Documentation EVE

***Environnement Virtuel d’Essais***

***Auteurs***

François EVEN

Julien FRANCHINEAU

Version 0.5

Sommaire

[Avant-propos 3](#_Toc315284476)

[1. Manuel d’installation développeur 4](#_Toc315284477)

[1.1 Installation de l’OS 4](#_Toc315284478)

[1.2 Installation de Glade 4](#_Toc315284479)

[1.2.1 Droits super-utilisateur / Activation mode root 4](#_Toc315284480)

[1.2.2 Version Glade 2.12 5](#_Toc315284481)

[1.3 Installation de RabbitVCS 6](#_Toc315284482)

[2. Manuel du développeur 7](#_Toc315284483)

[2.1 Précautions d’utilisation de Glade 7](#_Toc315284484)

[2.2 Fichiers d’EVE 8](#_Toc315284485)

[2.2.1 Fichiers de configurations 8](#_Toc315284486)

[2.2.2 Fichiers de l’IHM 8](#_Toc315284487)

[2.2.3 Fichiers EVE\_QS 9](#_Toc315284488)

[2.2.4 Fichiers sources EVE 10](#_Toc315284489)

[2.2.5 Fichier Makefile 10](#_Toc315284490)

[2.3 Fichiers de QSx86 11](#_Toc315284491)

[2.4 Documentation des QS\_data\_e 13](#_Toc315284492)

[2.5 Etat d’avancement et améliorations 15](#_Toc315284493)

[2.5.1 Etat global 15](#_Toc315284494)

[2.5.2 Développements - TODO LIST 16](#_Toc315284495)

[3. Manuel d’installation utilisateur 20](#_Toc315284496)

[4. Manuel d’utilisation 21](#_Toc315284497)

[4.1 Fonctionnement de l’interface CAN 21](#_Toc315284498)

[4.2 Terminal UART 22](#_Toc315284499)

[4.3 Carte électronique interactive 22](#_Toc315284500)

[4.4 Options de simulation 23](#_Toc315284501)

[Table des versions et révisions 24](#_Toc315284502)

**Table des illustrations**

[Figure 1 : Tableau récapitulatif des QS\_DATA 14](#_Toc315284503)

[Figure 2 : Interface d’affichage du CAN 21](#_Toc315284504)

[Figure 3 : Interface des terminaux UART 22](#_Toc315284505)

[Figure 4 : Interface de la carte électronique interactive 23](#_Toc315284506)

[Figure 5 : Interface des options de simulation 23](#_Toc315284507)

# Avant-propos

Dans ce document, nous allons proposer une documentation du logiciel en plusieurs parties afin de répondre aux différentes questions qui pourraient se poser lors de son utilisation.

Pour ce faire, nous expliquerons dans les deux premières parties les différentes manipulations à effectuer pour un développeur souhaitant améliorer et faire évoluer le logiciel EVE. Il s’agit ici d’une aide au développement afin de permettre de rapidement mettre en place l’environnement de développement.

Ensuite, nous détaillerons la procédure d’installation pour un simple utilisateur qui souhaite tester les codes des cartes du robot. L’objectif est ici aussi de rapidement mettre en place l’environnement de test du logiciel EVE pour que l’utilisateur puisse, de manière simple, réaliser des tests avec notre logiciel.

Enfin, nous proposerons un manuel d’utilisation du logiciel, le plus complet possible pour faciliter la prise en main de l’utilisateur devant toutes les options qui sont prévues.

Cette documentation pourra être complétée au fur et à mesure de l’utilisation du logiciel pour éclaircir certains points sombres qui n’auraient pas été assez détaillés ou expliqués.

# Manuel d’installation développeur

Nous détaillerons ici les différentes étapes permettant de mettre en place l’environnement de développement du logiciel EVE.

## Installation de l’OS

Le développement du logiciel est réalisé sous Linux, distribution Ubuntu. Ici, vous avez plusieurs possibilités pour mettre en place cet OS :

* Installation comme OS principal de la machine
* Installation en dual boot
* Installation sur une machine virtuelle

Les deux premières méthodes ne seront pas détaillées dans ce document, car nous avons choisi de développer grâce à une machine virtuelle faisant tourner Ubuntu.

Procédure d’installation :

1. Télécharger et installer la machine virtuelle Oracle VM VirtualBox
2. Récupérer une distribution d’Ubuntu. Pour notre projet, nous avons utilisé la distribution 10.04
3. Installer Ubuntu sur la VirtualBox

## Installation de Glade

L’installation de Glade se fait par le biais des téléchargements d’applications pour Ubuntu. Elle nécessite les droits super-utilisateur, une connexion internet et plusieurs manipulations. L’installation est proposée pour la version 2.12 de Glade.

### Droits super-utilisateur / Activation mode root

Pour obtenir les droits super-utilisateur, aussi appelé le mode root, vous devez suivre la procédure suivante :

* Ouvrir un terminal
* Taper la commande : sudo –s
* Taper votre mot de passe utilisateur lorsque cela vous est demandé

Si tout s’est correctement déroulé, vous devriez être en mode root. Ne fermez pas le terminal, vous êtes en mode root uniquement dans ce terminal, vous devrez y poursuivre l’installation.

### Version Glade 2.12

L’installation de Glade 2.12 peut se faire selon la procédure suivante :

1. Dans un terminal en mode root, tapez les commandes suivantes :

apt-get install libxml2-dev

apt-get install libgtk2.0-dev

1. Téléchargez ensuite Glade à l’adresse suivante :

<http://ftp.gnome.org/pub/GNOME/sources/glade/2.12/glade-2.12.2.tar.bz2>

1. Une fois le téléchargement fini, décompressez les sources.
2. Dans les fichiers suivants :

* glade/glade\_keys\_dialog.c
* glade/gbwidgets/gbclist.c
* glade/glade\_menu\_editor.c
* glade/glade\_project\_view.h

Remplacez la ligne : #include ‹gtk/gtkclist.h›

Par : #include ‹gtk/gtk.h›

1. Lancez l’installation de Glade en suivant la procédure décrite dans le fichier INSTALL du dossier glade-2.12 (s’y référer en cas de problème). Les lignes de commande à utiliser sont :

* cd <chemin du dossier glade-2.12>
* ./configure
* make
* make install
* make clean

1. Installez automake et autoconf grâce à la commande suivante :

apt-get install automake autoconf

Vous avez terminé l’installation de Glade 2.12 qui devrait maintenant faire parti de vos applications Ubuntu.

## Installation de RabbitVCS

TortoiseSVN n’existant pas sous Linux, il faut trouver une alternative pour pouvoir utiliser le SVN du club. Nous vous proposons ici l’alternative RabbitVCS. Il s’agit d’un équivalent à Tortoise pour Linux. La procédure d’installation est simple :

* Passez en mode root (voir 1.2.1)
* Tapez les commandes suivantes :
  + sudo add-apt-repository ppa:rabbitvcs
  + sudo aptitude update
  + sudo aptitude install rabbitvcs-nautilus

RabbitVCS est désormais installé ! Il apparait comme Tortoise dans les menus d’options des dossiers (clic droit, comme Tortoise sur Windows). Vous devrez surement redémarrer votre ordinateur pour prendre en compte l’installation et voir apparaître les options de RabbitVCS.

L’utilisation est sensiblement la même que Tortoise, la fonction « Update » étant remplacée par la fonction « Mise à jour » et la fonction « Commit » par « Envoyer ».

Pour plus d’informations, vous pouvez vous rendre sur le lien suivant : <http://blog.nicolargo.com/2010/04/rabbitvcs-le-tortoisesvn-pour-linux.html>

# Manuel du développeur

Nous détaillerons ici quelles sont les fonctions des différents fichiers, ainsi que des conseils de développement pour éviter les problèmes que nous avons connus au début de la création du projet.

La dernière partie sera consacrée à décrire l’état actuel du logiciel et les fonctions qui restent à développer. Nous encourageons les futurs développeurs à maintenir cette partie à jour afin que les futurs développeurs sachent rapidement où en est le logiciel et quelles seraient les fonctions et options intéressantes à ajouter pour l’améliorer.

## Précautions d’utilisation de Glade

Normalement, vous ne devriez pas avoir à retoucher grandement l’interface générale de Glade. Si le cas se présente, vous devriez prendre quelques précautions d’utilisations. En effet, Glade est un logiciel assez joueur. Faites des sauvegardes de backup régulièrement, il n’est pas rare que Glade perde, après une modification, des fichiers importants (comme le fichier principal contenant les données), vous obligeant à recommencer votre travail.

Glade est aussi pointilleux sur le placement des fenêtres et une prise en main est nécessaire afin de pouvoir réaliser des interfaces complexes (telles que celle d’EVE). Les tutoriels sont rares sur la toile, il est donc conseillé de faire des essais en amont du projet.

La génération automatique de code C n’est plus disponible sur les dernières versions de Glade (au moment de l’écriture de ces lignes). Il s’agit d’un élément intéressant à avoir pour générer les 6000 lignes de préparation de l’interface. Glade utilise la bibliothèque GTK pour créer l’interface graphique. Plusieurs tutoriels de GTK sont disponibles sur internet, ce qui facilite son utilisation, et permet de développer et d’améliorer certains modules d’affichage sans avoir à repasser par Glade en allant directement ajouter les fonctions dans le code (de bons exemples sont disponibles dans callbacks.c).

## Fichiers d’EVE

Cette partie a pour objectif de lister et de détailler succinctement les fonctions des différents fichiers sources permettant le fonctionnement d’EVE.

### Fichiers de configurations

Les fichiers se situant dans le dossier « \_Configuration » contiennent les différentes commandes qui sont nécessaires pour démarrer EVE comme on le souhaite. L’utilisation de ces fichiers est décrite dans les paragraphes du manuel d’utilisation.

Les fichiers ont différentes fonctions :

* data.config : contient la configuration de l’affichage des modules QS. L’utilisateur peut, en modifiant ce fichier, choisir comme il le souhaite l’affichage des modules de toutes les cartes. Ce fichier est chargé par les threads de l’IHM.
* eve.config : indique quels codes de cartes vont être utilisés pendant la simulation. Ce fichier est chargé par le thread principal (le Manager).
* filtres.config : contient la liste des messages CAN qui doivent être filtrés sur le CAN pour ne pas être affichés sur l’IHM (évite le flood). Ce fichier est directement chargé dans le thread du CAN qui s’occupe de l’affichage des messages sur l’interface.
* messages.config : contient des messages CAN prédéfinis qui seront chargés au démarrage du logiciel et qui pourront être plus facilement envoyés par l’utilisateur. Ce fichier est chargé par les threads de l’IHM.

Chaque fichier est composé de façon similaire. Les commandes sont toutes précédées du caractère ‘$’, et les fichiers se terminent par la commande ‘$END\_OF\_FILE’. Pour une lecture plus simple de l’utilisateur, les commentaires ont été précédés du caractère ‘#’. Seules les commandes sont réellement analysées par EVE lors de la lecture du fichier.

### Fichiers de l’IHM

Tous les fichiers sources de l’IHM sont contenus dans le dossier IHM. Ces fichiers sont uniquement destinés à gérer l’interface graphique du programme. L’IHM fonctionne grâce à deux threads indépendants. L’un s’occupe d’afficher l’interface et d’envoyer les demandes utilisateurs au reste du programme et aux codes cartes, et l’autre s’occupe de mettre à jour l’interface en fonction des messages reçus des autres threads et des codes cartes.

Les fichiers ont chacun des fonctions spécifiques :

* IHM\_callbacks : contient toutes les fonctions qui s’enclenchent lors de l’appui sur un bouton de l’interface. Généralement, à chaque appui, on envoie un message aux programmes concernés.
* IHM\_display : il s’agit du thread de gestion de l’affichage. Il prépare et lance l’interface graphique. La fonction gtk\_main() bouclera tant que l’interface sera démarrée et appellera les fonctions situées dans IHM\_callbacks à chaque action de l’utilisateur.
* IHM\_interface : contient une seule fonction (de 6000 lignes) qui crée l’interface graphique d’EVE.
* IHM\_receiver : il s’agit du thread de mise à jour de l’affichage. Il récupère toutes les informations en provenance d’EVE et des codes cartes pour les afficher sur l’interface graphique.
* IHM\_support : contient des fonctions supplémentaires créées par Glade pour faciliter l’utilisation de GTK.

Le dossier pixmaps contient les photos affichées par l’interface graphique.

### Fichiers EVE\_QS

Ces fichiers contiennent des fonctions utiles pour tous les autres fichiers du logiciel. Ils permettent de synthétiser des fonctions qui utilisent des fonctions de base du C ou encore de définir des choses utiles pour tout le monde.

Ces fichiers contiennent :

* EVE\_all : permet d’inclure tous les autres fichiers EVE\_QS, global\_vars, la bibliothèque GTK et les bibliothèques C nécessaires pour tous les fichiers. Ce fichier doit être inclus dans tout fichier source du projet.
* EVE\_functions : contient des fonctions utiles dans l’ensemble du programme EVE.
* EVE\_messages : contient les fonctions permettant la communication entre processus pour les files de messages et la mémoire partagée.
* EVE\_types : contient tous les types définis pour EVE, ainsi que les messages possible de retours d’erreurs.

Ces fichiers sont inclus dans tout fichier source du projet. Ils permettent d’uniformiser un certain nombre de choses pour qu’elles soient utilisables par tous (un peu sur le principe de la QS). Ces fichiers doivent être identiques à ceux contenus dans QSx86/QS/EVE\_QS qui seront compilés avec les codes des cartes (à l’exception du fichier EVE\_all, remplacé par EVE\_all\_x86 qui fournit les includes spécifiques à la nouvelle QS).

### Fichiers sources EVE

Enfin, dans le répertoire principal se trouvent les autres fichiers nécessaires au fonctionnement d’EVE.

Chaque fichier a une fonction spécifique :

* can : il s’agit du thread simulant le bus CAN du robot. Toutes les fonctions liées au CAN passent par ce fichier. Il s’occupe de récupérer les messages des différentes cartes, les réexpédie à toutes les cartes en fonctionnement, et affiche sur l’IHM le message qui vient de transiter par le thread, après un filtrage (pour éviter le flood).
* global\_clock : correspond au thread de l’horloge générale. Ce thread se contente de mettre à jour les différentes valeurs d’horloge d’EVE en fonction des demandes de simulation (lecture, pause, stop) qui permettent de cadencer le simulateur.
* global\_vars : contient toutes les variables globales d’EVE. Il est impératif d’utiliser un mutex lors de l’accès à une de ces variables (en lecture ou écriture). En effet, en multithread, il est très vivement conseillé de sécuriser l’accès aux variables globales pour éviter tout problème d’accès concurrent ou simultané (pour plus d’informations, consulter des ASTRE ^^).
* main : contient la fonction principale du programme qui se contente d’initialiser les variables globales et de démarrer le Manager.
* manager : il s’agit du thread principal qui s’occupe de charger la configuration d’EVE et de démarrer tous les threads, notamment ceux qui se chargeront de démarrer les codes des cartes dans un autre processus. Il s’agit d’un thread qui sert surtout au démarrage ou à l’arrêt du logiciel et qui gère les autres threads.

### Fichier Makefile

Le fichier Makefile sert à compiler le logiciel EVE, ainsi que les codes des cartes. Tout a été regroupé dans un seul fichier afin de minimiser le nombre de fichiers nécessaires à la compilation des différents programmes. Ainsi, EVE appelle ce makefile lors de son initialisation pour compiler les codes cartes.

Pour la compilation d’EVE :

* La commande –lpthread permet de compiler le programme avec les bibliothèques nécessaires au fonctionnement en multithreads.
* La commande `pkg-config --cflags --libs gtk+-2.0` permet de compiler le programme avec la bibliothèque GTK pour faire fonctionner l’IHM. Pour les plus curieux, vous pourrez aller regarder le makefile fournit par Glade pour compiler son IHM et constater qu’il est grandement simplifié ici :)

## Fichiers de QSx86

Cette partie a pour objectif de lister et de détailler succinctement les fonctions des différents fichiers sources permettant le fonctionnement de la QSx86 qui doit être compilée avec les codes de chaque carte pour les rendre compatibles avec EVE.

Comme pour la QS classique, le dossier QS, situé dans le dossier QSx86 du projet, doit être copié dans le répertoire des codes de cartes que l’on souhaite utiliser pendant la simulation.

Tous les fichiers sources de la QS normale sont encore présents dans cette nouvelle QS dédiée à EVE. Ils ont quasiment tous été modifiés pour pouvoir être utilisés sur un ordinateur avec EVE. Ils gardent tous leurs fonctionnalités d’origine qui ont juste été modifiées pour correspondre aux attentes de la simulation EVE. La liste des fichiers opérationnels est disponible dans la partie consacrée à l’état d’avancement d’EVE.

De nouveaux fichiers font tout de même leur apparition pour permettre à EVE de s’exécuter correctement. Le package EVE\_QS correspond au même package que dans les fichiers sources d’EVE. Seul le fichier EVE\_all\_x86 remplace le fichier EVE\_all pour permettre l’intégration dans le package QSx86 des modules de EVE\_QS.

Les fichiers suivants font aussi leur apparition :

* EVE\_config : contient des « define » à utiliser comme dans Global\_config. Pour l’instant, ils activent juste certaines fonctions d’IT timers. Il est impératif d’activer ici les fonctions d’IT sinon elles ne s’exécuteront pas lors de l’exécution du code.
* QS\_EVE\_manager : contient les fonctions que les codes cartes doivent appeler pour pouvoir exécuter la simulation. Elles permettent d’exécuter les fonctions d’update qui mettent à jour, entre autres, les timers et récupèrent les informations venant de l’IHM.
* QS\_EVE\_hard\_emulation : contient les fonctions permettant de simuler le matériel du dsPIC. Les structures des ports ont été en partie reconstituées pour permettre aux codes de compiler et de pouvoir s’exécuter correctement. Toute modification sur les ports est ici détectée et envoyée vers l’IHM.

Quelques lignes de commandes ont été ou sont à ajouter dans les « main » de chaque code pour permettre l’exécution correcte de la simulation.

* Commande d’initialisation

Les lignes suivantes sont à ajouter au tout début de la fonction main des codes cartes. Elles permettent d’initialiser toutes les variables qui vont être utiles à la simulation. Les « define » permettent d’éviter d’exécuter ces fonctions lorsque l’on compile avec la QS normale quand le code est destiné à être exécuté sur le dsPIC.

#ifdef USE\_QSx86

EVE\_manager\_card\_init() ;

#endif

* Commande d’update

Les lignes suivantes sont à ajouter au début du while(1) dans la fonction main de chaque code de carte. Elle permet de faire une mise à jour des informations pendant la simulation et de faire des traitements spéciaux requis pour EVE. Les « define » permettent d’éviter d’exécuter ces fonctions lorsque l’on compile avec la QS normale quand le code est destiné à être exécuté sur le dsPIC.

#ifdef USE\_QSx86

EVE\_manager\_card() ;

#endif

Ces quelques lignes sont les seules traces à ajouter dans les codes cartes pour permettre leur exécution avec EVE.

## Documentation des QS\_data\_e

Cette partie documente les différentes possibilités des messages de type QS\_data qui peuvent prendre de multiples formes. Elle permettra de rapidement comprendre comment fonctionnent les messages déjà mis en place. Il est vivement recommandé de compléter cette partie à chaque ajout de message.

QS\_xxx\_UARTx :

Ici, c’est uniquement l’identifiant qui est intéressant. Aucune donnée n’est à analyser dans le message.

QS\_LED\_xxx :

Pour ce message, l’identifiant référence la LED concernée, et la variable data1 contient l’état de la LED (0 pour éteinte et 1 pour allumée).

QS\_BUTTON\_x :

Comme pour les LED, l’identifiant référence le bouton qui est concerné, et la variable data1 contient l’état du bouton (0 pour relâché, 1 pour appuyé).

QS\_PWM :

L’identifiant référence que le message contient des informations sur les PWM. La variable data1 contient le numéro du canal, data2 contient la puissance de la PWM, et data3 contient le sens.

QS\_SIMULATION\_xxx :

Aucune donnée de contenue dans le message, seul l’identifiant est à analyser pour savoir quels sont les ordres de la simulation.

|  |  |  |  |
| --- | --- | --- | --- |
| Identifiant | Data1 | Data2 | Data3 |
| QS\_UNDEFINED\_DATA | - | - | - |
| QS\_START\_UART1 | - | - | - |
| QS\_STOP\_UART\_1 | - | - | - |
| QS\_START\_UART2 | - | - | - |
| QS\_STOP\_UART2 | - | - | - |
| QS\_LED\_RUN | Etat | - | - |
| QS\_LED\_CAN | Etat | - | - |
| QS\_LED\_UART | Etat | - | - |
| QS\_LED\_USER | Etat | - | - |
| QS\_LED\_USER2 | Etat | - | - |
| QS\_LED\_ERROR | Etat | - | - |
| QS\_BUTTON\_1 | Etat | - | - |
| QS\_BUTTON\_2 | Etat | - | - |
| QS\_BUTTON\_3 | Etat | - | - |
| QS\_BUTTON\_4 | Etat | - | - |
| QS\_BUTTON\_RESET | Etat | - | - |
| QS\_PWM | N° Canal | Puissance % | Sens |
| QS\_SIMULATION\_RUN | - | - | - |
| QS\_SIMULATION\_PAUSE | - | - | - |
| QS\_SIMULATION\_STOP | - | - | - |

Figure 1 : Tableau récapitulatif des QS\_DATA

## Etat d’avancement et améliorations

Cette partie est destinée à synthétiser l’état d’avancement du logiciel EVE. Elle permettra de stocker les informations sur les modules fonctionnels et de mémoriser les modifications/développements qu’il faudra effectuer sur le code pour les prochains développeurs. Le code devrait prendre de l’ampleur et se complexifier au fil des développements pour rendre EVE toujours plus complet et performant pour faire des tests, et cette partie permettra aux nouveaux développeurs de rapidement savoir ce qui est fait et ce qu’ils doivent faire.

### Etat global

**Etat du 27/01/2012**

EVE est aujourd’hui capable de s’exécuter et de démarrer les codes Stratégie et Propulsion. Les premiers tests n’ont pas montré de problèmes visibles lors de l’exécution du code Stratégie. Des tests plus poussés sont à faire sur la propulsion.

Les différents threads d’EVE disposent des fonctions minimales requises pour fonctionner :

* Le manager charge les fichiers de configurations et démarre l’ensemble des threads demandés. Lors du reset d’une carte, le manager s’occupe de redémarrer le thread concerné. Lors de la demande d’arrêt de l’utilisateur, le manager attend que tous les threads se soient correctement terminés avant de fermer le programme.
* L’horloge générale (global\_clock) s’exécute correctement en fonction de l’état de la simulation. Elle démarre à zéro à chaque nouvelle simulation, et le thread s’éteint correctement lors de la demande d’arrêt d’EVE. La base de temps minimum est réglée sur 1ms. Cela inclut que l’on ne peut pas faire de simulation précise en dessous de la milliseconde. Cela n’a pour l’instant pas posé de problème particulier vu la vitesse des processeurs d’ordinateur pour exécuter le code.
* Les variables globales (global\_vars) sont normalement toutes protégées par des mutex. Pas de problème notoire à signaler.
* Le module CAN récupère les messages, les réexpédie à toutes les cartes et met à jour l’IHM. Il faudrait tester avec plusieurs codes en fonctionnement, ce qui n’a pas encore été fait.
* Le package EVE\_QS est fonctionnel, aucun problème n’a été détecté. Quelques suppléments à ajouter sont détaillés dans la TODO LIST.
* L’IHM dispose des fonctions de base à son fonctionnement ce qui permet une utilisation basique du simulateur (CAN, UART, LED et boutons).

La plupart des fonctions minimales de la QSx86 sont implémentées pour permettre une utilisation basique d’EVE et de pouvoir tester les codes.

* Les fonctions de QS\_EVE\_manager semblent fonctionner correctement. Les updates des Timers sont à vérifier plus en profondeur mais ne semblent pas poser de problème.
* Les fonctions de QS\_EVE\_hard\_emulation semblent aussi fonctionner correctement.

Au niveau des modules de la QS normale modifiée :

* QS\_timers semble fonctionnel. D’autres tests doivent être effectués.
* QS\_buttons est fonctionnel et les boutons sont utilisables depuis l’IHM.
* QS\_can est fonctionnel et permet d’envoyer des messages CAN. La fonction de réception est encore à tester.
* QS\_macro est fonctionnel, les debug\_printf sont gérés et affichés par l’IHM. Il reste à revoir la fonction de reset lors des assert.
* QS\_pwm a été refait, mais il reste à tester.

### Développements - TODO LIST

#### Module Global\_vars

Les variables globales sont un point sensible de la programmation multithread. Des mutex ont été ajoutés lors de l’accès à ces variables, mais cela alourdit le code. Il faudrait envisager des fonctions d’accès GET/SET, comme c’est déjà fait pour certaines, pour toutes les variables afin de supprimer les blocages/déblocages des mutex un peu partout dans le code.

L’initialisation de toutes les variables n’est pas encore complète. De plus, la variable globale button\_save qui pointe sur l’interface via un bouton pourrait être modifiée par un pointeur sur la fenêtre qui pourrait être récupéré au moment de sa création.

#### Module CAN

Des tests doivent être effectués avec plusieurs cartes en exécution pour vérifier la bonne réaction du thread.

Les options de l’IHM spécifiques au CAN doivent être ajoutées car elles sont pour l’instant inexistantes (à part la fonction Clear qui nettoie l’affichage). Les messages prédéfinis doivent être chargés et pouvoir être envoyés au CAN, de même que les messages manuels (l’enregistrement des messages manuels est aussi à ajouter, ils doivent s’enregistrer à la suite des messages dans le fichier messages.config). Le thread CAN n’a pour l’instant rien prévu pour récupérer ces messages.

Enfin, les filtres activables via l’IHM ne sont pas fonctionnels (rien de développé pour l’instant). Ils doivent être modifiables dynamiquement par l’utilisateur pendant une simulation et être transmis au CAN qui devra ajouter les filtres à ses filtres existants.

#### Module Global\_clock

L’amélioration notable à ajouter à ce module est de pouvoir faire un saut de temps. Ce qu’il faut comprendre derrière ça, c’est de pouvoir faire avancer la simulation jusqu’à un certain temps et qu’elle se bloque d’elle-même. Il faut regarder ce qu’il est possible de faire en reliant cette fonction à l’IHM où l’utilisateur indiquera le temps de saut.

#### Package EVE\_QS

Aucune modification majeure n’est requise ici. Quelques détails permettraient d’améliorer la qualité et la stabilité du programme.

Un sémaphore pourrait être ajouté sur la mémoire partagée utilisée par l’horloge générale afin de totalement sécuriser l’accès. En pratique, on n’a pas rencontré de problème car seule l’horloge générale a la possibilité d’écrire, les autres processus ne pouvant que lire la mémoire. Il faut chercher des sémaphores multiprocessus !

La fonction EVE\_printf doit être revue pour accepter des arguments variables comme la fonction printf. De plus, il faudrait rediriger l’affichage vers le terminal utilisateur de l’IHM pour se passer du terminal en fonctionnement normal. Les fonctions d’erreurs seraient alors à modifier en conséquence.

Concernant les BAL, il faudrait ajouter des vérifications au niveau de Linux pour connaître les capacités maximales au démarrage, et ajouter des fonctions de vérifications dans les fonctions des BAL pour contrôler que l’on n’essaie pas de faire un truc impossible (typiquement, tenter un envoi alors que la BAL est pleine, ou ne pas créer trop de BAL simultanées). La fonction msgctl devrait être utile au futur développeur.

#### Modules IHM

De nombreuses choses sont à faire sur l’IHM. Des vérifications sur la création/utilisation des BAL sont à faire pour éviter de tenter des envois vers des choses inexistantes. De même, des contrôles sont à effectuer au niveau des UARTs avec plusieurs cartes en simultanées.

L’option « Garder » des UARTs n’est pas implémentée puisque les UART ne changent que lorsque l’utilisateur utilise le menu déroulant. Le changement automatique en fonction de la carte semble très difficile à implémenter. On pourrait envisager l’option de pouvoir sauvegarder dans un fichier à part le contenu de l’affichage d’un UART. On pourrait alors envisager la même chose pour sauvegarder le contenu du CAN.

Le terminal de l’utilisateur n’est pas encore fonctionnel. Il faudrait ajouter des fonctions permettant à EVE d’afficher des informations intéressantes sur son état. Il faut regarder avec la fonction EVE\_printf ce qu’il est possible de faire. De plus, les xxx\_errors() permettant d’afficher les erreurs seraient à modifier pour éviter d’inonder le terminal de messages (ce qui est arrivé souvent en tests pour l’instant). Il y a des choses à creuser de ce côté-là pour se passer du terminal Linux.

Il serait intéressant de pouvoir envoyer des données via les UARTs comme c’est prévu sur les cartes (notamment pour la Supervision). Aucune interface n’a pour l’instant été prévue. L’interface principale étant déjà très chargée et très complexe à modifier, on pourrait envisager de pouvoir ouvrir une fenêtre supplémentaire en popup permettant de transmettre des messages via un UART choisi.

L’interface d’options du CAN n’est pas encore opérationnelle. Les modifications à effectuer sont listées dans le paragraphe Module CAN.

Le menu Fichier/Edition… ne sert pour l’instant pas à grand-chose, et seule la fonction « Quitter » est implémentée. Il faudrait étudier quelles options seraient envisageables à ajouter pour améliorer le logiciel (comme les popup UART).

Le temps de match n’est pas encore affiché. Il faut faire des essais avec la Stratégie pour vérifier que le temps s’écoule correctement par rapport à la simulation et l’afficher directement sur l’interface (à partir du message CAN BROACAST\_START).

L’affichage des données QS est la grosse partie à développer. A partir du fichier de configuration data.config choisi par l’utilisateur, l’interface doit charger les modules d’affichage sur chaque onglet de carte (en tout 45 modules, 9 par carte). Il faut ensuite relier ces interfaces au reste d’EVE pour pouvoir afficher/envoyer les données QS. Sur la théorie, c’est simple, mais il y a un gros développement à réaliser.

Les interfaces d’affichage des modules de données QS sont à étudier au cas par cas pour être compatibles avec les modules QSx86. Voir la section sur les modules QSx86 pour plus de détails.

Enfin, les messages de type QS\_data\_e doivent être complétés pour correspondre aux messages supplémentaires qui doivent transiter entre les cartes et le simulateur EVE.

#### Modules QS\_EVE\_manager et QS\_EVE\_hard\_emulation

Il faut continuer de tester les fonctions en place, notamment sur d’autres codes que la Stratégie. Les initialisations sont à contrôler et à compléter.

Les structures de ports et autres variables liées au hardware du dsPIC sont à ajouter sur QS\_EVE\_hard\_emulation. Il faut voir tous les codes et envisager de tout mettre pour être compatible pour toutes les configurations du dsPIC (en étant optimiste). Les variables de registers\_save sont à contrôler également pour ne rien oublier, ni ajouter des choses inutiles.

Différentes fonctions disponibles dans les librairies du dsPIC doivent être ré-implémentées pour répondre aux codes des cartes. Il faut voir ici au cas par cas ce qui doit être fait.

#### Modules QS

Le module QS\_pwm a été implémenté mais non testé. Il faut que le code Propulsion ou Actionneurs soit opérationnel pour pouvoir effectuer des tests.

Tous les modules suivants ont été modifiés pour être compilables avec les codes, mais ne contiennent aucune fonction opérationnelle :

* QS\_adc
* QS\_ax12
* QS\_can\_over\_uart
* QS\_configBits
* QS\_configCheck
* QS\_DCMotor
* QS\_extern\_it
* QS\_ports
* QS\_qei
* QS\_qei\_on\_it
* QS\_servo
* QS\_uart
* QS\_watchdog

Ces modules doivent être développés en lien avec l’IHM qui devra afficher les valeurs attendues. Le format des messages envoyé devra être complété dans la rubrique 2.4 de ce document.

# Manuel d’installation utilisateur

L’utilisateur n’a pas d’installation très compliquée à faire pour pouvoir utiliser EVE. Il doit disposer de Linux pour pouvoir lancer le logiciel (procédure d’installation d’Ubuntu en machine virtuelle disponible au paragraphe 1.1). RabbitVCS est une alternative à TortoiseSVN pour Linux. La procédure d’installation est décrite dans le paragraphe 1.3, dans la section d’installation du développeur.

Pour utiliser EVE, vous devez disposer des codes à jour que vous souhaitez tester. L’arborescence actuelle du SVN est impérative à conserver car EVE s’occupe d’aller chercher les codes directement dans leurs répertoires. Vous ne devez pas déplacer l’exécutable pour démarrer la simulation.

Selon la dernière mise à jour, il se peut qu’il y ait des problèmes d’accès au fichier binaire. Si vous ne parvenez pas à lancer le programme, modifiez les droits d’accès du fichier en accordant les droits d’écriture/lecture à tout le monde. Si le programme ne se lance toujours pas, contactez le dernier développeur qui aura mis à jour l’exécutable.

Pour terminer, vous devez déplacer le fichier QS qui se trouve dans le répertoire QSx86 dans chaque projet de code de carte que vous désirez compiler, comme la QS normale.

# Manuel d’utilisation

Ce manuel est amené à être amélioré et fortement complété au fur et à mesure des développements sur le logiciel. L’objectif ici est de fournir une documentation exhaustive du logiciel pour faciliter sa prise en main et limiter le temps d’apprentissage. De plus, il sera possible de proposer une énorme quantité d’options au fur et à mesure des améliorations qui ne pourront être utilisées de manière viable que si la documentation adéquate est fournie.

Ce manuel est destiné à documenter uniquement les fonctionnalités qui sont considérées comme validées et fonctionnelles. N’hésitez pas à prévenir les développeurs si un point reste flou ou incomplet.

## Fonctionnement de l’interface CAN

L’interface CAN fonctionne sur le même principe que le logiciel Docklight qui est actuellement utilisé au club Robot. Le principal terminal est très simple d’utilisation. La zone d’affiche indique l’ensemble des messages CAN qui ont transité par le CAN, après un filtrage pour éviter d’être trop inondé. Le bouton « Clear » permet de nettoyer la zone d’affichage.

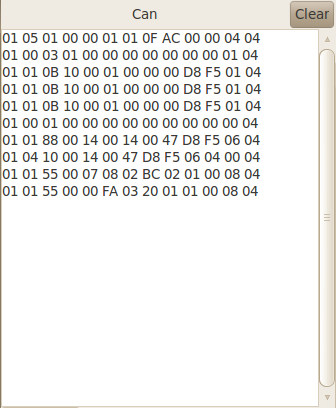


Figure 2 : Interface d’affichage du CAN

## Terminal UART

L’interface des UART est très simple d’utilisation. L’utilisateur peut choisir la sortie UART qu’il souhaite afficher dans l’une des deux zones d’affichage. La demande d’affichage reste active tant que le choix n’est pas changé (reset de la carte ou redémarrage de la simulation). Les options « Clear » permettent de nettoyer leurs affichages respectifs.

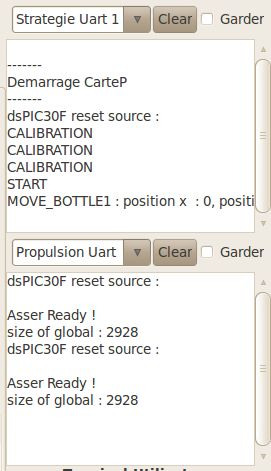


Figure 3 : Interface des terminaux UART

## Carte électronique interactive

Les différents onglets permettent d’afficher l’interface de chaque carte. Lorsque la carte est en fonctionnement, l’afficheur passe au vert, et reste grisé si la carte n’est pas active. Chaque bouton permet de simuler les vrais boutons de la carte, il n’y a qu’à cliquer dessus pour exécuter leurs commandes. Les afficheurs sur la gauche représentent les différentes LED de la carte affichée (selon l’onglet). Si l’image est grisée, la LED est éteinte, sinon l’image correspond à la couleur de la LED et signifie qu’elle est allumée.

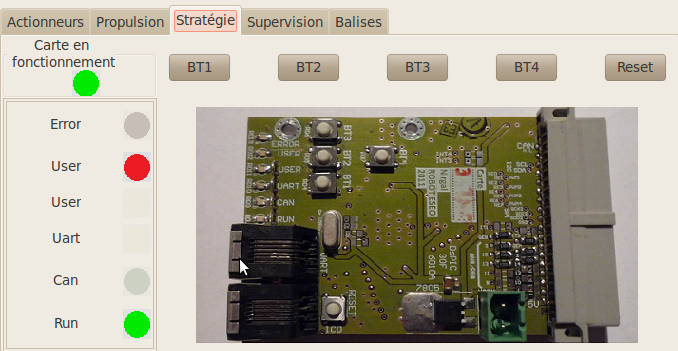


Figure 4 : Interface de la carte électronique interactive

## Options de simulation

L’affichage « Temps actuel » indique le temps courant de la simulation en millisecondes. Le bouton « PLAY » permet de démarrer une simulation. Cela équivaut à mettre les cartes électroniques sous tension sur le robot réel. Le bouton « PAUSE » permet de figer le temps de la simulation en bloquant l’exécution de tous les codes. Enfin le bouton « STOP » permet d’éteindre toutes les cartes et d’arrêter la simulation. Cela équivaut à couper le courant des cartes électroniques sur le robot réel (tel qu’un appui sur l’arrêt d’urgence). Les fonctions « Saut de temps » et « Temps de match » ne sont pas encore implémentées.

optionsDeSimulation.png

Figure 5 : Interface des options de simulation

# Table des versions et révisions

|  |  |  |  |
| --- | --- | --- | --- |
| Version/Révision | Date | Modifications | Auteurs |
| 0.1 | 23/11/2011 | Création du document | Julien FRANCHINEAU |
| 0.2 | 24/11/2011 | Ajout installation Glade pour l’installation développeur | Julien FRANCHINEAU |
| 0.3 | 23/01/2012 | Compléments sur le manuel développeur | Julien FRANCHINEAU  François EVEN |
| 0.4 | 24/01/2012 | Compléments sur les manuels d’utilisation | Julien FRANCHINEAU |
| 0.5 | 25/01/2012 | Compléments sur le manuel d’utilisation et QS\_DATA | Julien FRANCHINEAU  François EVEN |
|  |  |  |  |
|  |  |  |  |