Communications entre JMRI et Arduino

Pour pilotez votre réseau avec JMRI

Le 1er janvier 2019. Par : Dominique, Nopxor

JMRI [1] est un projet et une communauté de développeurs qui ont créé des outils en Java pour le contrôle par ordinateur des modèles réduits ferroviaires. Ces outils sont accessibles selon les règles de la Free Software Foundation's GNU General Public License (GPL). Il comprend notamment DecoderPro® qui permet de programmer les décodeurs et un panneau de commande PanelPro™ pour configurer la topographie et la signalisation.

On trouvera un tour d'horizon des applications et possibilités de JMRI en français ici [2], ainsi qu'une liste des matériels supportés parmi lesquels on trouve maintenant les centrales DCC à base du logiciel DCC++ [3], donc l'Arduino, mais il manquait une interface plus générale entre JMRI et Arduino pour remonter les états de la rétro-signalisation et pour actionner les appareils de voie.

Geoff Bunza est un modéliste américain de Portland (Orégon) qui publie de nombreux projets sur le modélisme et l'Arduino [4]. Il nous a donné l'autorisation [5] de publier une traduction en français de deux de ses contributions sur les communications entre JMRI et Arduino.

C'est l'ensemble de ces opportunités qui nous a amené a publier cet article largement inspiré du blog de Geoff. Toute l'équipe de Locoduino le remercie (ainsi que son traducteur Nopxor) d'en faire profiter ainsi les modélistes français.

Avant propos important

JMRI (Java Model Railroad Interface) est un logiciel gratuit fonctionnant sous Windows, Linux ou Macintosh et pouvant exécuter une multitude de fonctions. Il est bien connu pour la programmation des décodeurs DCC complexes grâce à son application **DecoderPro**. Il est également utilisé pour la modélisation et la gestion de réseau, la rétro-signalisation, la signalisation, la commutation des aiguilles, le contrôle des itinéraires, le contrôle des trains, l'interface avec les contrôles WiFi et les commandes avec des téléphones portables. La version 4.14 est la version actuelle de JMRI, du moins à la date de publication de cet article.

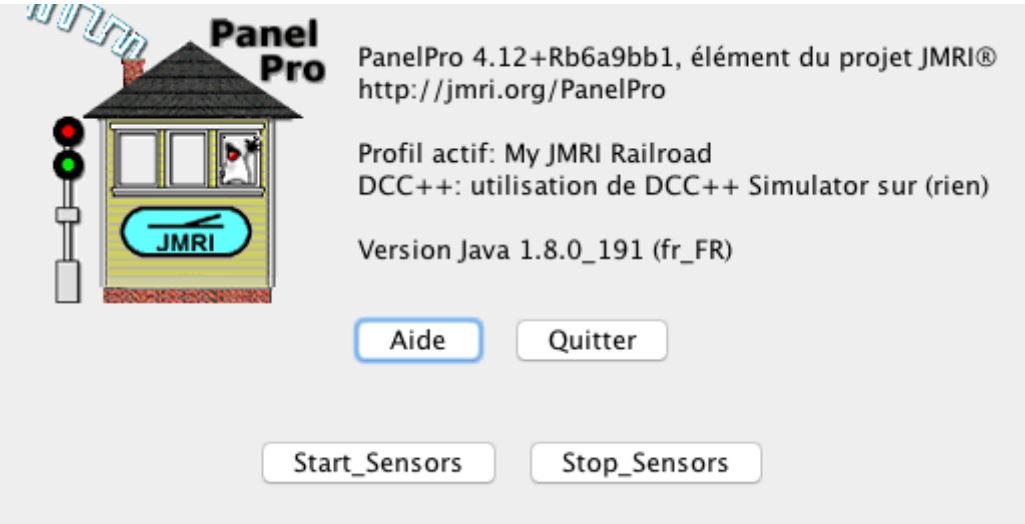
Cet article n'est pas une présentation ni un mode d'emploi complet de JMRI, il se limite à la mise en oeuvre des interfaces avec l'Arduino décrites ci-après.

Pour démarrer avec JMRI, il est recommandé de se rendre sur le site de JMRI indiqué plus haut. Pour les francophones purs et durs, on pourra démarrer avec la table des matières et l'index général en français où le lecteur pourra consulter en détail chacune des notions traitées dans cet article comme les tableaux de capteurs et les tableaux d'aiguillages.

JMRI a, en particulier, l'avantage de s'interfacer à de nombreuses technologies grâce à l'ajout de scripts Python (langage Jython très proche). Ici aussi, il n'est pas question de présenter Python, mais chacun pourra comprendre la programmation qui est utilisée dans les scripts de cet article car ils ne font appel qu'à des notions extrêmement basique où Python se révèle semblable au C, le langage de l'Arduino.

A titre d'exemple, les scripts présentés ici servent à établir des communications entre JMRI sur votre PC et l'Arduino, via le port USB. Accessoirement JMRI offre la possibilité d'appeler manuellement un script à l'aide d'un bouton que vous pourrez ajouter sur la fenêtre d'accueil de JMRI comme celle-ci :

Communications entre JMRI et Arduino



Exemple de fenêtre de démarrage personnalisée
Les boutons *Start_sensors* et *Stop_sensors* sont bien entendu des exemples.

L'article vous montrera en détail comment mettre en oeuvre ces scripts qui sont livrés tout fait avec le programme Arduino.

Autre remarque importante : On peut choisir la langue française dans les préférences de JMRI, comme dans tout logiciel. Sauf que les développeurs de JMRI n'ont pas traduit toute l'interface et il reste des menus déroulants et boites à cocher en anglais. Le texte de cet article en tient compte.

1. La remontée de rétro-signalisations à JMRI à partir des ports d'entrée/sortie d'un Arduino.

1ère Partie - Communications de Arduino vers JMRI - rétro-signalisation

D'après l'article : JMRI Sensors A plenty for Any Layout

Plusieurs des fonctions de JMRI doivent savoir où se trouvent les locomotives et les wagons. Il doit donc recevoir les états de la rétro-signalisation. JMRI utilise des capteurs pour cela, regroupés dans un tableau de capteurs.

Traditionnellement, les modélistes utilisent la connexion entre JMRI et leur centrale pour remonter la rétro-signalisation ; selon des protocoles tels que S88, LocoNet, ExpressNet, etc.. qui nécessitent l'achat d'une centrale du commerce. Nous allons voir comment faire communiquer ces capteurs de rétro-signalisation avec JMRI, par l'intermédiaire de l'Arduino, sans matériel supplémentaire.

Des quantités de capteurs pour JMRI à partir d'un Arduino

Cet article détaille l'implémentation d'un jeu ou *canal* de capteurs, qui se connecte directement à JMRI et non via une station de base DCC et vous pourrez exploiter plusieurs canaux en branchant plusieurs Arduino !

Ce projet vous montrera comment connecter jusqu'à 68 capteurs à JMRI avec un seul Arduino et comment faire pour configurer vos capteurs dans JMRI. Cela vous permettra de choisir n'importe quel type de capteur actif à l'état bas (ou à l'état haut, on verra comment) et de l'acheminer vers un

tableau de capteurs JMRI. Il initialisera automatiquement tous les capteurs que vous avez configurés lors de la connexion à JMRI. La connexion matérielle sera faite avec la plupart des cartes Arduino du commerce, mais il y aura beaucoup moins de capteurs avec un Arduino Uno ou Nano. Les Arduino se connecteront via n'importe quel port USB à votre ordinateur exécutant JMRI. Ce sera le même port série USB que vous utilisez pour programmer l'Arduino. On verra aussi comment configurer la table des capteurs dans JMRI.

Le code pour Arduino et pour JMRI est disponible ici : Sensor_Scan1_4

sensor_scan1_4

La désignation AR est complètement arbitraire, mais si vous souhaitez la modifier, vous devez éditer le script **JMRI Sensor_Scan Python** fourni.

Lorsque vous téléchargez le sketch Arduino sur votre carte, vous devez **noter le port série de votre ordinateur utilisé pour la communication entre l'IDE Arduino et votre carte**. Vous devez toujours brancher votre câble USB Arduino sur le même port (prise USB) afin que votre système d'exploitation puisse attribuer le même port série. Si vous changez d'ordinateur, il est probable que le numéro de port USB soit différent. Ceci est important et vous devez vous assurer que le script JMRI Sensor_Scan Python indique bien le port utilisé ligne 82 :

Sur Windows :

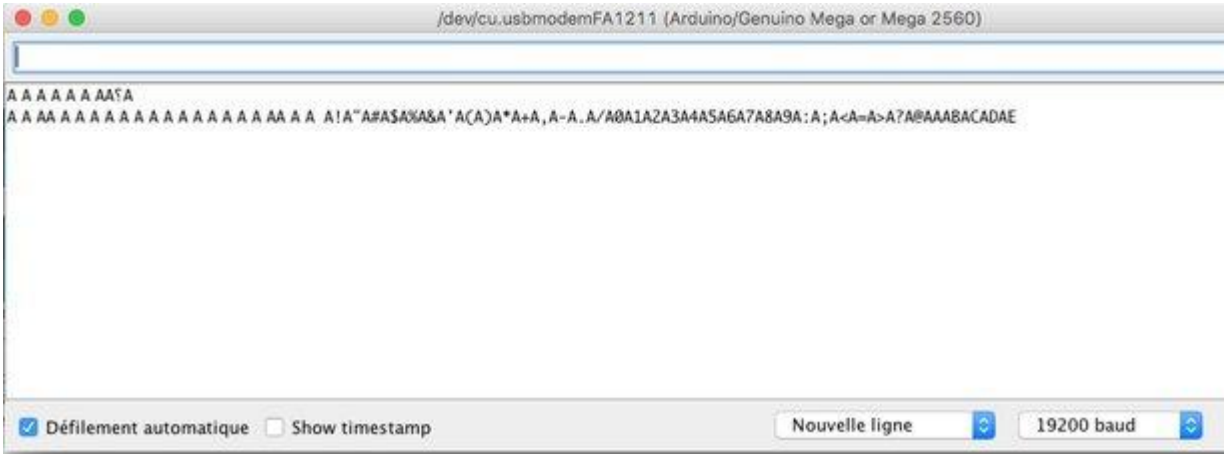
```
1. a = SerialSensorMux ("COM5")
```

Sur Macintosh :

```
1. a = SerialSensorMux("/dev/cu.usbmodemFA1211")
```

Téléchargez le sketch approprié sur votre Arduino, montez un sensor shield si nécessaire et laissez la carte connectée à votre ordinateur sur lequel vous exécuterez **JMRI**.

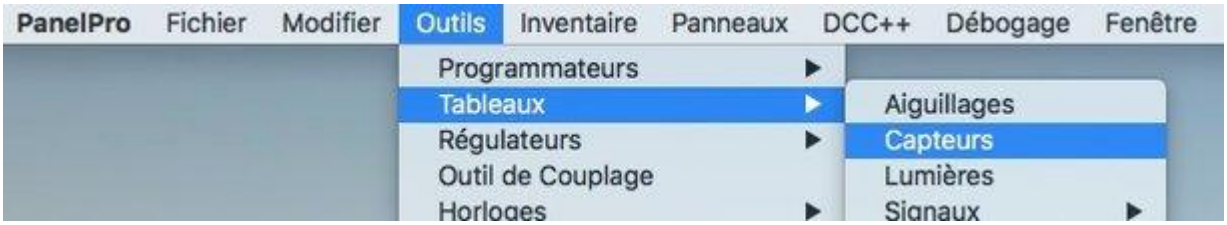
Vous pouvez tester que le sketch fonctionne en ouvrant la fenêtre du moniteur série réglée sur la vitesse 19200 bauds, puis en tapant un caractère puis validez : une série de lettres `A` suivie d'un caractère variable s'affiche : C'est bon signe !



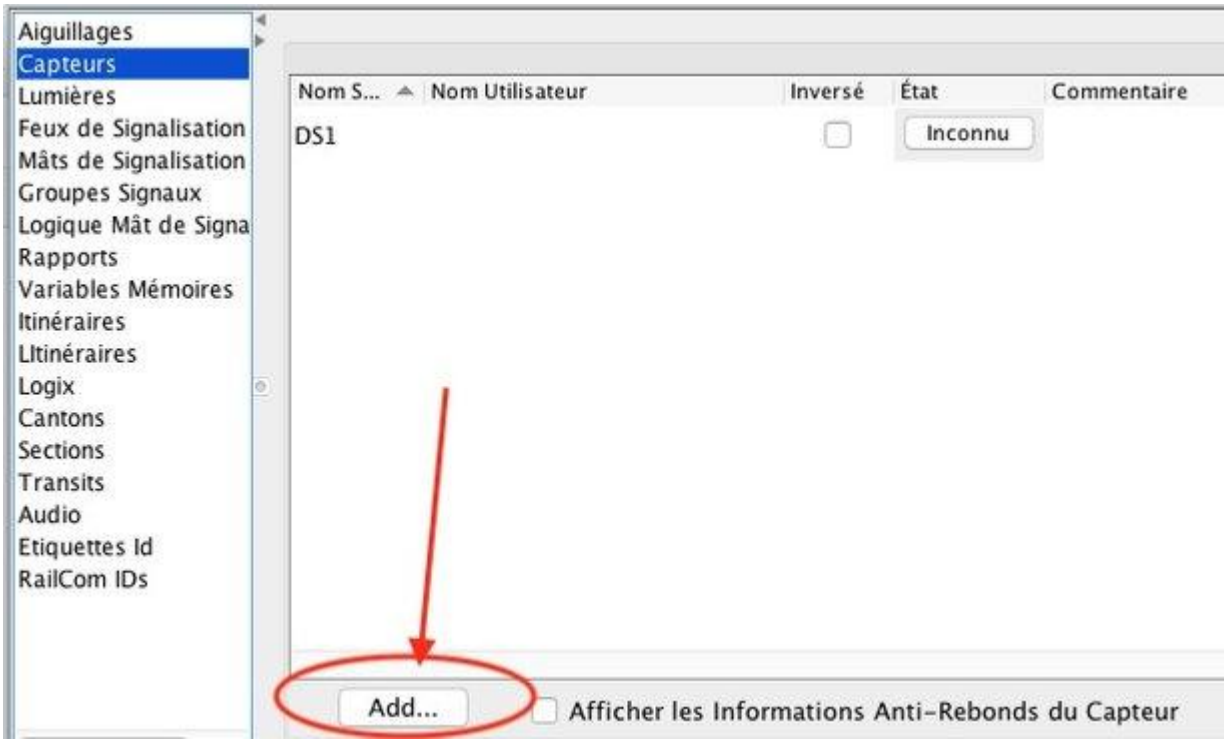
Communications entre JMRI et Arduino

Maintenant, avant de démarrer **JMRI**, vous devez soit avoir mis votre carte sous tension, soit simplement avoir appuyé sur le bouton de reset de votre carte.

Ouvrez **PanelPro** et ouvrez votre tableau de capteurs.

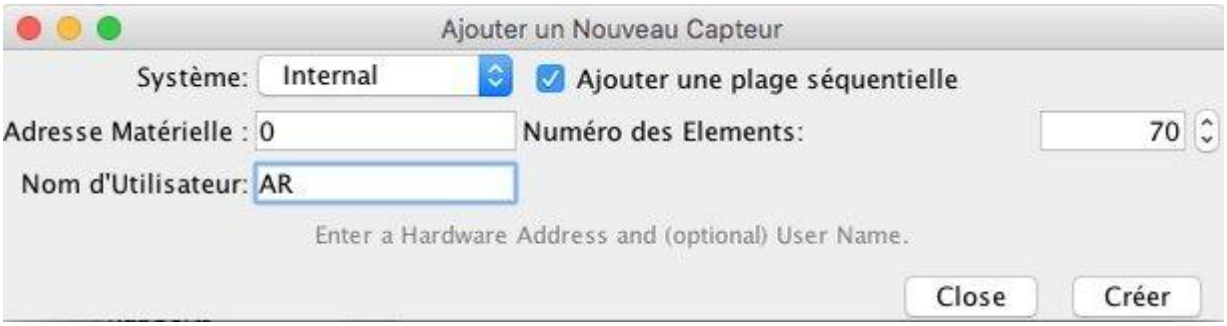


Allez au bas de la fenêtre du tableau des capteurs et cliquez sur **Add**…



La connexion système peut être définie pour des adresses internes ou matérielles.
Sélectionnez *Internal* dans la liste déroulante.

- Dans l'espace *Adresse matérielle*, indiquez simplement *0* (zéro) ;
- Dans l'espace *Nom d'utilisateur*, mettez simplement *AR* ;
- Cochez la case *Ajouter une plage séquentielle* ;
- Pour le nombre d'éléments, entrez *70* pour un Mega 2560 et *20* pour un Uno, Pro Mini ou un Nano.
- Puis cliquez sur *Créer*



Retournez à votre tableau de capteurs et cliquez sur l'étiquette *Nom d'utilisateur* en haut de la colonne pour obtenir un tri propre par tous les noms d'utilisateur que vous venez de créer.

Supprimez maintenant les entrées *IS0* et *IS1*, car nous ne les utiliserons pas. Vous devez maintenant avoir 68 capteurs dans le tableau avec les noms d'utilisateur *IS2* à *IS68*. Ceux-ci seront initialisés par votre Mega au démarrage. Enregistrez votre travail en tant que panneau en utilisant *Panneaux* ➔ *Enregistrer les panneaux…*, nommez-le et enregistrez-le pour plus tard.

Maintenant, localisez l'endroit ou vous avez dézippé le fichier *Sensor_Scan.py* que vous avez modifié avec le port COM approprié.

Réinitialisez l'Arduino Mega (avec le bouton reset). Dans le menu JMRI, allez à *Panneaux* ➔ *Exécuter le script* et sélectionnez votre fichier *Sensor_Scan.py*. Exécutez le script et vous devriez voir tous vos capteurs passer d'Inconnu à Inactif ou Actif, en fonction de ce que vous avez connecté à chaque broche.

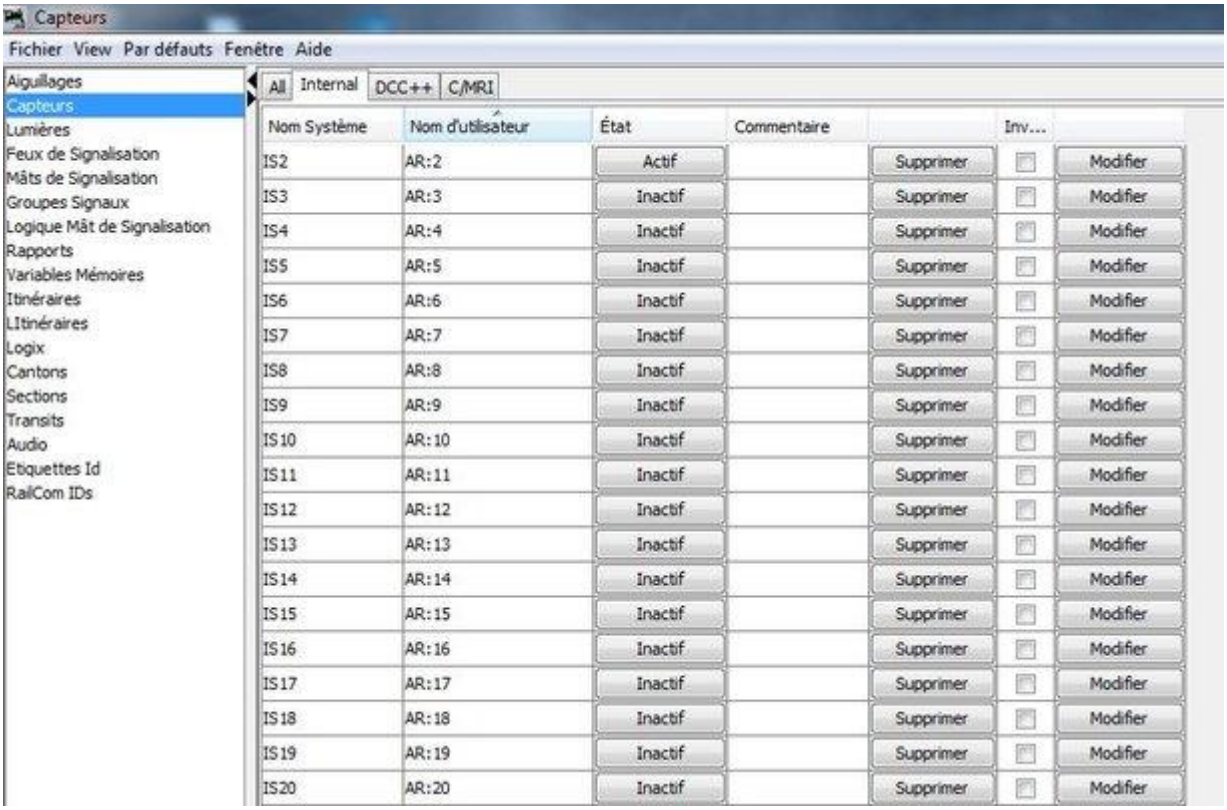
Rappelez-vous que connecter une broche de signal numérique (S) à la masse (G ou GND) active le capteur. Si vous utilisez des capteurs qui sont « Actif haut », c'est-à-dire qu'ils sont au niveau haut lorsqu'ils détectent quelque chose, alors il faut changer la ligne 18 du croquis Arduino :

```
1. #define Sensors_Active_Low 1
```

par :

```
1. #define Sensors_Active_Low 0
```

Communications entre JMRI et Arduino



Le capteur IS2 (AR2) est au niveau bas, il est donc actif.

Comment ça marche

Les Arduino ont une ligne série disponible pour une utilisation générale. C'est le même port série utilisé pour télécharger le sketch dans la carte. Une fois configuré, le même port série est disponible pour la communication avec votre ordinateur.

Il existe actuellement de nombreux composants internes à JMRI, et JMRI fournit un accès à sa structure interne via des *scripts* écrits en Python, ainsi qu'en *jython* pour Java Python.

Ces scripts permettent aux modélistes d'ajouter des fonctionnalités supplémentaires à JMRI sans développer une interface majeure pour JMRI telle que Digitrax, CMRI ou DCC++.

L'Arduino attend qu'un jeu de caractères du script JMRI initie la communication. Lorsque le script JMRI *Sensor_Scan.py* démarre, il envoie le message d'initialisation à l'Arduino. Il active ensuite la mise à jour de tous les capteurs dont il est responsable (c'est-à-dire toutes les broches valides pour lesquelles il est configuré). L'Arduino envoie ensuite un message de mise à jour du capteur sur 2 octets à 19200 bauds à JMRI, qui met ensuite à jour le capteur approprié, en faisant correspondre le numéro du capteur reçu de l'Arduino au nom défini par l'utilisateur, comme AR : 54.

Le programme Arduino

Ce programme est écrit en anglais, la langue de l'auteur. Nous avons décidé de le laisser tel quel, mais vous comprendrez facilement comment il marche.

Communications entre JMRI et Arduino

Ce programme est très simple et très court : Avant le setup(), la configuration et variables locales sont définies.

```
#define Sensor_Pin_Max   70  // Max sensor pin NUMBER (plus one) Mega=70,Uno,Pro Mini,Nano=20
#define Sensor_Pin_Start  2  // Starting Sensor Pin number (usually 2 as 0/1 are TX/RX
#define Sensor_Offset     0  // This Offset will be ADDED to the value of each Sensor_Pin to determine the sensor
                             // number sent to JMRI, so pin D12 will set sensor AR:(12+Sensor_Offset) in JMRI
                             // This would allow one Arduino Sensor channel to set sensors 2-69 and another to
                             // Set sensors 70-137 for example; this offset can also be negative
#define Sensors_Active_Low 1 // Set Sensors_Active_Low to 1 if sensors are active LOW
                             // Set Sensors_Active_Low to 0 if sensors are active HIGH
#define open_delay 15        // longer delay to get past script initialization
 #define delta_delay 4        // Short delay to allow the script to get all the characters
 int i;
 char  sensor_state [70];     // up to 70 sensors on a Mega 2560
 char  new_sensor_state ;     // temp to process the possible state change
 char  incomingByte = 0;      // working temp for character processing
```

Puis, dans le setup(), le programme attend une séquence de caractères en provenance de JMRI, puis lit et sauvegarde l'état de tous les capteurs dans sa table `sensor_state[]` et envoie à PanelPro les états correspondants pour mettre à jour le tableau des capteurs. Maintenant il est prêt à transmettre tous les prochains changements.

```
void setup(){
    Serial.begin(19200);              // Open serial connection.
    while (Serial.available() == 0);  // wait until we get a charater from JMRI
    incomingByte=Serial.read();       // get the first character
    while ((Serial.available() > 0) && (incomingByte != '!')) incomingByte=Serial.read(); //get past !!!
    while ((Serial.available() > 0) ) incomingByte=Serial.read(); //flush anything else
    delay(open_delay);                 // take a breath
    for ( i=Sensor_Pin_Start; i<Sensor_Pin_Max; i++)  {  //Initialize all sensors in JMRI and grab each sensor
       pinMode(i, INPUT_PULLUP);       // define each sensor pin as coming in
        sensor_state[i] = (digitalRead( i ))^Sensors_Active_Low;    // read & save each sensor state & invert if necessary
        Serial.print("A"); Serial.print (char((sensor_state[i]<<7)+i+Sensor_Offset));  // send "A <on/off><snesor #>" to JMRI script
        delay(delta_delay);             // in milliseconds, take a short breath as not to overwhelm JMRI's seraial read
     }
 }
```

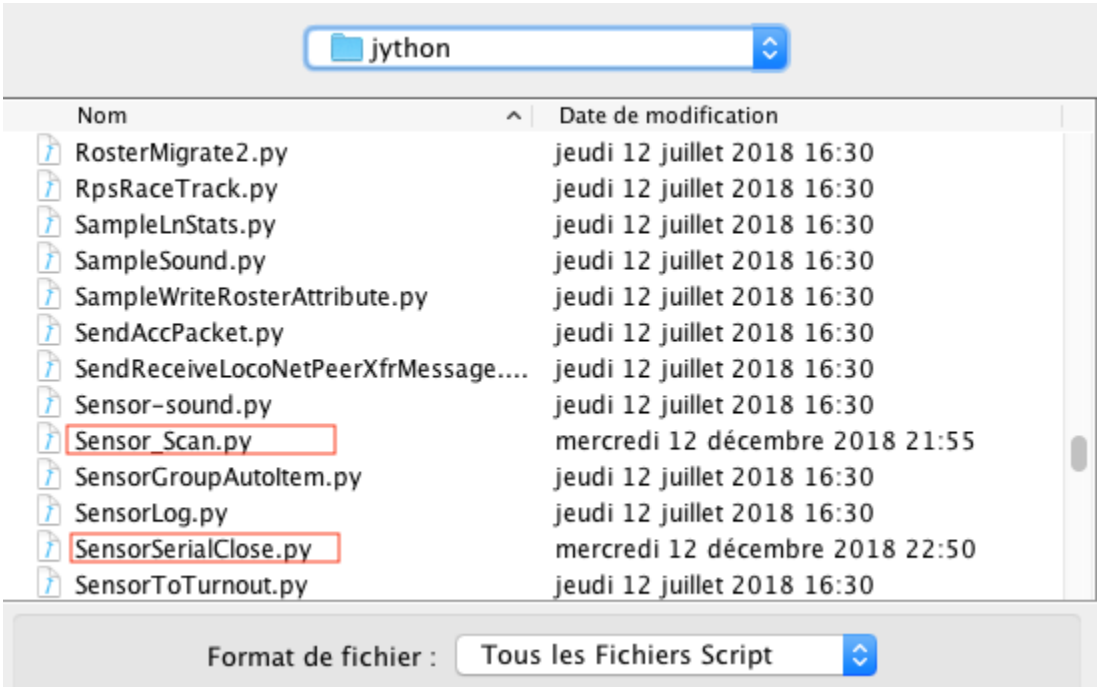
Ensuite dans la loop(), de façon répétitive, le programme teste tous les capteurs et, en cas de changement, envoie la lettre A suivie d'un caractère signifiant on/off et enfin le numéro du capteur.

```
1. void loop()  {
2.    for ( i=Sensor_Pin_Start; i<Sensor_Pin_Max; i++)  {     // scan every
   sensor over and over for any sensor changes
3.        new_sensor_state = (digitalRead( i ))^Sensors_Active_Low;  // read &
   save each sensor state & invert if necessary
4.        if (new_sensor_state != sensor_state[i] )  {               // check if
   the sensor changed -> if yes update JMRI
5.          Serial.print("A"); Serial.print
   (char((new_sensor_state<<7)+i+Sensor_Offset)); // send "A <on/off><sensor #>"
   to JMRI script
6.          sensor_state[i] = new_sensor_state ;                    // save the
   updated sensor state
7.          delay(delta_delay);            // in milliseconds, take a short
   breath as not to overwhelm JMRI's seraial read
8.        }
9.     }
10. }
```

Notes JMRI supplémentaires

Ranger vos scripts au bon endroit

Les scripts Python que vous avez téléchargés en lisant cet article se trouvent quelque part dans votre ordinateur, dans le dossier de téléchargement de votre navigateur, mais ce n'est pas leur meilleure place ! Lorsque vous exécutez la commande Panneaux / Exécuter le script… vous voyez que le dossier par défaut se nomme jython et se trouve dans le même dossier que l'application PanelPro. Une bonne pratique consiste donc à déplacer les scripts que vous avez téléchargés dans ce dossier jython, après avoir modifié le nom du port de communication et, éventuellement, inversé les ports qui seraient actifs bas.

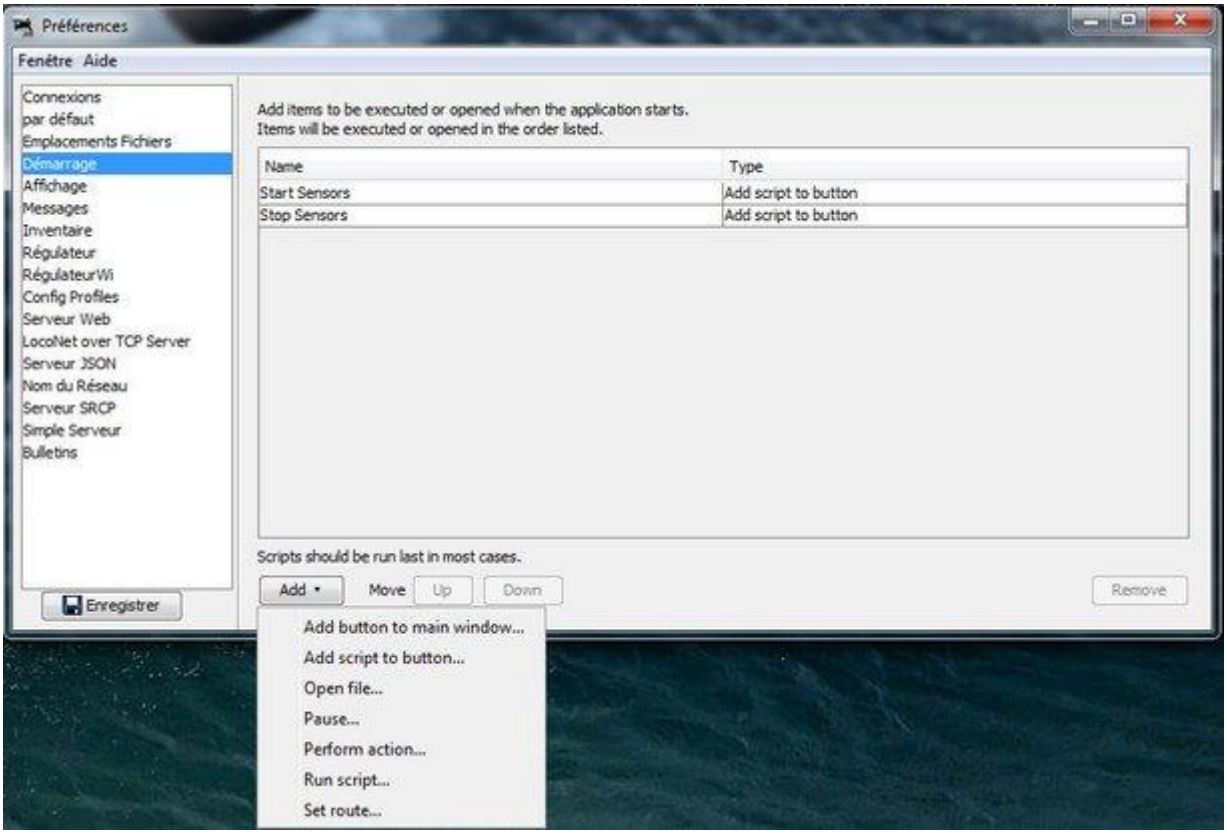


Communications entre JMRI et Arduino

L'ajout de boutons sur la fenêtre d'accueil Panel Pro

Une fois que vous ouvrez JMRI PanelPro, dans le menu, cliquez sur *Modifier* ➔ *Préférences…* Une fenêtre Préférences s'ouvrira. Cliquez sur Démarrage, puis en bas de la fenêtre, cliquez sur *Add,* puis sur *Add script to button…*

Donnez un nom au bouton (ce sera le texte sur le bouton) et naviguez jusqu'au script que vous voulez exécuter lorsque vous cliquez sur le bouton puis sur OK



Si vous avez enregistré votre panneau, vous pouvez également cliquer sur *Add* ➔ *Open file…* et laisser JMRI ouvrir votre panneau au démarrage.

Vous pouvez également cliquer sur le bouton *Add button to main window…* pour avoir des boutons pour ouvrir des tableaux, des fenêtres de sortie de script, allumer l'alimentation, etc.

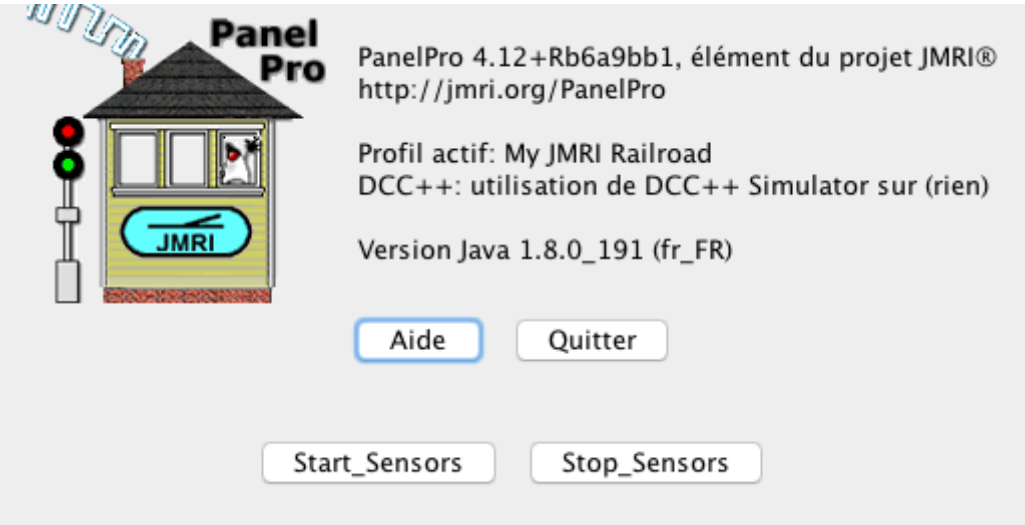
Toutes ces mêmes actions peuvent être configurées pour s'exécuter à la mise sous tension en sélectionnant *Add* ➔ *Perform action…* de manière similaire.

Une fois que vous avez configuré tout cela, assurez-vous de l'ordre de la liste des actions de démarrage en sélectionnant un élément dans la liste (clic simple) et en cliquant sur Move Up / Down au bas de la fenêtre.

Une fois que vous avez terminé, cliquez sur Enregistrer dans le coin inférieur gauche de la fenêtre. Il vous sera demandé si vous voulez redémarrer.

Communications entre JMRI et Arduino

Quand vous redémarrez, JMRI apparaîtra avec tous les boutons placés dans l'ordre d'entrée dans la table, et toutes les actions que vous avez spécifiées seront effectuées au démarrage de JMRI.



Tout ou partie de ceux-ci peuvent être supprimés en sélectionnant la ligne d'action et en utilisant le bouton *Remove* en bas à droite de la fenêtre.

Mais là nous allons au delà de l'objectif de cet article. Nous vous conseillons de lire la documentation de JMRI.

Avant de démarrer JMRI, n'oubliez pas d'allumez l'Arduino ou d'appuyez sur le bouton reset de l'Arduino.

Il existe un script appelé *SensorSerialClose.py* dans le fichier zip. Si vous le lancez, il force la fermeture du port série pour lequel il est configuré. Si le script *Sensor_Scan.py* est en cours d'exécution, cela a pour effet de mettre fin à ce script.

Cela évite à JMRI de laisser ouvert le port COM, ce qui empêcherait de ré-exécuter *Sensor_Scan.py* avec succès.

Si vous exécutez *SensorSerialClose.py*, le port sera fermé et cela vous permettra de :

1. Redémarrer l'Arduino en appuyant sur le bouton reset
2. Redémarrer le script Sensor_Scan avec succès.

Les scripts *Sensor_Scan.py* et *SensorSerialClose.py* doivent tous deux faire référence au même port COM. Vous pouvez copier et renommer ces 2 scripts pour chaque canal de port COM que vous créez.

Vous pouvez avoir plusieurs canaux ou groupes de capteurs fonctionnant en même temps via différents ports série USB. Dans les sketchs Arduino, vous trouverez des informations sur la manière de définir la plage de numéros de capteurs pouvant être attribués à chaque groupe.

Vous pouvez utiliser autant de groupes de capteurs que vous avez des ports série USB.

Communications entre JMRI et Arduino

Pour exécuter plusieurs groupes de capteur, vous devez rendre chaque script de balayage de capteurs indépendant l'un de l'autre. Vous devrez éditer et copier chaque script pour chaque port COM et modifier les lignes suivantes :

ligne 17 : `global extporta # le nom extporta doit être unique pour chaque port de communication`
ligne 22 : `self.port = extporta # comme ci-dessus`
ligne 23 : `extporta = self.port # comme ci-dessus`
ligne 82 : `a = SerialSensorMux ("COM8") # Ce COMx DOIT correspondre à un canal Arduino spécifique.`
ligne 85 : `a.setName ("script SerialSensorMux8") # Attribuez-lui un nom unique afin d'éviter toute confusion.`

Pour une raison interne au logiciel, on ne peut pas utiliser export8, export9, etc. Ceux-ci doivent se terminer par un caractère alphabétique et non par un nombre, comme exporta, exportb, exportc, …

Cet article vous permet donc d'établir une connexion entre des données de JMRI et des broches d'un ou plusieurs Arduino, dans les 2 sens. Cela suppose que vous avez prévu de connecter directement vos équipements (capteurs et/ou actionneurs) sur un ou plusieurs Arduino Mega (ou Uno/Nano/Mini).

Source :

https://www.locoduino.org/spip.php?article240