

15/07/2026

Le jeu TSC met à disposition ses données, via une librairie "RailDriver64.dll".

Le 13/07/2026 : Version 5.0. Exécutable et sources C# 64 bits disponibles.

Il envoie aussi les données de la carte Arduino branché sur le pupitre (Position des interrupteurs, valeurs analogique, info du KVB,...) vers TSC.



[Sur ce site](#), je présente un système global sur une carte Arduino DUE, pour gérer un véritable pupitre de locomotive.

J'ai longtemps utilisé le programme "TS Classic Raildriver and Joystick Interface V3.3.0.9" pour recevoir les informations du simulateur TSC tournant sous Windows.

Mais ce programme ne permet pas l'envoi de commande depuis le pupitre, vers le jeu. On est obligé de passer par une émulation de clavier, pour envoyer le code des touches correspondant aux commandes activées. Avec une

BB7200, on ne peut pas envoyer directement une valeur analogique, comme le régulateur de traction, ce qui n'est pas pratique.

Avec la BB7200 de SimExpress, on ne peut pas activer les interrupteurs SAL (Alerte) et Surcharge.

On ne peut pas envoyer au simulateur TSC, l'activation des boutons [MV] et [FC] du KVB et la valeur des roues codeuses.

Pour échanger les données, dans les deux sens de communication, j'ai réalisé ce programme d'interface **"InterfaceTSCArduinoJLF"** en C#.

Le programme sur Windows, TSC permet un échange de données, via une librairie "RailDriver64.dll".

J'ai utilisé en partie le code posté ici : RailSim-Remote Serveur web <https://github.com/YoRyan/railsim-remote>.

J'ai remplacé l'échange via un serveur web, par une liaison série.

Mon programme récupère les données de TSC, et les envoie à ma carte Arduino sous le même format que "Train Simulator Classic" avec l'interface "TSCClassic Raildriver and Joystick Interface V3.3.0.9".

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

Je n'utilise pas le champ "Type de données", qui reste vide.

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

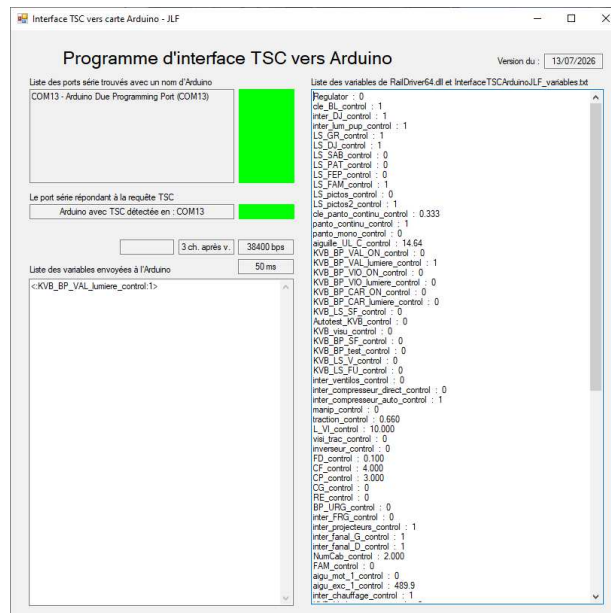
Je commence le numéro de version à V5, pour facilement repérer que les versions V5, compatibles entre-elle (Poste de conduite, KVB, Interface).

Il existe maintenant deux solutions, pour équiper un pupitre :

- Fonctionne ensemble : La carte principale du poste de conduite Arduino DUE en V5 (*PDC_Arduino_JLF_BB7200_SE_DLL*), le KVB en V5, et le nouveau programme d'interface "InterfaceTSCArduinoJLF.exe" en V5.
- Fonctionne ensemble : La carte principale du poste de conduite Arduino DUE en V4 (*PDC_Arduino_JLF_BB7200_SE*), le KVB en V2 ou V5, le programme d'interface "TSCClassic Raildriver and Joystick Interface 3.3.0.9", ou mon interface pour OpenRails "InterfaceORArduinoJLF.exe".

Ce nouveau programme "InterfaceTSCArduinoJLF.exe" est compatible avec le montage "Poste de conduite - Réalisation JLF" en version V5.0 pour la BB7200, et la version V5.0 du KVB.

Aspect du programme "InterfaceTSCArduinoJLF" :



J'ai réalisé mon programme d'interface, pour apporter le moins possible de modification dans mon code de ma carte Arduino : https://www.la-tour.info/uts/uts_page15.html#conduite avec TSC.

Il faudra utiliser la version V5.0 du poste de conduite et le programme Arduino "PDC_Arduino_JLF_BB7200_SE_DLL".

Pour info, le premier programme d'interface utilisé "TSC Classic Raildriver and Joystick Interface 3.3.0.9" pour "Train Simulator Classic RailWorks de DTG" : <https://forums.dovetailgames.com/threads/ts-classic-raildriver-and-joystick-interface.72488/>

Informations :

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/>

2 / 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 "**InterfaceTSCArduinoJLF_config.txt**".

Quand le programme se lance, il va lire le fichier de configuration "**InterfaceTSCArduinoJLF_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 "**InterfaceTSCArduinoJLF_config.txt**" :

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

```
* Fichier InterfaceTSCrduinoJLF_config.txt Le 13/07/2026
* Fichier de configuration pour le programme : Interface Open Rails vers carte Arduino - JLF.exe
* -----
```

Il faut obligatoirement donner le chemin complet pour accéder au fichier "RailDriver64.dll" du jeu TSC.

repertoire_dll = E:\Train Simulator Classic\plugins

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, sont sous la forme : 0,52617883682250977.

On note parfois 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 choisit 2 chiffres après la virgule, on enverra "0,51" ce qui donne une précision de 1 %, (0,00 à 0,99).

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

Si l'on choisit 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, comme un compteur de vitesse 0 à 160 km/h.

Pour définir la précision, exemple :

precision = 3

Avec mon programme, on peut définir un intervalle de rafraîchissement 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

Ce programme prend 2 % du cpu, pour un timer à 50 milli secondes.

Le 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 le taux de cpu utilisé par mon programme à 0,2 %.

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).

Mettre "oui" par défaut. 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". Par défaut mettre "oui". 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.

3 / Démarrage du programme

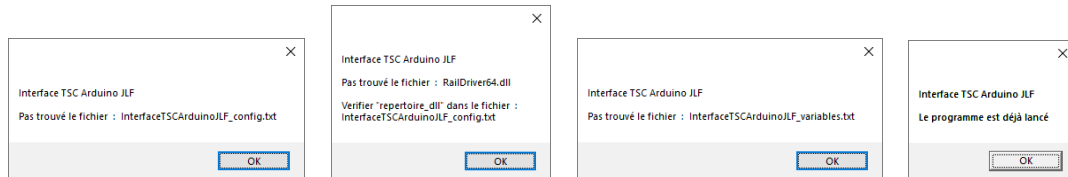
Penser à vérifier les mises à jour sur le site : https://www.la-tour.info/uts/uts_page15.html

Mettre dans le même répertoire, les trois fichiers :

InterfaceTSCArduinoJLF.exe (64 bits)
InterfaceTSCArduinoJLF_config.txt (Renseigné)
InterfaceTSCArduinoJLF_variables.txt (Renseigné ou vide)

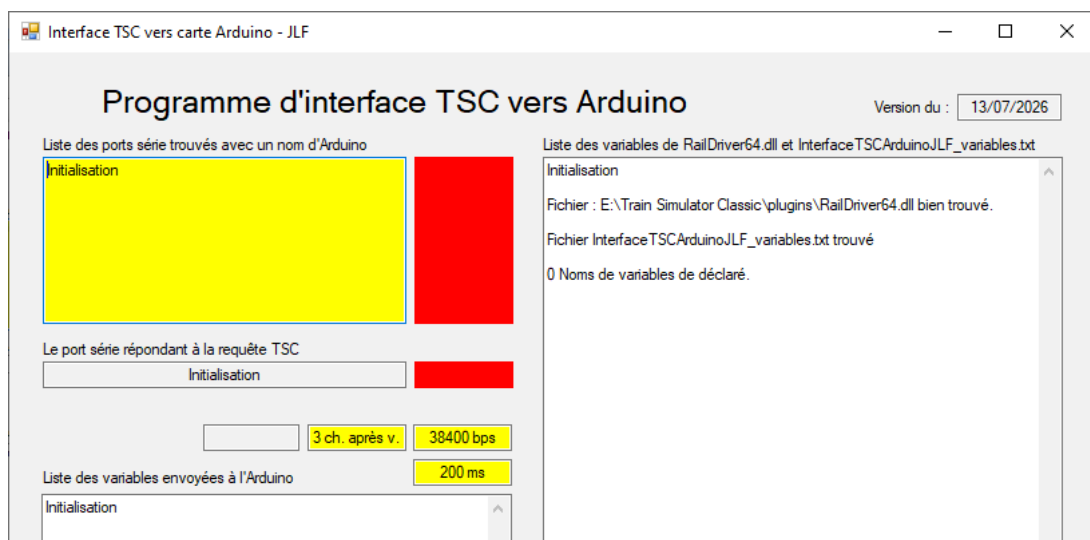
Lancer "**InterfaceTSCArduinoJLF.exe**".

Les fenêtres d'erreur possible :



Il n'y pas de boutons ni de menu. La configuration se fait dans le fichier "**InterfaceTSCArduinoJLF_config.txt**".

Si le fichier "InterfaceTSCArduinoJLF_config.txt" comporte des erreurs, les cases passent en couleur jaune.



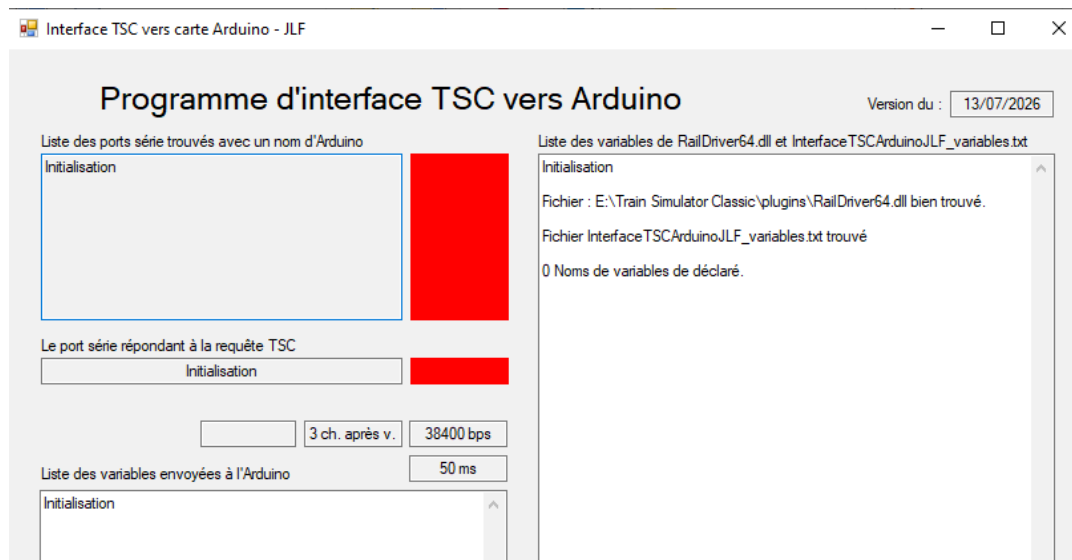
Si les paramètres du fichier de configuration ne sont pas lisibles, ou hors limites, le programme affiche des rectangles jaunes. Dans ce cas, le programme utilisera les paramètres par défaut.

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

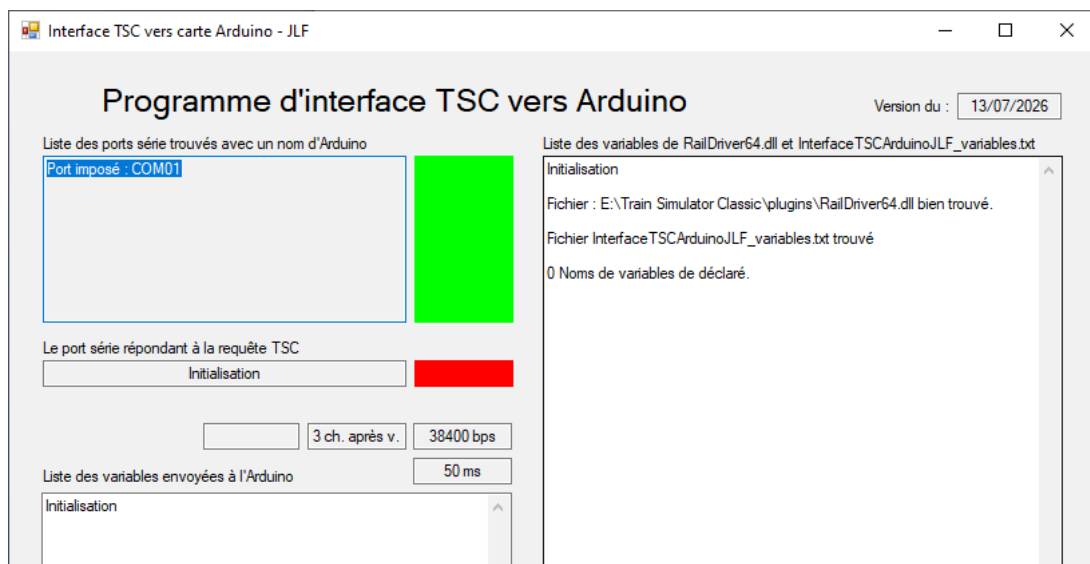
Si l'on a choisi "auto", le programme affiche les ports existants, marqués "Arduino".

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

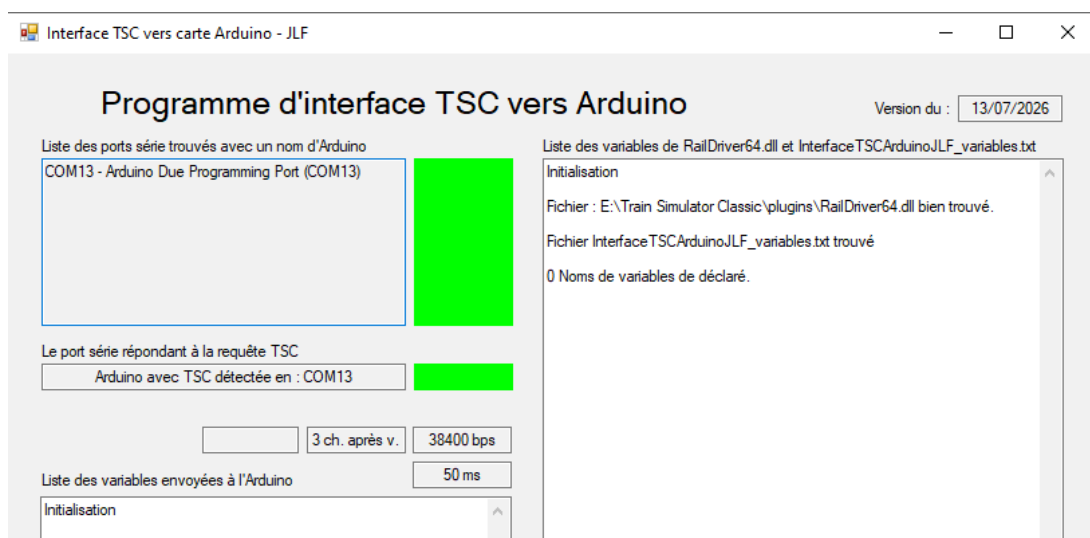
La couleur du petit rectangle rouge, indique si le programme a détecté au moins un port "Arduino" pour Open Rails.



Avec le port COM01 imposé :



Avec le port en mode "auto" :



Les connexions se font en deux temps, d'abord avec l'Arduino au bout de 5 secondes, puis après avec TSC au bout de 5 secondes.

Dans un premier temps, le programme cherche un port série avec une carte Arduino, toutes les 5 secondes.

En mode port automatique, le programme envoie "<:JLF?:0>", et retient la carte qui répond "<:JLF!>".

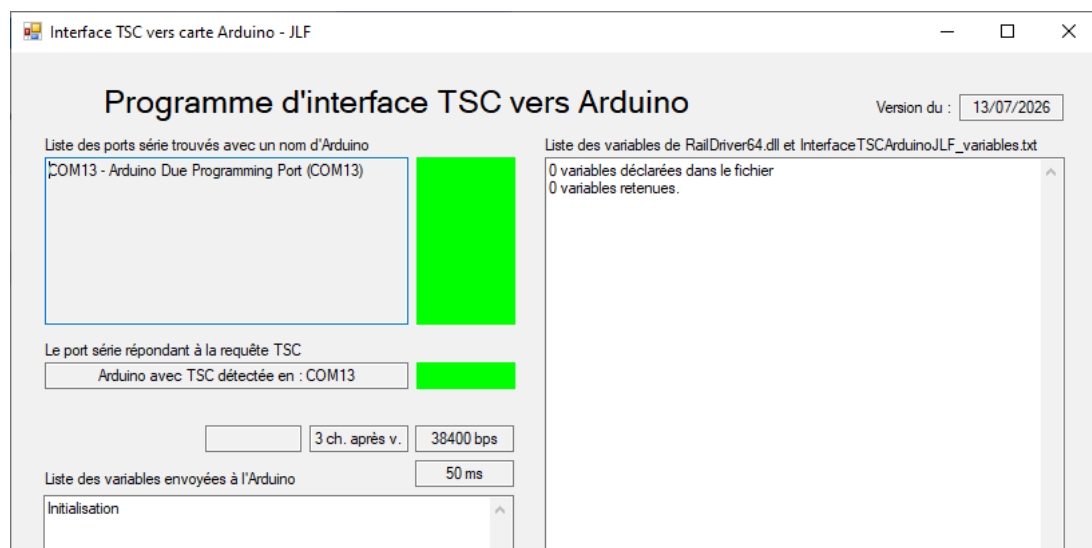
En mode port fixe imposé (COMxx), si le programme peut ouvrir le port, celui-ci passe directement au vert.

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 TSC.

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



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

On affiche dans le titre, le nombre de variables lues et le nombre de variables retenues une fois les doublons supprimés. Le maximum est de 300 variables lues et 300 variables retenues.

On note :

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

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

[50 ms] = On va chercher la page web toutes les 50 milli secondes.

Dans le fichier de configuration, il faut spécifier "optimisation = oui" par défaut. Au bout d'une minute, on ne rafraichira plus les listes de variables à l'écran. Ca économise du cpu. On passe de 2 % à 0,2 % de cpu.

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

A chaque accès à TSC, 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 TSC donne de nouvelles valeurs.

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.

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.

Mettre le timer à 50 msec, fonctionne bien chez moi. J'utilise un Arduino DUE, 32 Bits à 84 Mhz.

Mettre optimisation = oui.

Il faut maintenant configurer manuellement un fichier avec toutes les variables demandées, propre à un type de locomotive.

4 / La configuration de la liste des variables échangées

Par soucis d'optimisation, il ne faut pas faire envoyer par mon programme, toutes les données disponibles depuis TSC.

Il y a des variables comme "H_secondes_control", qu'il faut éviter d'envoyer toutes les secondes.

Le fichier "**InterfaceTSCArduinoJLF_variables.txt**" doit être renseigné à la main. C'est la liste des variables demandées à TSC.

Pour avoir la liste de toutes les variables disponible, le fichier "**InterfaceTSCArduinoJLF_simu_out.txt**" est automatiquement renseigné par le programme. Il liste toutes les variables disponibles dans TSC.

Pour réaliser cette liste dans le fichier "**InterfaceTSCArduinoJLF_simu_out.txt**", il faut dans cet ordre précis :

- Démarrer dans TSC, la locomotive correspondant au pupitre.
- Avoir une carte Arduino qui réponde à mon programme, ou un port COMxx fixe.
- Lancer mon programme "InterfaceTSCArduinoJLF.exe".
- Rendre visible et actif TSC.

Mon programme a alors renseigné le fichier "InterfaceTSCArduinoJLF_simu_out.txt". Cette action ne se fait qu'une seule fois, en phase de démarrage. Le fichier est écrasé à chaque fois.

Exemple de contenu du fichier de sortie "**InterfaceTSCArduinoJLF_simu_out.txt**" :

```
InterfaceTSCArduinoJLF_donnees      BB7391_EnVoyage.  
Pour que ce fichier soit renseigné, il faut que l'on soit dans une cabine de locomotive,  
avant de lancer ce programme.  
  
Variable : Valeur minimum : Valeur maximum  
  
0_VI_units_control : 0 : 1  
2tons_control : -0,25 : 0,25  
AbsoluteSpeed_control : 0 : 100  
acquit_auto_control : 0 : 1  
aigu_exc_1_control : -1000 : 2000  
...  
volet_ADC_control : 0 : 1  
volet_cond_control : 0 : 1  
Wheelslip : 1 : 10  
Wipers : 0 : 1  
Z_alerte_control : 0 : 1  
Z_SUR_control : 0 : 1
```

On a la liste de toutes les variables que TSC peut échanger.

Sur la même ligne, on a le nom de variable, la valeur minimum et la valeur maximum.

Si le minimum est '0' et le maximum '1', certaines variables retournent '0' ou '1', mais d'autres variables peuvent retourner une valeur en flottant de 0.000 à 1.000.

On va renseigner à la main le fichier "**InterfaceTSCArduinoJLF_variables.txt**".

Pour recevoir une variable comme "AbsoluteSpeed_control", on écrit ce nom de variable dans le fichier "**InterfaceTSCArduinoJLF_variables.txt**".

Remarques :

Il ne faut mettre dans ce fichier, que les variables que l'on veut recevoir sur l'Arduino.

Le retour d'information est nécessaire pour les interrupteurs.

Le retour d'information n'est pas souhaitable pour les boutons poussoir momentanés. Ca ne sert à rien, car ils ont un état stable par défaut. Ca réduit le trafic inutile sur la liaison série.

Par exemple, pour allumer le bouton poussoir [VAL] du KVB sur le pupitre, on y met dans ce fichier la variable : KVB_BP_VAL_lumiere_control.

Quand on appuie ce bouton poussoir (*Momentanément*), on envoie le nom de la variable "<:KVB_BP_VAL_ON_control:1>" du pupitre vers TSC.

Dans ce cas, ce nom de variable n'a pas besoin d'être dans ce fichier, car l'état du simulateur correspondra toujours à l'état du pupitre.

Il ne faut pas mettre : KVB_BP_VAL_ON_control (*C'est l'état momentané du bouton poussoir !*).

Dans ce fichier seul le premier champ de chaque ligne est utilisé. On peut écrire "aiguille_UL_C_control" ou "aiguille_UL_C_control : 0 : 16."

Une ligne qui commence par * sera ignoré.

Exemple de contenu du fichier "InterfaceTSCArduinoJLF_variables.txt" :

* InterfaceTSCArduinoJLF_donnees

* Variables

* BB 7200 de SimExpress

AbsoluteSpeed_control

aigu_exc_1_control

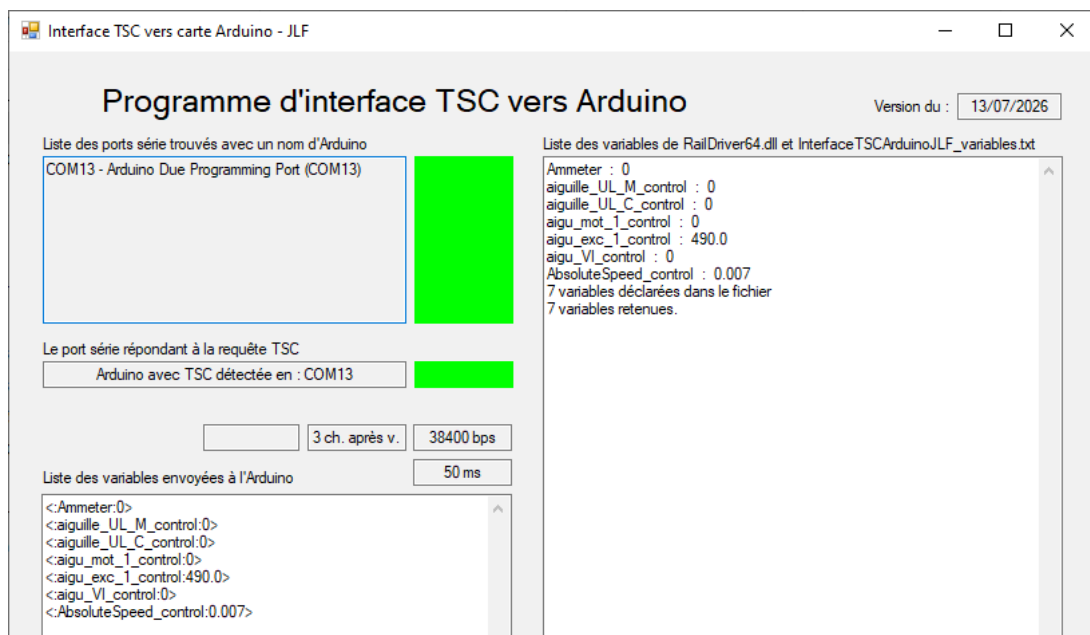
aigu_mot_1_control

aigu_VI_control : 0 : 160

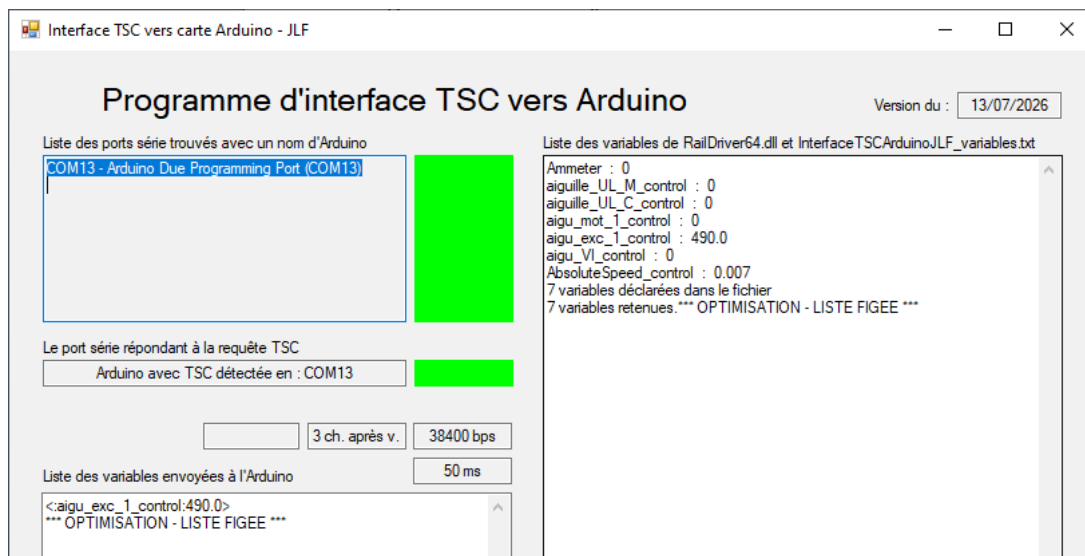
aiguille_UL_C_control : 0 : 16

aiguille_UL_M_control : 0 : 25

Ammeter : -1000 : 2100



Au bout d'une minute, l'affichage n'est plus rafraichi, pour économiser du cpu. C'est indispensable une fois la configuration établie.



4 / 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 et ce champ reste vide.
- : = 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 0.000 et 1.000.

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

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.

TSC envoie souvent des valeurs qui changent très fréquemment sans être significatif (Avec 8 chiffres après la virgule). Il faut filtrer cela pour éviter la saturation de la ligne série.

Pour éviter d'envoyer trop souvent des données ou des données trop longues sur la ligne série, j'ai ajouté des traitements dans le code "InterfaceTSCArduinoJLF".

Simplification de l'envoi des valeurs '0' et '1'.

<:disjoncteur:0.000> est envoyé avec le texte : <:disjoncteur:0>

<:disjoncteur:1.000> est envoyé avec le texte : <:disjoncteur:1>

Evite les nombreux chiffres non significatifs, après la virgule pour le KVB. Le KVB n'échange que des entiers.

<:KVB*****:0.001258> est envoyé avec le texte : <: KVB*****:0>

<:KVB*****:0.998745> est envoyé avec le texte : <: KVB*****:1>

La vitesse est trop souvent envoyée, car la partie décimale change tout le temps.

Pour <:AbsoluteSpeed_control:0.001>, on ne laisse que 1 chiffre après la virgule, car elle n'est pas stable.

Réduit d'office le nombre de chiffres après la virgule, pour les valeurs importantes (>1000, >100 et > 10).

<:courant:1000.0002>> est envoyé avec le texte : <:courant:1000>

<:courant:100.0002>> est envoyé avec le texte : <:courant:100.0>

<:courant:10.0002>> est envoyé avec le texte : <:courant:10.00>

5 / 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 : "**InterfaceTSCArduinoJLF.zip**".

Pour lancer Visual Studio, cliquer sur "**InterfaceTSCArduinoJLF.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/InterfaceTSCrduinoJLF_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_TSCI()

Va lire la librairie TSC.

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_TSC_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 = 300; Nombre de variables maximum.

const int variable_envoi_nb_max = 300; Nombre de variable, 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.

6 / Exemple de données envoyées à l'Arduino

Avec une BB22000 :

```
<: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.

7 / 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

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 BB7200_JLF_DLL V5 sur mon site.

1 / Si l'on veut modifier le programme source

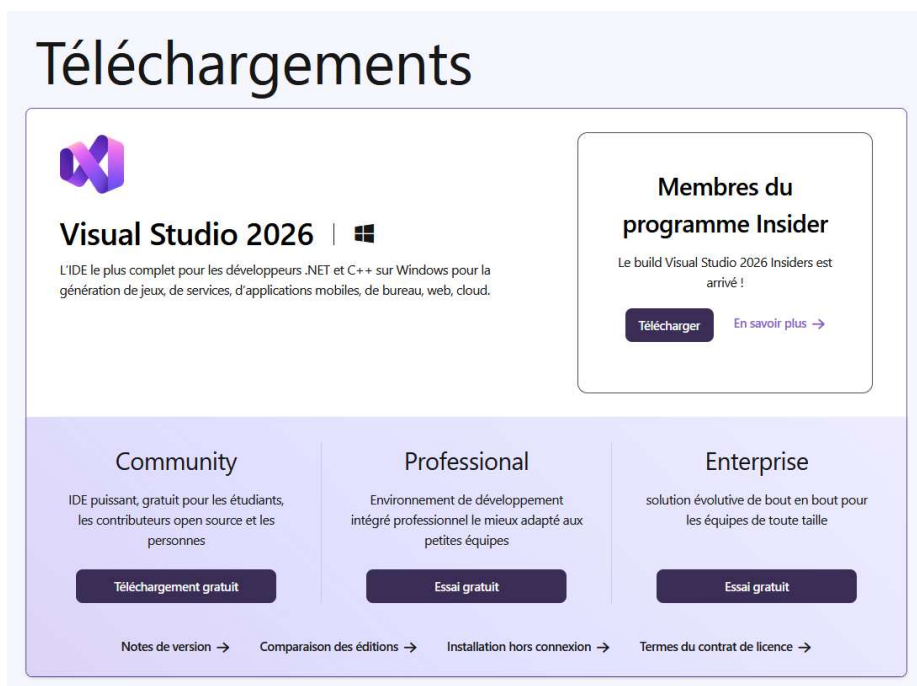
Le programme est écrit en C#. Je l'ai écrit dans l'environnement gratuit, C# Visual Studio 2026 de Microsoft, version "Community". <https://visualstudio.microsoft.com/fr/downloads>

Mode d'utilisation de Visual Studio : Community

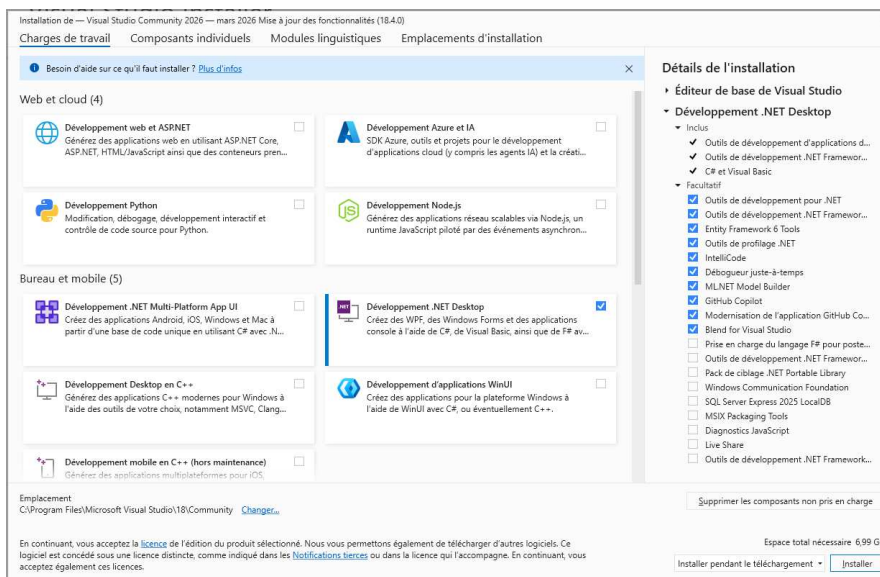
Type de programme : Développement .NET Desktop

Environnement : .NET Framework 4.8 (Et ultérieur)

A partir de ce lien, choisir la version "Community" :



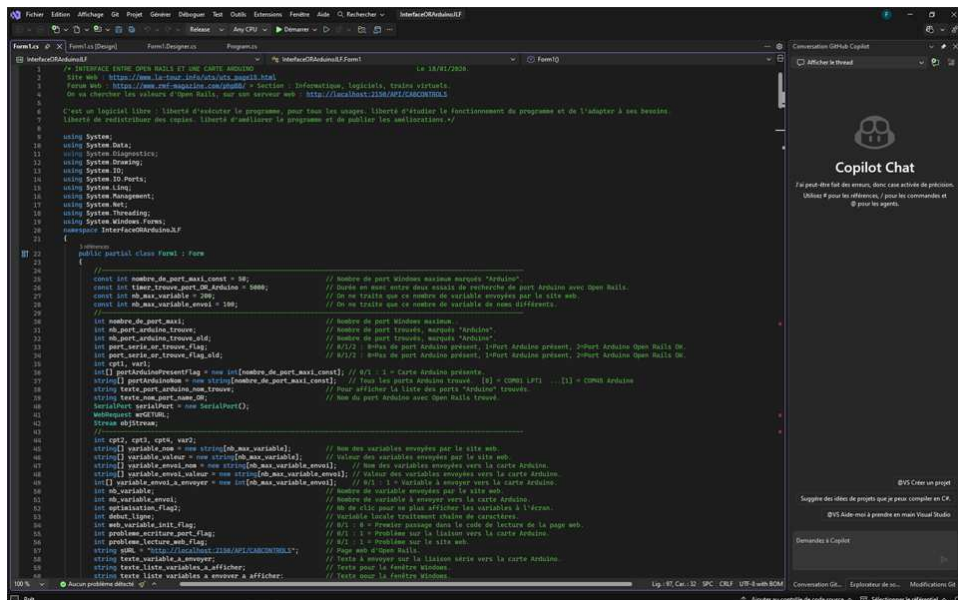
Choisir la version "Développement .NET Desktop" :



Le programme prend 7 Go sur disque.

Il faudra un compte Microsoft, pour utiliser ce logiciel. C'est gratuit.

Une fois Visual Studio installé, cliquer sur le fichier livré, "InterfaceTSCArduinoJLF.csproj" :



Si vous vous voulez simplement recompiler le code, choisir le mode "Release".

Sur le bandeau, c'est affiché [Release v] ou [Debug v]. Choisir [Relesé v].

Pour créer le programme, Menu : Générer > Générer la solution.

Dans le répertoire "InterfaceTSCArduinoJLF\bin\Release", on trouve l'exécutable "InterfaceTSCArduinoJLF.exe" mis à jour.

Si vous modifier le programme, passer en mode "Debug". Il sera préférable de compiler la version finale en mode "Release".

Program.cs	Programme principal, appelé au démarrage.
Form1.cs	Sous programme principal qui fait tout.
Form1.cs [Design]	Elaboration de la fenêtre d'affichage.
Resources.resx	Autres fichiers utilisés. Contient les rectangles rouges et verts utilisés dans la fenêtre.

J'accède à librairie "RailDriver64.dll", ce qui oblige à compiler mon programme en 64 bits.

Il faut générer le .exe en 64 bits, pour pouvoir lire la librairie 64 bits.

Configuration obligatoire : Projet > Propriétés InterfaceTSCArduino > Build > Plateforme cible = x64.

Si vous voulez une version 32/64 bits, il faut modifier la configuration du projet, et utiliser la librairie 32 bits.

A+