarFunctions()

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

dnp

Propriétés des objets YRealTimeClockProxy

realtimeclock→AdvertisedValue *[lecture seule]*

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

dnp

realtimeclock→DisableHostSync *[modifiable]*

Vrai si la synchronisation transparente de l'horloge avec l'hôte est désactivée, sinon faux.

dnp

realtimeclock→FriendlyName *[lecture seule]*

Identifiant global de la fonction au format NOM_MODULE.NOM_FONCTION.

dnp

realtimeclock→FunctionId *[lecture seule]*

Identifiant matériel de l'horloge à temps réel, sans référence au module.

dnp

realtimeclock→HardwareId *[lecture seule]*

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

dnp

realtimeclock→IsOnline *[lecture seule]*

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

dnp

realtimeclock→LogicalName *[modifiable]*

Nom logique de la fonction.

dnp

realtimeclock→SerialNumber *[lecture seule]*

Numéro de série du module, préprogrammé en usine.

dnp

realtimeclock→UtcOffset *[modifiable]*

Nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone).

dnp

Méthodes des objets YRealTimeClock

realtimeclock→clearCache()

Invalide le cache.

cpp m pas vb cs java py php ts es

realtimeclock→describe()

Retourne un court texte décrivant de manière non-ambigüe l'instance de l'horloge à temps réel au format TYPE(NAME)=SERIAL.FUNCTIONID.

cpp m pas vb cs java py php ts es

realtimeclock→get_advertisedValue()

Retourne la valeur courante de l'horloge à temps réel (pas plus de 6 caractères).

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

realtimeclock→get_dateTime()

Retourne l'heure courante au format "AAAA/MM/JJ hh:mm:ss".

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

realtimeclock→get_disableHostSync()

Retourne vrai si la synchronisation transparente de l'horloge avec l'hôte est désactivée, sinon faux.

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

realtimeclock→get_errorMessage()

Retourne le message correspondant à la dernière erreur survenue lors de l'utilisation de l'horloge à temps réel.

cpp m pas vb cs java py php ts es

realtimeclock→get_errorType()

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

cpp m pas vb cs java py php ts es

realtimeclock→get_friendlyName()

Retourne un identifiant global de l'horloge à temps réel au format `NOM_MODULE.NOM_FONCTION`.

cpp m cs java py php ts es dnp

realtimeclock→get_functionDescriptor()

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

cpp m pas vb cs java py php ts es

realtimeclock→get_functionId()

Retourne l'identifiant matériel de l'horloge à temps réel, sans référence au module.

cpp m vb cs java py php ts es dnp

realtimeclock→get_hardwareId()

Retourne l'identifiant matériel unique de l'horloge à temps réel au format `SERIAL.FUNCTIONID`.

cpp m vb cs java py php ts es dnp

realtimeclock→get_logicalName()

Retourne le nom logique de l'horloge à temps réel.

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

realtimeclock→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

realtimeclock→get_module_async(callback, context)

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

realtimeclock→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

realtimeclock→get_timeSet()

Retourne vrai si l'horloge à été mise à l'heure, sinon faux.

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

realtimeclock→get_unixTime()

Retourne l'heure courante au format Unix (nombre de seconds secondes écoulées depuis le 1er janvier 1970).

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

realtimeclock→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

realtimeclock→get_utcOffset()

Retourne le nombre de secondes de décallage entre l'heure courante et l'heure UTC (time zone).

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

realtimeclock→isOnline()

Vérifie si le module hébergeant l'horloge à temps réel est joignable, sans déclencher d'erreur.

cpp m pas vb cs java py php ts es dnp

realtimeclock→isOnline_async(callback, context)

Vérifie si le module hébergeant l'horloge à temps réel est joignable, sans déclencher d'erreur.

realtimeclock→isReadOnly()

Indique si la fonction est en lecture seule.

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

realtimeclock→load(msValidity)

Met en cache les valeurs courantes de l'horloge à temps réel, avec une durée de validité spécifiée.

cpp m pas vb cs java py php ts es

realtimeclock→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

realtimeclock→load_async(msValidity, callback, context)

Met en cache les valeurs courantes de l'horloge à temps réel, avec une durée de validité spécifiée.

realtimeclock→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

realtimeclock→nextRealTimeClock()

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

cpp m pas vb cs java uwp py php ts es

realtimeclock→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

realtimeclock→set_disableHostSync(newval)

Modifie l'état de la synchronisation transparente de l'horloge avec l'hôte.

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

realtimeclock→set_logicalName(newval)

Modifie le nom logique de l'horloge à temps réel.

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

realtimeclock→set_unixTime(newval)

Modifie l'heure courante.

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

realtimeclock→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

realtimeclock→set_utcOffset(newval)

Modifie le nombre de secondes de décalage entre l'heure courante et l'heure UTC (time zone).

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

realtimeclock→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

realtimeclock→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

9.8. La classe YWakeUpMonitor

Interface pour intéragir avec les moniteurs de réveil, disponibles par exemple dans le YoctoHub-GSM-4G, le YoctoHub-Wireless-SR, le YoctoHub-Wireless-g et le YoctoHub-Wireless-n

La classe `YWakeUpMonitor` prend en charge le contrôle global de toutes les sources de réveil possibles ainsi que les mises en sommeil automatiques.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

YWakeUpMonitor.FindWakeUpMonitor(func)

Permet de retrouver un moniteur de réveil d'après un identifiant donné.

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

YWakeUpMonitor.FindWakeUpMonitorInContext(yctx, func)

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

java uwp ts es

YWakeUpMonitor.FirstWakeUpMonitor()

Commence l'énumération des moniteurs de réveil accessibles par la librairie.

cpp m pas vb cs java uwp py php ts es

YWakeUpMonitor.FirstWakeUpMonitorInContext(yctx)

Commence l'énumération des moniteurs de réveil accessibles par la librairie.

java uwp ts es

YWakeUpMonitor.GetSimilarFunctions()

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

dnp

Propriétés des objets YWakeUpMonitorProxy

wakeupmonitor→AdvertisedValue *[lecture seule]*

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

dnp

wakeupmonitor→FriendlyName *[lecture seule]*

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

dnp

wakeupmonitor→FunctionId *[lecture seule]*

Identifiant matériel du moniteur de réveil, sans référence au module.

dnp

wakeupmonitor→HardwareId *[lecture seule]*

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

dnp

wakeupmonitor→IsOnline *[lecture seule]*

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

dnp

wakeupmonitor→LogicalName *[modifiable]*

Nom logique de la fonction.

dnp

wakeupmonitor→NextWakeUp *[modifiable]*

Prochaine date/heure de réveil agendée (format UNIX).

dnp

wakeupmonitor→PowerDuration *[modifiable]*

Temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

dnp

wakeupmonitor→SerialNumber *[lecture seule]*

Numéro de série du module, préprogrammé en usine.

dnp

Méthodes des objets `YWakeUpMonitor`

wakeupmonitor→clearCache()

Invalide le cache.

cpp m pas vb cs java py php ts es

wakeupmonitor→describe()

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

cpp m pas vb cs java py php ts es

wakeupmonitor→get_advertisedValue()

Retourne la valeur courante du moniteur de réveil (pas plus de 6 caractères).

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

wakeupmonitor→get_errorMessage()

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

cpp m pas vb cs java py php ts es

wakeupmonitor→get_errorType()

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

cpp m pas vb cs java py php ts es

wakeupmonitor→get_friendlyName()

Retourne un identifiant global du moniteur de réveil au format NOM_MODULE.NOM_FONCTION.

cpp m cs java py php ts es dnp

wakeupmonitor→get_functionDescriptor()

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

cpp m pas vb cs java py php ts es

wakeupmonitor→get_functionId()

Retourne l'identifiant matériel du moniteur de réveil, sans référence au module.

cpp m vb cs java py php ts es dnp

wakeupmonitor→get_hardwareId()

Retourne l'identifiant matériel unique du moniteur de réveil au format SERIAL.FUNCTIONID.

cpp m vb cs java py php ts es dnp

wakeupmonitor→get_logicalName()

Retourne le nom logique du moniteur de réveil.

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

wakeupmonitor→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

wakeupmonitor→get_module_async(callback, context)

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

wakeupmonitor→get_nextWakeUp()

Retourne la prochaine date/heure de réveil agendée (format UNIX).

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

wakeupmonitor→get_powerDuration()

Retourne le temp d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

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

wakeupmonitor→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

wakeupmonitor→get_sleepCountdown()

Retourne le temps avant le prochain sommeil.

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

wakeupmonitor→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

wakeupmonitor→get_wakeUpReason()

Renvoie la raison du dernier réveil.

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

wakeupmonitor→get_wakeUpState()

Revoie l'état actuel du moniteur.

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

wakeupmonitor→isOnline()

Vérifie si le module hébergeant le moniteur de réveil est joignable, sans déclencher d'erreur.

cpp m pas vb cs java py php ts es dnp

wakeupmonitor→isOnline_async(callback, context)

Vérifie si le module hébergeant le moniteur de réveil est joignable, sans déclencher d'erreur.

wakeupmonitor→isReadOnly()

Indique si la fonction est en lecture seule.

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

wakeupmonitor→load(msValidity)

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

cpp m pas vb cs java py php ts es

wakeupmonitor→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

wakeupmonitor→load_async(msValidity, callback, context)

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

wakeupmonitor→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

wakeupmonitor→nextWakeUpMonitor()

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

cpp m pas vb cs java uwp py php ts es

wakeupmonitor→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

wakeupmonitor→resetSleepCountDown()

Réinitialise le compteur de mise en sommeil.

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

wakeupmonitor→set_logicalName(newval)

Modifie le nom logique du moniteur de réveil.

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

wakeupmonitor→set_nextWakeUp(newval)

Modifie les jours de la semaine où un réveil doit avoir lieu.

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

wakeupmonitor→set_powerDuration(newval)

Modifie le temps d'éveil maximal en secondes avant de retourner en sommeil automatiquement.

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

wakeupmonitor→set_sleepCountdown(newval)

Modifie le temps avant le prochain sommeil .

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

wakeupmonitor→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

wakeupmonitor→sleep(secBeforeSleep)

Déclenche une mise en sommeil jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

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

wakeupmonitor→sleepFor(secUntilWakeUp, secBeforeSleep)

Déclenche une mise en sommeil pour un temps donné ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

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

wakeupmonitor→sleepUntil(wakeUpTime, secBeforeSleep)

Déclenche une mise en sommeil jusqu'à une date donnée ou jusqu'à la prochaine condition de réveil, l'heure du RTC du module doit impérativement avoir été réglée au préalable.

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

wakeupmonitor→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

wakeupmonitor→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

wakeupmonitor→wakeUp()

Force un réveil.

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

9.9. La classe YWakeUpSchedule

Interface pour intéragir avec les réveils agendés, disponibles par exemple dans le YoctoHub-GSM-4G, le YoctoHub-Wireless-SR, le YoctoHub-Wireless-g et le YoctoHub-Wireless-n

La classe `YWakeUpSchedule` implémente une condition de réveil. Le réveil est spécifié par un ensemble de mois et/ou jours et/ou heures et/ou minutes où il doit se produire.

Pour utiliser les fonctions décrites ici, vous devez inclure:

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

Fonction globales

YWakeUpSchedule.FindWakeUpSchedule(func)

Permet de retrouver un réveil agendé d'après un identifiant donné.

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

YWakeUpSchedule.FindWakeUpScheduleInContext(yctx, func)

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

java uwp ts es

YWakeUpSchedule.FirstWakeUpSchedule()

Commence l'énumération des réveils agendés accessibles par la librairie.

cpp m pas vb cs java uwp py php ts es

YWakeUpSchedule.FirstWakeUpScheduleInContext(yctx)

Commence l'énumération des réveils agendés accessibles par la librairie.

java uwp ts es

YWakeUpSchedule.GetSimilarFunctions()

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

dnp

Propriétés des objets YWakeUpScheduleProxy

wakeupschedule→AdvertisedValue *[lecture seule]*

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

dnp

wakeupschedule→FriendlyName *[lecture seule]*

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

dnp

wakeupschedule→FunctionId *[lecture seule]*

Identifiant matériel du réveil agendé, sans référence au module.

dnp

wakeupschedule→HardwareId *[lecture seule]*

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

dnp

wakeupschedule→Hours *[modifiable]*

S heures où le réveil est actif..

dnp

wakeupschedule→IsOnline *[lecture seule]*

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

dnp

wakeupschedule→LogicalName *[modifiable]*

Nom logique de la fonction.

dnp

wakeupschedule→MinutesA *[modifiable]*

S minutes de l'interval 00-29 de chaque heures où le réveil est actif.

dnp

wakeupschedule→MinutesB *[modifiable]*

S minutes de l'interval 30-59 de chaque heure où le réveil est actif.

dnp

wakeupschedule→MonthDays *[modifiable]*

S jours du mois où le réveil est actif.

dnp

wakeupschedule→Months *[modifiable]*

S mois où le réveil est actif.

dnp

wakeupschedule→NextOccurence *[lecture seule]*

Date/heure de la prochaine occurence de réveil.

dnp

wakeupschedule→SecondsBefore *[modifiable]*

Nombre de secondes d'anticipation pour allumer le système avant l'heure de réveil choisie **Modifiable**.

dnp

wakeupschedule→SerialNumber *[lecture seule]*

Numéro de série du module, préprogrammé en usine.

dnp

wakeupschedule→WeekDays *[modifiable]*

S jours de la semaine où le réveil est actif.

dnp

Méthodes des objets YWakeUpSchedule

wakeupschedule→clearCache()

Invalide le cache.

cpp m pas vb cs java py php ts es

wakeupschedule→describe()

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

cpp m pas vb cs java py php ts es

wakeupschedule→get_advertisedValue()

Retourne la valeur courante du réveil agendé (pas plus de 6 caractères).

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

wakeupschedule→get_errorMessage()

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

cpp m pas vb cs java py php ts es

wakeupschedule→get_errorType()

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

cpp m pas vb cs java py php ts es

wakeupschedule→get_friendlyName()

Retourne un identifiant global du réveil agendé au format NOM_MODULE.NOM_FONCTION.

cpp m cs java py php ts es dnp

wakeupschedule→get_functionDescriptor()

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

cpp m pas vb cs java py php ts es

wakeupschedule→get_functionId()

Retourne l'identifiant matériel du réveil agendé, sans référence au module.

cpp m vb cs java py php ts es dnp

wakeupschedule→get_hardwareId()

Retourne l'identifiant matériel unique du réveil agendé au format SERIAL.FUNCTIONID.

cpp m vb cs java py php ts es dnp

wakeupschedule→get_hours()

Retourne les heures où le réveil est actif..

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

wakeupschedule→get_logicalName()

Retourne le nom logique du réveil agendé.

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

wakeupschedule→get_minutes()

Retourne toutes les minutes de chaque heure où le réveil est actif.

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

wakeupschedule→get_minutesA()

Retourne les minutes de l'interval 00-29 de chaque heures où le réveil est actif.

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

wakeupschedule→get_minutesB()

Retourne les minutes de l'interval 30-59 de chaque heure où le réveil est actif.

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

wakeupschedule→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

wakeupschedule→get_module_async(callback, context)

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

wakeupschedule→get_monthDays()

Retourne les jours du mois où le réveil est actif.

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

wakeupschedule→get_months()

Retourne les mois où le réveil est actif.

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

wakeupschedule→get_nextOccurence()

Retourne la date/heure de la prochaine occurence de réveil.

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

wakeupschedule→get_secondsBefore()

Retourne le nombre de secondes d'anticipation pour allumer le système avant l'heure de réveil choisie

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

wakeupschedule→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

wakeupschedule→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

wakeupschedule→get_weekDays()

Retourne les jours de la semaine où le réveil est actif.

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

wakeupschedule→isOnline()

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

cpp m pas vb cs java py php ts es dnp

wakeupschedule→isOnline_async(callback, context)

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

wakeupschedule→isReadOnly()

Indique si la fonction est en lecture seule.

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

wakeupschedule→load(msValidity)

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

cpp m pas vb cs java py php ts es

wakeupschedule→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

wakeupschedule→load_async(msValidity, callback, context)

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

wakeupschedule→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

wakeupschedule→nextWakeUpSchedule()

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

cpp m pas vb cs java uwp py php ts es

wakeupschedule→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

wakeupschedule→set_hours(newval)

Modifie les heures où un réveil doit avoir lieu.

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

wakeupschedule→set_logicalName(newval)

Modifie le nom logique du réveil agendé.

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

wakeupschedule→set_minutes(bitmap)

Modifie toutes les minutes où un réveil doit avoir lieu

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

wakeupschedule→set_minutesA(newval)

Modifie les minutes de l'interval 00-29 où un réveil doit avoir lieu.

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

wakeupschedule→set_minutesB(newval)

Modifie les minutes de l'interval 30-59 où un réveil doit avoir lieu.

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

wakeupschedule→set_monthDays(newval)

Modifie les jours du mois où un réveil doit avoir lieu.

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

wakeupschedule→set_months(newval)

Modifie les mois où un réveil doit avoir lieu.

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

wakeupschedule→set_secondsBefore(newval)

Modifie le nombre de secondes d'anticipation pour allumer le système avant l'heure de réveil choisie.

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

wakeupschedule→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

wakeupschedule→set_weekDays(newval)

Modifie les jours de la semaine où un réveil doit avoir lieu.

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

wakeupschedule→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

wakeupschedule→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

10. Problèmes courants

10.1. 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].

10.2. 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"
```

10.3. 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.

10.4. 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].

10.5. Module alimenté mais invisible pour l'OS

Si votre YoctoHub-GSM-4G 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.

10.6. 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

10.7. Déconnexions, comportement erratique

Si votre YoctoHub-GSM-4G 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].

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

Si une mise à jour du firmware de votre YoctoHub-GSM-4G 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-GSM-4G 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.

10.9. 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.

10.10. 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.

10.11. Impossible de contacter les sous-modules par USB

Le but du YoctoHub-GSM-4G 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-GSM-4G ne sert qu'à l'alimenter et le configurer. Pour accéder aux modules connectés au hub, vous devez impérativement passer par une connexion réseau.

10.12. Network Readiness coincé à 3- LAN ready

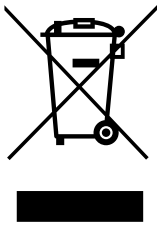
Vérifiez que votre connexion Internet sortante fonctionne et que vous n'avez pas un callback invalide défini dans la configuration to YoctoHub-GSM-4G.

10.13. Module endommagé

Yoctopuce s'efforce de réduire la production de déchets électroniques. Si vous avez l'impression que votre YoctoHub-GSM-4G 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

[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

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-GSM-4G, 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.

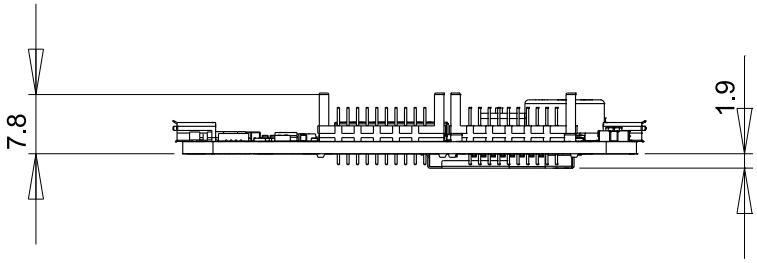
11. Caractéristiques

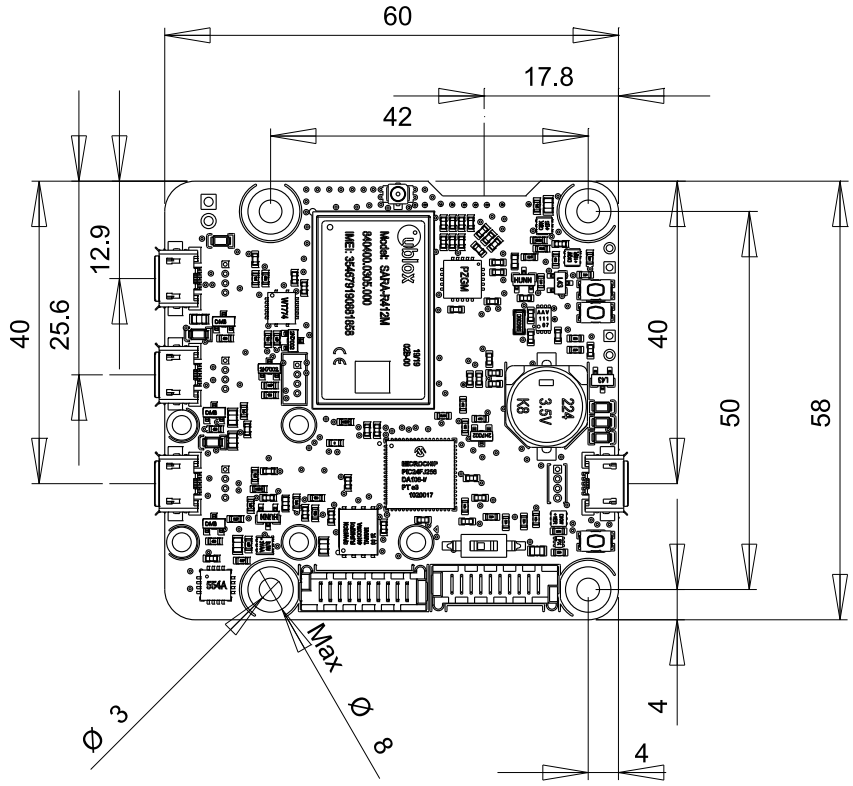
Vous trouverez résumées ci-dessous les principales caractéristiques techniques de votre module YoctoHub-GSM-4G

Identifiant produit	YHUBGSM5
Révision matérielle†	
Connecteur USB	micro-B
Epaisseur	9.5 mm
Largeur	58 mm
Longueur	60 mm
Poids	33.2 g
Chipset	u-Blox SARA-R412M-02B
Classe de protection selon IEC 61140	classe III
Temp. de fonctionnement normale	5...40 °C
Temp. de fonctionnement étendue‡	-20...70 °C
Consommation USB	50 mA
Conformité RoHS	RoHS III (2011/65/UE+2015/863)
USB Vendor ID	0x24E0
USB Device ID	0x0034
Boîter recommandé	YoctoBox-HubWlan-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-GSM-4G

A4

Scale
1:1
Echelle