

# Interface entre Open Rails et une carte Arduino 18/01/2026

Sur ce site : [https://www.la-tour.info/uts/uts\\_page15.html#conduite](https://www.la-tour.info/uts/uts_page15.html#conduite)

Je présente un système global sur une carte Arduino DUE, pour gérer un pupitre de locomotive.

Il est conçu pour fonctionner avec "Train Simulator Classic RailWorks de DTG", et son interface "TSClassic Raildriver and Joystick Interface".

Le 18/01/2026 : Version 1.0. Exécutable et sources C# disponibles.

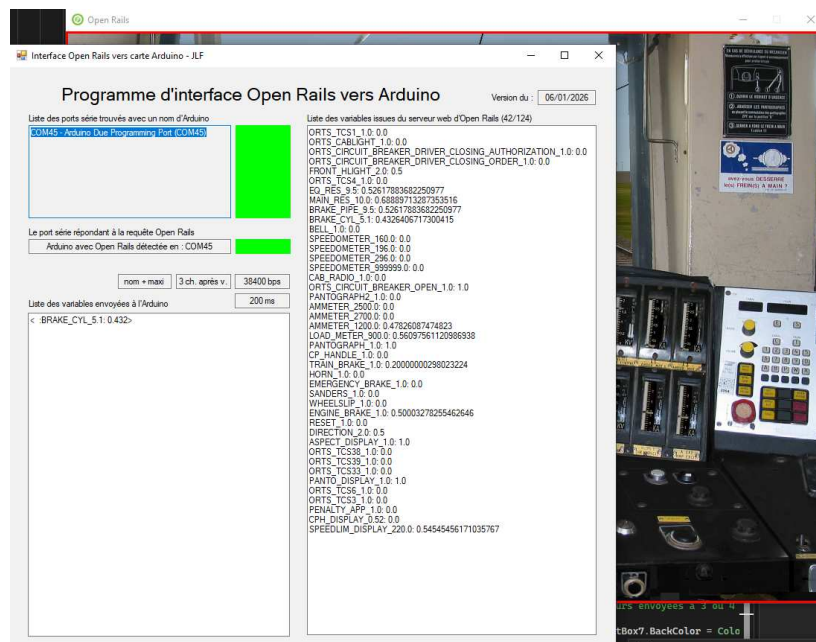
Voulant tester mon système avec un autre simulateur de train sur Windows, j'ai réalisé ce programme d'interface "InterfaceORArduinoJLF" pour "Open Rails".

Mon programme récupère les données d'Open Rails, et les envoie à ma carte Arduino sous le même format que "Train Simulator Classic" avec "TSClassic Raildriver and Joystick Interface".

Ce format de trame est le suivant : < Type de données : Nom de la donnée : Valeur de la donnée >

Le tout au format texte, lisible. Exemples : <:VITESSE:0.441> <:FREIN\_CG:0.83> <:PHARES:1.0>

Aspect du programme "InterfaceORArduinoJLF" :



Le programme sur Windows, Open Rails mets à disposition ses données sur son serveur web. Dans sa notice, Open Rails préconise un accès à sa page web toutes les 500 msec ou plus.

J'ai optimisé ce programme pour être rapide et économe en cpu. Sur un ordinateur avec un "i5", il consomme moins de 0,5 % de cpu.

Avec Open Rails et ce programme en marche, si l'on prend un taux de rafraichissement de :

- 50 msec (20 fois par secondes), on consomme 8 % de cpu en plus (Ce programme + Open Rail).
- 100 msec (10 fois par secondes), on consomme 4 % de cpu en plus (Ce programme + Open Rail).

La majorité du cpu utilisé vient d'Open Rails.

Une fois Open Rails de lancé, avec un navigateur internet, on peut afficher ces données sur la page : localhost:2150/API/CABCONTROLS

On a alors ce texte d'affiché :

```
[ { "TypeName": "ORTS_TCS1", "MinValue": 0.0, "MaxValue": 1.0, "RangeFraction": 0.0 }, { "TypeName": "ORTS_CABLIGHT", "MinValue": 0.0, "MaxValue": 1.0, "RangeFraction": 0.0 }, { "TypeName": "ORTS_CIRCUIT_BREAKER_DRIVER_CLOSING", "MinValue": 0.0, "MaxValue": 1.0, "RangeFraction": 0.0 },  
/ .....  
{ "TypeName": "PENALTY_APP", "MinValue": 0.0, "MaxValue": 1.0, "RangeFraction": 0.0 }, { "TypeName": "CPH_DISPLAY", "MinValue": 0.0, "MaxValue": 0.52, "RangeFraction": 0.0 }, { "TypeName": "SPEEDLIM_DISPLAY", "MinValue": 0.0, "MaxValue": 220.0, "RangeFraction": 0.54545456171035767 } ]
```

Il existe déjà un programme d'interface de ce genre : "FS-ORTS-Link-main.zip", disponible ici :

<https://github.com/ferrovisim/FS-ORTS-Link>

J'ai réalisé mon programme d'interface, pour apporte le moins de modification possible pour utiliser directement ma carte Arduino : [https://www.la-tour.info/uts/uts\\_page15.html#conduite](https://www.la-tour.info/uts/uts_page15.html#conduite) avec Open Rails.

J'ai testé mon programme d'interface avec une requête web toutes les 50 msec, sans souci. J'ai bien les données rafraichies à un taux de 50 msec sur ma carte Arduino DUE.

Comme la recherche de données sur la page web se fait très fréquemment, finalement il n'y qu'une ou deux variables au maximum, à transmettre à chaque fois.

A 50 ou 100 msec de taux de rafraichissement, les servomoteurs sont déjà plus fluides.

Les liens.

Mon site : [https://www.la-tour.info/uts/uts\\_page15.html#conduite](https://www.la-tour.info/uts/uts_page15.html#conduite)

Open Rails : <https://www.openrails.org/>

Le programme d'interface "TSClasic Raildriver and Joystick Interface" pour "Train Simulator Classic RailWorks de DTG" : <https://forums.dovetailgames.com/threads/ts-classic-raildriver-and-joystick-interface.72488/>

Le site d'animation d'un pupitre avec Open Rails : <https://ferrovisim.fr/>

Le forum RMF : <https://www.rmfmagazine.com/phpBB/viewforum.php?f=21>

Le forum RAILSIM : <https://www.railsim-fr.com/forum/index.php?/forum/1-train-simulator-202x-ts-classic/>

Le programme est écrit en C#. Je l'ai écrit dans l'environnement gratuit, C# Visual Studio 2026 de Microsoft, version "Community". <https://visualstudio.microsoft.com/fr/downloads>

# 1 / Le fichier de configuration.

La fenêtre Windows du programme ne sert qu'à afficher des données. Le paramétrage de l'application se fait dans le fichier "**InterfaceORArduinoJLF\_config.txt**".

Quand le programme se lance, il va lire le fichier de configuration "**InterfaceORArduinoJLF\_config.txt**".

Ce fichier est placé dans le même répertoire que le programme.

Un fichier déjà renseigné est fourni en exemple avec le programme.

## Contenu du fichier "**InterfaceORArduinoJLF\_config.txt**" :

Les lignes de commentaires commencent par une '\*'. Exemple :

```
* Fichier InterfaceORArduinoJLF_config.txt                                Le 15/01/2026
* Fichier de configuration pour le programme : Interface Open Rails vers carte Arduino - JLF.exe
* -----
```

On peut imposer un port série, ou laisser le programme trouver tout seul la carte Arduino.

Si l'on a configuré la carte Arduino, pour lorsque l'on lui envoie "<:JLF?:0>", elle répond "<JLF!>", le programme la détectera tout seul.

Pour que le programme trouve la carte Arduino automatiquement, utiliser le mot "auto".

Dans les autres cas, pour définir un port série fixe, le nommer en clair. Exemples :

```
port = auto
port = COM45
```

Il faut donner la vitesse du port série vers l'Arduino. C'est le débit en baud de la liaison avec la carte Arduino Open Rails. Les vitesses possibles normalisées sont = 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 230400 ou 460800 bps.

La vitesse de 38400 bps est un très bon compromis, entre la vitesse d'envoi d'une trame sur la liaison série (7 msec en moyenne), et le temps que met l'Arduino à traiter cette trame (9 msec en moyenne pour un servomoteur).

Pour définir la vitesse, exemple :

```
vitesse = 38400
```

Les valeurs numériques disponibles sur la page web d'Open Rails, sont sous la forme : 0,52617883682250977.

On note 17 chiffres après la virgule. Si la valeur bouge un tout petit peu (0,00001%), ça ne sert à rien de l'envoyer à la carte Arduino.

Le programme fait alors deux choses. Il limite l'envoi de la valeur à n chiffres après la virgule, et il n'envoiera cette valeur que lorsque ce nombre tronqué sera modifié.

Ca limite ainsi le nombre de trames envoyées sur la ligne série.

On peut choisir d'envoyer de 2 à 8 chiffres après la virgule.

Si l'on choisi 2 chiffres après la virgule, on enverra "0,51" ce qui donne une précision de 1 %, (0,00 à 0,99).

Si l'on choisi 3 chiffres après la virgule, on enverra "0,514" ce qui donne une précision de 0,1 %.

Si l'on choisi 4 chiffres après la virgule, on enverra "0,5148" ce qui donne une précision de 0,01 %.

Par défaut utiliser '3', ce qui donne une précision suffisante pour les instruments du pupitre.

Pour définir la précision, exemple :

```
precision = 3
```

Le programme cherche à intervalle régulier, les données sur le port web d'Open Rails.

Dans la notice d'Open Rails, il est conseillé de consulter la page web, pas plus de 2 fois par seconde, soit un intervalle de 500 msec.

Avec mon programme, on peut définir un intervalle compris entre 50 et 5000 msec. J'ai testé à 50 msec et ça fonctionne très bien. Cet intervalle ne pose pas de problème à un Arduino DUE, qui traite les données au fil de l'eau très rapidement. Aucune saturation n'est possible.

Ce délai de 50 msec, à l'avantage de rendre le mouvement des servomoteurs plus fluide, et donc des manomètres.

Essayer 100 msec puis 50 msec. Pour définir ce délai, exemple :

```
timer = 50
```

Les variables disponibles sur la page web d'Open Rails, ont souvent leurs noms dupliqués. On se retrouve avec trois ou quatre noms identiques à la suite.

On peut corriger cela en partie, en reprenant le fichier "Locomotive.cvf", et en différenciant les noms des variables.

Pour faciliter, l'exploitation des noms des variables, le programme peut aussi ajouter automatiquement la valeur "MaxValue", à la fin du nom envoyé.

Pour plusieurs variables " SPEEDOMETER ", le programme peut envoyer "SPEEDOMETER\_160" si avec MaxValue = 160. Ca améliore la différenciation des variables au nom identique.

Pour activer cette fonction, mettre "oui", sinon mettre "non". Exemple :

```
nomavecmaxi = oui
```

Ce programme prend 2 % du cpu et Open Rails 6 % de cpu, pour un timer à 50 milli secondes.

La moitié du cpu utilisée par mon programme d'interface, sert à l'affichage des variables sur la fenêtre Windows.

Si l'affichage des variables est utile pour la mise au point, une fois en fonctionnement nominal, on peut réduire par deux le taux de cpu utilisé par mon programme.

Je n'ai pas trouvé la fonctionnalité, Visual Studio, pour éviter d'afficher ces variables quand la fenêtre n'est pas visible.

Ce paramétrage est surtout sensible quand le timer est faible (50 ou 100 msec).

Si l'on indique "oui", le programme d'affichera plus les variables au bout de 1 mn, pour réduire le taux de cpu utilisé.

Pour activer cette fonction, mettre "oui", sinon mettre "non". Exemple :

```
optimisation = oui
```

Bien respecter la mise en forme, avec un espace avant et après le signe égal.

Utiliser les majuscules pour "COM", et les minuscules pour le reste.

En cas d'erreur, les cases seront en jaune sur la fenêtre du programme.

## 2 / Démarrage du programme

Mettre au même emplacement, les deux fichiers :

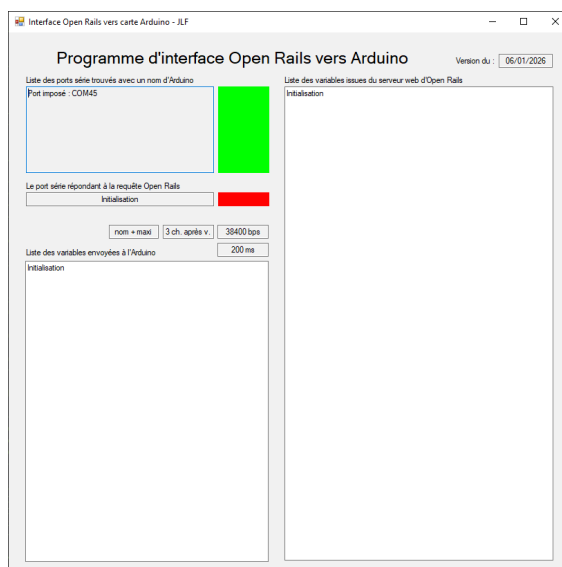
"InterfaceORArduinoJLF.txt" et "InterfaceORArduinoJLF\_config.txt".

Penser à vérifier les mises à jour sur le site : [https://www.la-tour.info/uts/uts\\_page15.html](https://www.la-tour.info/uts/uts_page15.html)

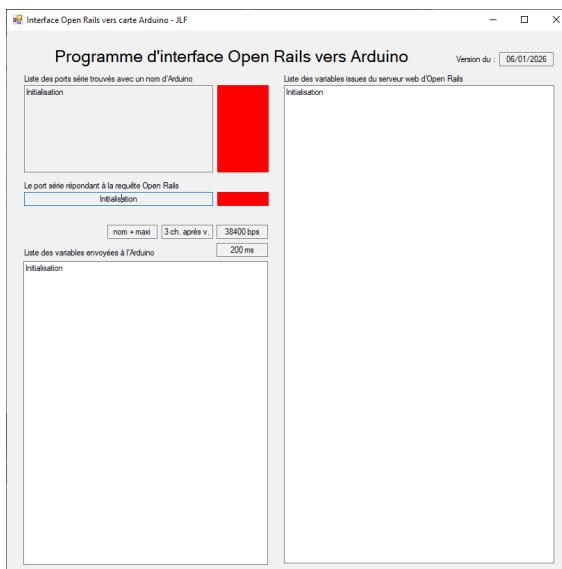
Lancer "InterfaceORArduinoJLF.exe".

Il n'y a pas de boutons ni de menu, la configuration se fait dans le fichier "InterfaceORArduinoJLF\_config.txt".

Si l'on a configuré un port série fixe imposé, il apparaît seul dans la liste des ports existants.



Si l'on a choisi "auto", le programme affiche les ports existants, marqués "Arduino".

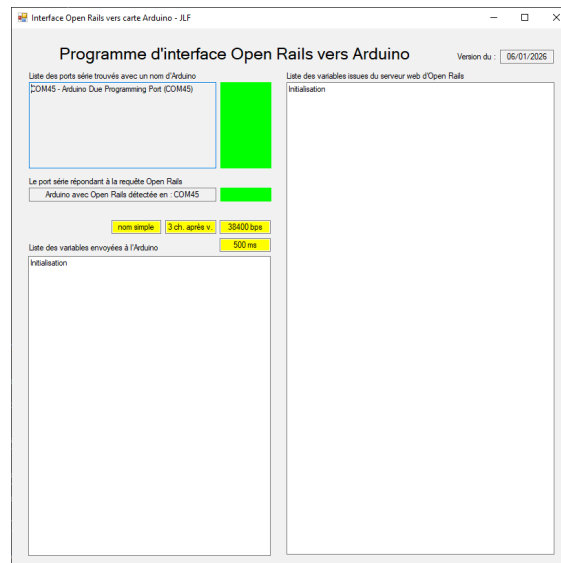


La couleur du gros rectangle, indique si le programme a détecté au moins un port "Arduino".

La couleur du petit rectangle, indique si le programme a détecté au moins un port "Arduino" pour Open Rails.

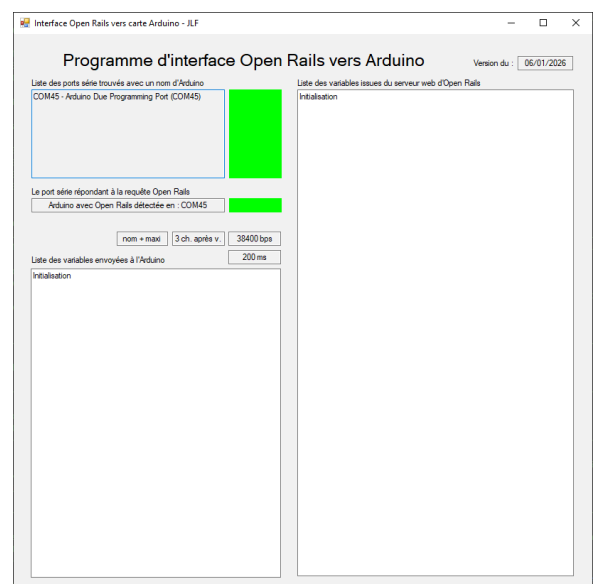
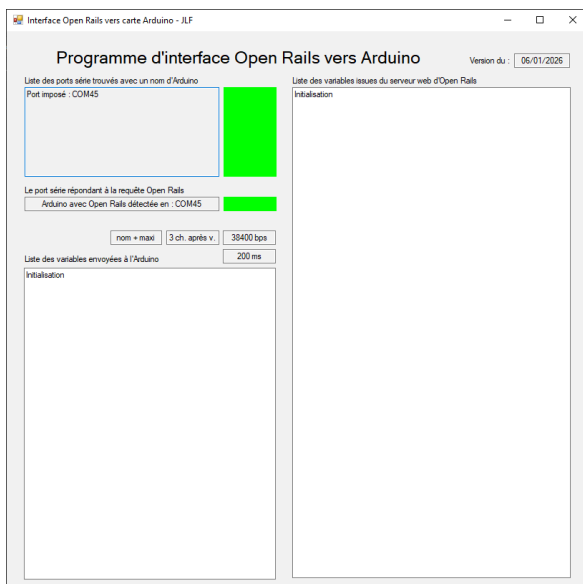
Si les paramètres du fichier de configuration ne sont pas lisibles, ou hors limites, le programme affiche des rectangles jaunes.

Dans ce cas, le programme utilisera les paramètres par défaut.



Ensuite, dans un premier temps, le programme cherche un port série avec une carte Arduino, toutes les 5 secondes. En mode automatique, le programme envoie "<:JLF?:0>", et retient la carte qui répond "<JLF!>". En mode port fixe imposé, si le programme peut ouvrir le port, celui-ci passe directement au vert.

Quand les deux rectangles sont verts, la connexion à la carte Arduino est établie.



Si le port Arduino est occupé, par exemple par l'IDE Arduino, la connexion restera rouge.

Une fois que la connexion à l'Arduino est établie, le programme va chercher les données sur la page web d'Open Rails.

Le programme va chercher à contacter la page web, toutes les 5 secondes. Une fois le contact établi, le programme ira chercher les données toutes les xxx msec.

Dés que la page web répond, il affiche la liste des variables reçues.

On affiche dans le titre, le nombre de variables lues (124) et le nombre de variables retenues (42) une fois les doublons supprimés. Le maximum est de 200 variables lues et 100 variables retenues.

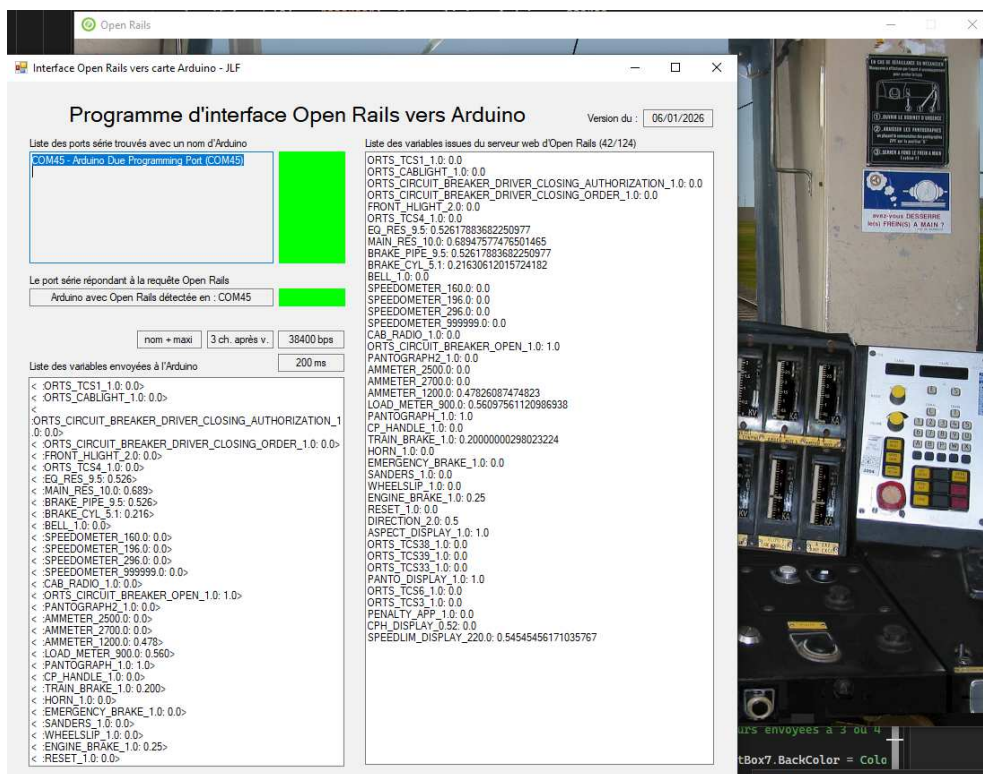
On note :

[ nom + maxi ] = Envoi des noms de variables accolés à la valeur maximum. Exemple : SPEEDOMETER\_160

[ 3 ch après v. ] = Envoi de la valeur avec 3 chiffres après la virgule.

[ 38400 bps ] = Vitesse de la liaison série vers l'Arduino.

[ 200 ms ] = On va chercher la page web toutes les 200 milli secondes.



Si dans le fichier de configuration, on a spécifié "optimisation = oui", au bout d'une minute, on ne rafraichira plus les listes de variables à l'écran.

Ca économise du cpu. Dans ce cas sur l'écran, il sera indiqué "\*\*\* OPTIMISATION - LISTE FIGEE \*\*\*" à la fin des listes des variables.

A chaque accès à la page web d'Open Rails, toutes les xxx msec, le programme affiche la liste de toutes les variables avec leurs valeurs (A droite).

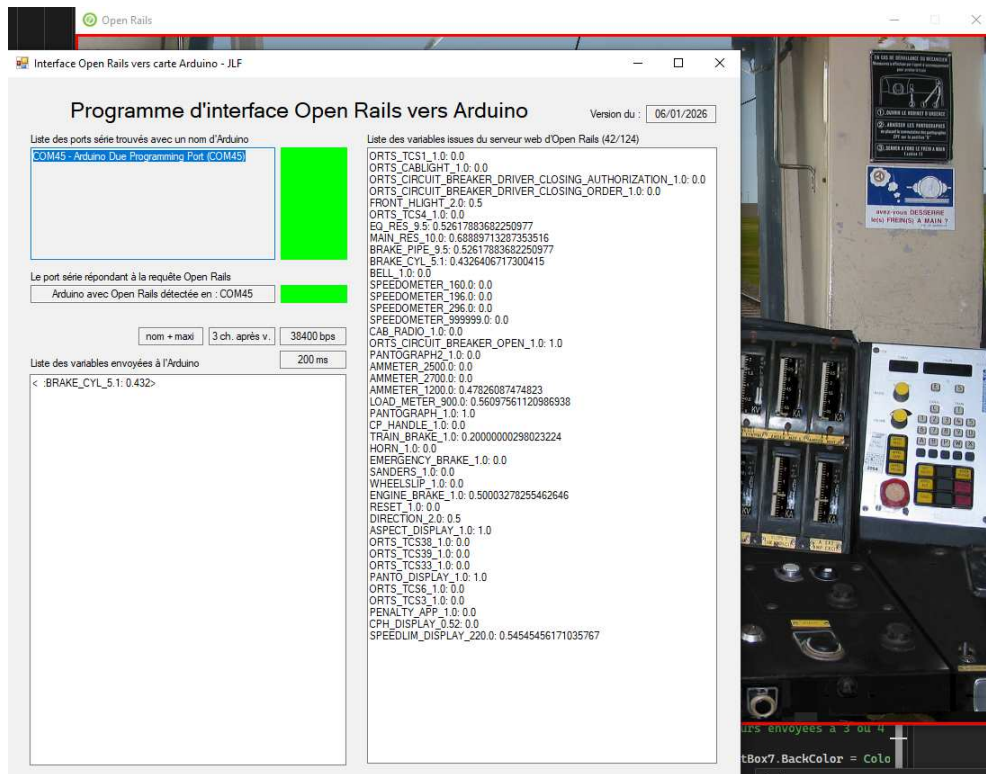
Par contre les noms dans cette liste n'évolueront pas. On gardera à droite, toujours ces mêmes noms de variables.

Le programme compare avec les anciennes valeurs, et en cas d'évolution les enverra à la carte Arduino.

La liste des variables envoyées (A gauche), apparait quand les données évoluent et sont envoyées.

Les données envoyées restent visibles, jusqu'à ce que le rafraichissement de la page web donne de nouvelles valeurs.

Ici, on a activé le frein (A gauche) :



Si l'on a configuré 3 chiffres après la virgule, la variable sera envoyée, si elle passe par exemple de 0,123 à 0,124.

Eviter d'utiliser plus de 4 chiffres après la virgule, car certaines variables sont instables, et si elles changent souvent de valeurs, elles seront transmises à chaque fois à l'Arduino, alors que l'Arduino ne sera pas capable de déplacer une aiguille du cadran pour 0,01% de changement.

En pratique utiliser 3 chiffres après la virgule, pour avoir suffisamment de précision (0,1 %). On peut afficher 378, 379, 380 km/h sans soucis.

On ne met pas à l'échelle la valeur envoyée, car de toute façon elle sera mise à l'échelle sur l'Arduino, par un calcul en flottant.

Cela fait un petit gain de cpu sur l'ordinateur. Du côté de l'Arduino, le calcul en flottant, n'est fait que sur une variable reçue, donc une fois toutes les 50 msec au maximum.



Mon programme envoie donc une valeur comprise en 0.0 et 1.0. ca peut être une valeur comme 0.123.

On retrouve ces valeurs sur la page web d'Open Rails. Une fois Open Rails de lancé, avec un navigateur internet, afficher les données sur la page : localhost:2150/API/CABCONTROLS, on affiche ce texte :

```
{ "TypeName": "BRAKE_PIPE", "MinValue": 0.0, "MaxValue": 9.5, "RangeFraction": 0.526178836 },...{
```

On trouve ici, les coefficients minimum et maximum, pour calculer la valeur finale dans l'Arduino.

Pour calculer la valeur attendue sur l'Arduino, utiliser la formule  $\text{MinValue} + (\text{MaxValue} - \text{MinValue}) * \text{RangeFraction}$ .

On peut aussi procéder de la manière suivante, par exemple pour le manomètre qui affiche de 0 à 9,5 Bars.

On a déterminé au préalable, que le servomoteur affiche 0 Bar pour 1118 µsec, et 9,5 Bars pour 2017 µsec.

Sur ma carte Arduino, ça donne le code suivant pour avoir la valeur en µsec :

```
// < :BRAKE_PIPE:5.50> ==> Sortie servo 0 - Aiguille CG
if (!strcmp(buffer_rx[1], "RE_control")) { // Affichage 9,5 Bars maximum sur cadran 10 Bars.
    buffer_rx_fvaleur = atof(buffer_rx[2]); // Conversion du dernier champ : Ascii-> Flottant.
    buffer_rx_fvaleur = 899*buffer_rx_fvaleur; // Coefficient = (2017-1118) = 899
    buffer_rx_valeur = 1118 + int(buffer_rx_fvaleur); //
    buffer_rx_valeur = min(buffer_rx_valeur, 2400); // Valeur < 2400 µsec.
    i2c_srv_val[0] = max(buffer_rx_valeur, 600); // Valeur > 600 µsec.
    bitWrite(i2c_srv_maj, 0, 1);
}
```

Quand la carte Arduino reçoit < :BRAKE\_PIPE:0.000> le servomoteur est positionné sur 1118 µsec.

Quand la carte Arduino reçoit < :BRAKE\_PIPE:1.000> le servomoteur est positionné sur 2017 µsec.

On n'a qu'un seul calcul flottant à faire sur l'Arduino.

Sur la page web d'Open Rails, on retrouve souvent 3 ou 4 fois le même nom de variable. Le programme ne conservera qu'une seule variable à envoyer. C'est pour cela que le titre au dessus de la liste des variables affiche (42/124).

Pour éviter cette pagaille, on peut configurer "nomavecmaxi = oui" dans le fichier de configuration. Dans ce cas le programme ajoute la valeur "MaxValue" à la fin du nom de variable.

Exemple : "SPEEDOMETER\_160", avec MaxValue = 160 à la fin du nom. Ca améliore la différenciation des variables au nom identique.

On peut aussi reprendre le fichier "Locomotive.cvf", en différenciant les noms des variables.

Mettre le timer à 50 msec, fonctionne bien chez moi. J'utilise un Arduino DUE (32 Bits à 84 Mhz), alors qu'une carte MEGA 2560 (8 bits à 16 Mhz) sera beaucoup moins puissante.

### 3 / Les données envoyées à la carte Arduino

Les données sont envoyées à la carte Arduino, suivant le format : <champ1:champ2:champ3>

< = Caractère de début d'envoi de variable.

champ1 = Type de champ, LEV = LEVIER, INT = INTERRUPTEUR. Ici cela ne sert pas.

: = Délimiteur de champ

champ2 = Nom de la variable = BELL\_1, MANOMETRE\_9, SPEED...

: = Délimiteur de champ

champ3 = Valeur, pour le moment je n'ai que trouvé que des valeurs numériques = 0.1, 1.0, 0.184788741.

> = Caractère de fin d'envoi de variable.

Exemple d'envoi : <:EQ\_RES\_9:0.441><:BRAKE\_PIPE\_9:0.441><:TRAIN\_BRAKE\_1:0.200>

La valeur numérique est comprise entre 0.00 et 1.00.

Vous pouvez modifier le code source du programme, pour changer cette méthode de traitement des données.

Si une ou plusieurs données ont changé de valeur, depuis le dernier accès à la page web, elles sont envoyées à l'Arduino.

#### Exemples d'envoi de trame :

Pour envoyer <:ORTS\_CIRCUIT\_BREAKER\_DRIVER\_CLOSING\_AUTHORIZATION:0.0>, 60 caractères à 38400 bps, ça prend 16 msec.

Pour envoyer <:SPEEDOMETER\_160:0.123>, 24 caractères, ça prend 6,2 msec.

Pour envoyer <:BELL\_1:0.0>, 14 caractères, ça prend 3,6 msec.

J'utilise une carte Arduino DUE, pour gérer un poste de conduite : [https://www.la-tour.info/uts/uts\\_page15.html#conduite](https://www.la-tour.info/uts/uts_page15.html#conduite)

Après l'arrivée d'une trame de commande, la carte Arduino DUE met 1 msec pour traiter une valeur qui a changé.

Cela prend 1 msec, c'est surtout le temps d'envoyer l'information aux cartes de sorties sur le bus I2C.

Mais il faut attendre la fin du cycle PWM, pour mettre effectivement ces sorties à jour. La carte Servomoteur + Sorties PWM tourne à 62 Hz et les deux cartes de sortie tout ou rien à 1 KHz.

Ca prend donc 16 msec supplémentaires pour les manomètres et galvanomètres, et 1 msec pour les voyants.

Globalement, le montage (Arduino + carte I2C PWM) mettra :

2 msec pour changer l'état d'un sortie tout ou rien.

17 msec pour changer l'état d'un sortie PWM ou Servomoteur.

Si l'on change simultanément l'état de 5 servomoteurs ou sorties PWM, ça ne prendra que 5 + 16 msec, soit 21 msec.

Finalement la vitesse de la liaison à 38400 bps est optimale. Avec un temps moyen d'envoi d'une commande à 6 msec, la carte Arduino peut gérer la file de commandes qui arrive, puisque cela prend 1 msec de cpu par commande reçue.

Globalement, le programme sur ordinateur n'envoie qu'une ou deux variables à la fois. C'est donc assez rapide, et cela ne pose pas de soucis à réduire le timer à 100 msec, voir 50 msec sur l'ordinateur.

Au démarrage, quand le programme envoie toutes les variables (*43 Pour une BB22000*) ça représente 1200 caractères, soit 310 msec. Le code de la carte Arduino DUE est adapté à ce pic de flux.

Et sur l'ordinateur, si la tâche d'envoi n'a pas fini son travail en 50 msec, elle continue d'envoyer ses données. L'ordinateur la réactivera dans 50 msec.

## 4 / Le programme source

Le programme est écrit en C#. Maitrisant mieux le C++, je n'ai pas trop utilisé les classes d'objets.

Je l'ai écrit dans l'environnement gratuit, C# Visual Studio 2026 de Microsoft, version "Community".

<https://visualstudio.microsoft.com/fr/downloads>

La solution complète se trouve dans le fichier : "**InterfaceORArduinoJLF.zip**".

Pour lancer Visual Studio, cliquer sur "**InterfaceORArduinoJLF.csproj**".

Le programme principal est : "Program.cs".

La routine principale est : "Form1.cs".

La fenêtre est décrite dans les fichiers : "Form1.Designer.cs" et "Form1.resx".

Le fichier de configuration lu au lancement du programme est : "bin/InterfaceORArduinoJLF\_config.txt".

Pour animer la fenêtre, j'utilise les images : rectangle\_74x20\_rouge.png, rectangle\_74x20\_vert.png, rectangle\_74x125\_rouge.png et rectangle\_74x125\_vert.png.

Le code source de "Form1.cs" est découpé en plusieurs parties.

### 1 /timer1\_Tick(object sender, EventArgs e)

Tous les ticks du timer on passe ici.

Si la connexion avec l'Arduino n'est pas réalisée, on fait appel à "DemandeDeConnexion\_Arduino()".

En mode établi, on ne teste plus la liaison avec la carte Arduino. Ce n'est que lorsqu'une donnée change et qu'on doit l'envoyer, que l'on teste la liaison.

### 2 / DemandeDeConnexion\_Arduino()

Effectue une demande de connexion à la carte Arduino.

### 3 / DemandeDeConnexion\_WebOpenRail()

Va lire la page web d'Open Rails.

### 4 / DemandeDeLecture\_FichierConfiguration()

Lit une seul fois le fichier de configuration, au lancement du programme.

Les variables principales :

**port\_serie\_or\_trouve\_flag =**

0 : On n'a pas trouvé de port série marqué "Arduino".

1 : On a trouvé des port série marqué "Arduino" avec ou sans Open Rail (*En mode automatique*).

2 : On a trouvé la carte Arduino (En mode COM45), ou la carte Arduino Open Rails (*En mode automatique*).

#### **probleme\_ecriture\_port\_flag =**

0 : Ok.

1 : Dans la routine d'accès à la page web, impossible d'envoyer les données sur le port série Arduino.

#### **probleme\_lecture\_web\_flag =**

0 : Ok.

1 : Dans la routine d'accès à la page web, impossible d'accéder à la page web, ou page web inexploitable.

En cas de problème, le timer principal repasse de xxx msec à 5 secondes.

On recherche une carte Arduino toutes les 5 secondes.

Puis on tentera d'accéder à la page web toutes les 5 secondes.

Dans le programme, timer = const int timer\_trouve\_port\_OR\_Arduino = 5000;

Il faut d'abord avoir une liaison établie avec la carte Arduino, pour que le programme fasse un accès à la page web.

Le nombre de variables pouvant être reçues et pouvant être envoyées est fixé ici :

const int nb\_max\_variable = 200;                      Nombre de variables sur la page web.

const int variable\_envoi\_nb\_max = 100;            Nombre de variables au nom unique, envoyées.

Pour changer la méthode d'en capsulage des données, il faut modifier ces lignes de code :

```
if (variable_envoi_valeur[cpt4].Length > variable_precision) // On limite à x chiffres après la virgule.
{ texte_variable_a_envoyer = "< :" + variable_envoi_nom[cpt4] + variable_envoi_valeur[cpt4].Substring(0, variable_precision)
+ ">"; }
else
{ texte_variable_a_envoyer = "< :" + variable_envoi_nom[cpt4] + variable_envoi_valeur[cpt4] + ">"; }
variable_envoi_a_envoyer[cpt4] = 0;
texte_liste_variables_a_envoyer_a_afficher += texte_variable_a_envoyer + "\r\n"; // Affiche de la liste des variables
envoyées, sur la fenêtre Windows,
// texte_variable_a_envoyer = < :EQ_RES_9.5: 0.441>< :BRAKE_PIPE_9.5: 0.441>< :TRAIN_BRAKE_1.0: 0.200>
try { serialPort.Write(texte_variable_a_envoyer); }
```

En fonctionnement établi, si un problème se pose, le programme tentera tout seul de rétablir la situation toutes les 5 secondes.

## 5 / Exemple de données envoyées à l'Arduino

Avec une BB22000, on obtient la liste de 42 variables suivantes :

```
<:ORTS_TCS1_1:0.0>
<:ORTS_CABLIGHT_1:0.0>
<:ORTS_CIRCUIT_BREAKER_DRIVER_CLOSING_AUTHORIZATION_1:0.0>
<:ORTS_CIRCUIT_BREAKER_DRIVER_CLOSING_ORDER_1:0.0>
<:FRONT_HLIGHT_2:0.0>
<:ORTS_TCS4_1:0.0>
<:EQ_RES_9:0.526>
<:MAIN_RES_10:0.689>
<:BRAKE_PIPE_9:0.526>
<:BRAKE_CYL_5:0.216>
<:BELL_1:0.0>
<:SPEEDOMETER_160:0.0>
<:SPEEDOMETER_196:0.0>
<:SPEEDOMETER_296:0.0>
<:SPEEDOMETER_999999:0.0>
<:CAB_RADIO_1:0.0>
<:ORTS_CIRCUIT_BREAKER_OPEN_1:1.0>
<:PANTOGRAPH2_1:0.0>
<:AMMETER_2500:0.0>
<:AMMETER_2700:0.0>
<:AMMETER_1200:0.478>
<:LOAD_METER_900:0.560>
<:PANTOGRAPH_1:1.0>
<:CP_HANDLE_1:0.0>
<:TRAIN_BRAKE_1:0.200>
<:HORN_1:0.0>
<:EMERGENCY_BRAKE_1:0.0>
<:SANDERS_1:0.0>
<:WHEELSLIP_1:0.0>
<:ENGINE_BRAKE_1:0.25>
<:RESET_1:0.0>
<:DIRECTION_2:0.5>
<:ASPECT_DISPLAY_1:1.0>
<:ORTS_TCS38_1:0.0>
<:ORTS_TCS39_1:0.0>
<:ORTS_TCS33_1:0.0>
<:PANTO_DISPLAY_1:1.0>
<:ORTS_TCS6_1:0.0>
<:ORTS_TCS3_1:0.0>
<:PENALTY_APP_1:0.0>
<:CPH_DISPLAY_0:0.0>
<:SPEEDLIM_DISPLAY_220:0.545>
```

Avec 3 chiffres après la virgule, et le nom de variable composé avec l'ajout de la valeur maximum.

Les champs sont délimités par le caractère ':':

Le premier champ est le type de variable, interrupteur, gauge,,, mais ce n'est pas utilisé ici.

## 6 / Le code de réception sur une carte Arduino DUE

Le code complet se trouve ici : [https://www.la-tour.info/uts/uts\\_page15.html#conduite](https://www.la-tour.info/uts/uts_page15.html#conduite)

Le code de réception des trames est robuste. Il sait traiter de trames erronées, sans planter.

```
// Variables globales pour les sous-programmes : 'Data_Recues_du_PC_TSCRJI.ino' et
'Data_Envoyees_vers_PC_TSCRJI'.
byte buffer_rx_cpt;           // Compteur de n° du caractère courant.
byte buffer_rx_buf;          // Compteur de n° du buffer courant.
char car_lu;                 // Caractère lu.
char buffer_rx[3][80];       // 3 Buffers de réception des caractères reçus.
int buffer_rx_valeur;        // Conversion du dernier champ : Ascii-> Entier.
float buffer_rx_fvaleur;     // Conversion du dernier champ : Ascii-> Flottant.
boolean debut_rx;            // Flag de début de réception, true = indicateur de début de trame reçu.
boolean final_rx;            // Flag de fin réception, true = indicateur de fin de trame reçu.
/* Cette version est pour 'TS Classic Raildriver and Joystick Interface' avec 'RailsWorks Classic'.
   On reçoit sur la liaison série, ce type de trame : <Nom du groupe de données:Nom de la donnée:Valeur>
   qu'il faut répartir en 3 chaines de caractères (80 max) :
       - buffer_rx[0] = Nom du groupe de données (Texte).
       - buffer_rx[1] = Nom de la donnée (Texte).
       - buffer_rx[2] = Valeur au format texte (Nombre entier ou avec virgule).

   A chaque passage, on dépile tous les caractères reçus entre temps.
   On ne fournira une action à réaliser, que lorsque l'on aura reçu une commande complète.
   Indicateurs reçus : Début de trame = '<' Séparation de champs début de trame = ':' Fin de trame = '>'
*/

while(( x = Serial.available()) > 0) {
    char car_lu = Serial.read();           // La fonction 'Serial.read()' retourne un entier !
    'car_lu' est déclaré en char.
    if(car_lu == '<') {                     // On a reçu l'indicateur de début de trame '<'.
        debut_rx = true;                   // D'office, on initialise tout.
        final_rx = false;                  // indicateur de début de trame reçue = true, de fin
de trame = false.
        buffer_rx_buf = 0;                 // Initialise compteur de champ et de caractères.
        buffer_rx_cpt = 0;                 //
        buffer_rx[0][0] = '\0';            // Initialise les chaines de caractères.
        buffer_rx[1][0] = '\0';            //
        buffer_rx[2][0] = '\0';            //
    }
    else if(car_lu == '>') {                 // On a reçu l'indicateur de fin de trame '>'.
        if ((debut_rx == true) && (buffer_rx_buf == 2)) { //
            buffer_rx[buffer_rx_buf][buffer_rx_cpt] = '\0'; // On écrit le caractère 'Null' pour finir la chaine
de caractères.
            final_rx = true;                 // On a bien reçu les trois champs, on valide la
réception de fin de trame.
            break;                           // On sort de la boucle pour traiter cette trame
complète, avant de traiter au prochain passage les autres caractères.
        }
        else { debut_rx = false; }          // On a reçu l'indicateur de fin de trame, sans
recevoir celui de début de trame et les 3 champs. On laisse tomber.
    }
    else if(car_lu == ':') {                 // On a reçu l'indicateur de changement de champ de
données ':'.
        if (debut_rx == true) {
```

```

    buffer_rx[buffer_rx_buf][buffer_rx_cpt] = '\0'; // On écrit le caractère 'Null' pour finir la chaine
de caractères.
    buffer_rx_buf++;
    buffer_rx_cpt = 0;
    if (buffer_rx_buf >= 3) { debut_rx = false; } // On vient de lire un troisième indicateur de
changement de champ ': ', il y a une erreur !
    } // On laisse tomber.
}
else if (debut_rx == true) { // On a déjà reçu l'indicateur de début de trame!
    buffer_rx[buffer_rx_buf][buffer_rx_cpt] = car_lu; // On a reçu un caractère qui n'est pas un
indicateur, on le stocke.
    buffer_rx_cpt++;
    if (buffer_rx_cpt > 78) { debut_rx = false; } // On vient de lire 78 caractères sans indicateur de
fin de trame ! On laisse tomber car le buffer est dimensionné à 80 caractères.
    } // On n'a pas encore reçu l'indicateur de fin de
trame. Donc, on jette le caractère reçu.
    }
// On a vidé le buffer de réception, ou l'on a reçu une trame complète.

// -----
    if ((debut_rx == true) && (final_rx==true)) {
        debut_rx = false; final_rx = false;
/* On a reçu une trame de commande.
    - buffer_rx[0] = Nom du groupe de données (Texte).
    - buffer_rx[1] = Nom de la donnée (Texte).
    - buffer_rx[2] = Valeur au format texte (Nombre entier ou avec virgule).
Suivant l'utilisation, on convertira cette dernière chaine de caractère, en entier ou en flottant.
buffer_rx_valeur = atoi(buffer_rx[2]); // Conversion du dernier champ : Ascii-> Entier. Pas de contrôle !
Si Nok valeur = 0!
buffer_rx_fvaleur = atof(buffer_rx[2]); // Conversion du dernier champ : Ascii-> Flottant. Pas de contrôle !
Si Nok valeur = 0!
*/

// Et après on traite les trames reçues
if (!strcmp(buffer_rx[1], "RE_control")) { . . . . }
else if (!strcmp(buffer_rx[1], "CP_control")) { . . . . }
// On récupère la valeur : buffer_rx_fvaleur = atof(buffer_rx[2]); // Conversion du dernier champ : Ascii->
Flottant.

```

Pour plus d'information, voir le code Arduino pour la CC72000 sur mon site.

A+