Yoctopuce Toolbox pour MATLAB

(version Beta)

Cette librairie offre un ensemble de classes permettant d'utiliser les capteurs et les actuateurs Yoctopuce dans un environnement MATLAB. Elle inclut une librairie dynamique qui gère la communication à bas niveau avec les modules, sans devoir installer de pilote de périphérique supplémentaire.

Prise en main

Toutes les classes sont rassemblées dans un package **YoctoProxyAPI**. Pour simplifier le code ci-dessous, nous importons toutes les classes du package, mais ce n'est évidemment pas obligatoire:

```
import YoctoProxyAPI.*
```

Toutes les fonctions générales de la librairie sont rassemblées sous forme de méthodes statiques dans la classe **`YAPIProxy`**. La première méthode à appeler, **`RegisterHub()`**, sert à indiquer où trouver les modules Yoctopuce. La librairie peut utiliser des modules connectés par le réseau, ou connectés directement par USB.

```
YAPIProxy.RegisterHub('usb')
```

```
ans =

  0×0 empty char array
```

Si nous avions voulu utiliser des modules connectés via un YoctoHub-Wireless-n par exemple, nous aurions remplacé la chaîne `'usb'` par une adresse IP comme `'192.168.1.100'`.

Voyons maintenant comment lister tous les modules détectés:

```
YModuleProxy.GetSimilarFunctions()
```

```
ans = 2×1 cell
'THRMCPL1-13405C.module'
'MXPWRRLY-FA815.module'
```

Cette fonction retourne une liste d'identifiant matériels, correspondant à chaque module trouvé. De manière similaire, on peut lister tous les capteurs de températures, quelle que soit leur type ou leur répartition sur des modules physiques, dans le cas de modules comportant plusieurs entrées:

```
YTemperatureProxy.GetSimilarFunctions()
```

```
ans = 2×1 cell
'THRMCPL1-13405C.temperature2'
'THRMCPL1-13405C.temperature1'
```

Pilotage des modules de commande

La librairie Yoctopuce fournit des classes distinctes permettant de piloter les sorties des modules de commande par le biais de propriétés. Par exemple, la classe `YRelayProxy` permet de piloter les sorties de relais, et la classe `YCurrentLoopOutputProxy` permet de piloter les sorties en boucle de courant 4-20mA. Voici un premier exemple d'utilisation:

```
pump = YRelayProxy.FindRelay('MXPWRRLY-FA815.relay3');
pump.State = 'State_B';
```

Si vous avez donné un nom logique à la sortie, par exemple à l'aide du VirtualHub, vous pouvez aussi retrouver le relais par son nom:

```
pump = YRelayProxy.FindRelay('WaterPump');
pump.pulse(2000);
```

Dans ce deuxième exemple, plutôt que de commuter le relais de manière définitive, nous lui avons demandé de commuter à l'état B pour une durée de 2000ms seulement.

Vous pouvez facilement consulter les propriétés intéressantes des objets d'interface:

```
pump
```

```
pump =
  YoctoProxyAPI.YRelayProxy with properties:

   General
    TargetFunction: 'WaterPump'
        HardwareId: 'MXPWRRLY-FA815.relay3'
      FriendlyName: 'MXPWRRLY-FA815.WaterPump'
             State: State_A

  Show all properties
```

Utilisation des modules de type capteur

Pour utiliser les capteurs, le principe est très similaire: vous recherchez la fonction désirée par son identifiant matériel ou le nom logique que vous lui avez préalablement attribué, puis vous pouvez lire sa valeur dans la propriété **`CurrentValue`**. L'unité de mesure est disponible dans la propriété **`Unit`**. Si vous affichez l'objet, vous y trouverez les autres propriétés utiles pour le capteur, comme par exemple ici dans le cas d'un Yocto-Thermocouple:

```
tc = YTemperatureProxy.FindTemperature('')
```

```
tc =
  YoctoProxyAPI.YTemperatureProxy with properties:

   General
    TargetFunction: '(any)'
        HardwareId: 'THRMCPL1-13405C.temperature2'
      FriendlyName: 'THRMCPL1-13405C.temperature2'
      CurrentValue: 27.6200
              Unit: ''C'

  Show all properties
```

```
measure = sprintf("%f %s", tc.CurrentValue, tc.Unit)
```

```
measure =
"27.620000 'C"
```

Vous pouvez constater que la description de l'objet inclut non seulement la température mesurée, mais aussi le type de thermocouple qui a été configuré.

Un petit exemple d'une pièce

Voici un petit exemple de processus de contrôle,:

```
import YoctoProxyAPI.*
YAPIProxy.RegisterHub('usb');
light = YLightSensorProxy.FindLightSensor('diningRoom');
lightControl = YRelayProxy.FindRelay('lightSwitch');
redButton = YAnButtonProxy.FindAnButton('redButton');

while ~redButton.IsPressed
    if light.CurrentValue < 30
        % turn on the light
        lightControl.State = 'State_B';
    end
end

% Libère toutes les ressources
YAPIProxy.FreeAPI();
```

Quelques remarques sur ce code:

- Vous pouvez constater que l'essentiel de l'interaction avec les modules Yoctopuce se fait via des propriétés, ce qui facilite la lecture du code.
- De plus, l'accès aux propriétés est garanti d'être quasi-instantané, car elles sont rafraichie en tâche de fond par un processus indépendant. C'est la caractéristique de nos librairies "Proxy".
- L'appel à **`FreeAPI()`** à la fin libère le port USB et déconnecte les objets MATLAB des modules Yoctopuce

Plateformes supportées

Toutes les classes sont définies en langage MATLAB, mais elles délèguent une grande partie du travail à la librairie C++ Proxy, fournie sous forme de librairie dynamique. La distribution de la toolbox inclut des binaires de la librairie dynamique pour Windows et Linux en 32/64 bit et pour MacOS en 64 bit, ce qui devrait permettre de l'utiliser sur toutes ces plateformes (on a pas encore testé MacOS). Le code source de la librairie dynamique est disponible.

Ce qui n'est pas encore disponible dans cette version Beta

Les exemples pour chaque module Yoctopuce

Les librairies Yoctopuce incluent toutes un exemple d'utilisation pour chaque capteur. La librairie MATLAB ne fera pas exception, vous aurez droit à un Live Script pour chaque module lorsque la librairie sera officiellement disponible. Mais pour l'instant, il faudra vous contenter des exemples dans ce fichier. L'aide en ligne dans MATLAB devrait néanmoins vous permettre deretrouver ce dont vous avez besoin durant la période de Beta-test.

La documentation exhaustive

Nous n'avons pas encore généré le fichier de référence détaillant l'usage de chaque classe. Par contre vous avez déjà l'aide en ligne dans MATLAB. Si vous voulez des explications sur les propriétés et la liste des

classes, vous pouvez consulter en attendant la documentation de la librairie ".Net Proxy" qui est basée sur la même structure.

Utilisation du Data Logger

Les capteurs Yoctopuce sont capables d'enregistrer des mesures sur leur mémoire flash intégrée. Nous avons prévu de permettre d'y accéder facilement sous forme de `TimeTable`, mais cela n'est pas encore disponiible. Le seul défaut des *TimeTables* est qu'elles n'ont apparu que dans MATLAB 2016b. Si possible, nous fournirons donc aussi une interface basée sur les *Time Series*.

Utilisation avec Simulink

Nous avons choisi d'implémenter les classes de notre librairie sous forme de MATLAB System Objects. Cela devrait permettre de les utiliser directement dans Simulink à l'aide de System Blocks, mais nous reviendrons ultérieurement avec plus d'informations sur ce sujet lorsque nous aurons avancé sur le sujet.