MANUEL D’INSTALLATION DE LA LIBRAIRIE MATERIEL SOUS QT

**ASCENSEUR A POISSONS**



**SOMMAIRE**

INTRODUCTION page 3

ARCHITECTURE DE L’INTERFACE LOGICIELLE page 3

COMPOSITION DE L’INTERFACE LOGICIELLE page 3

INSTALLATION DE MSVC SUR QT page 3

**INTRODUCTION**

Les fonctions de l'interface logicielle permettent à des programmeurs de réaliser des applications pour Windows utilisant les cartes CAN NSI. Ces fonctions permettent de configurer les paramètres du bus CAN, de définir des messages CAN, segmentés ou non, puis de les émettre et de les recevoir. Toutes les fonctions de l’interface logicielle sont définies en Langage C ANSI dans la librairie : **NSICANEX.DLL**.

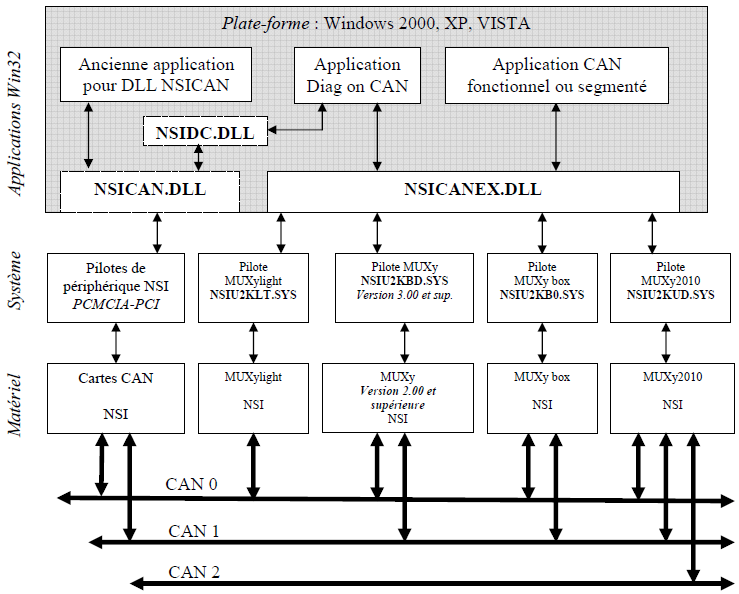
Ces fonctions sont exécutées par une DLL qu'une ou plusieurs applications peuvent appeler simultanément. Cette DLL est identique pour toutes les plateformes Windows supportées.

L'interface logicielle permet à l'utilisateur de faire abstraction aussi bien du système d'exploitation que du type de carte CAN utilisé.

**ARCHITECTURE DE l’INTERFACE LOGICIELLE**

**Application Qt**

**NSICANEX.LIB**



***Extrait du document Guide installation interface logicielle NSI.pdf***

La DLL **NSICANEX**, est l’élément intermédiaire permettant de faire communiquer les ordres de l'application vers la carte CAN (MUXy2010 dans notre cas) au travers du pilote de périphérique **NSIU2KUD.SYS**

Le rôle de chaque fichier est expliqué dans la page suivante.

**COMPOSITION DE l’INTERFACE LOGICIELLE**

L’interface logicielle est composée de 3 types de fichier :

* ***NSICANEX.DLL***

Le fichier **NSICANEX.DLL** (Dynamic Link Library) exporte toutes les fonctions de l'interface logicielle. Les applications doivent appeler ce fichier DLL pour utiliser l'interface logicielle. La DLL NSICANEX communique les ordres de l'application vers la carte CAN au travers d'un pilote de périphérique. Pour lier l’application Qt avec ce fichier DLL, nous allons utiliser un fichier **LIB** qui définit les points d’entrées de chaque fonction exportée.

* ***NSICANEX.LIB***

Le fichier **NSICANEX.LIB** permet de lier une application avec **NSICANEX.DLL** lors de la compilation de l’application. Ce fichier peut-être généré à partir du fichier DLL grâce à un utilitaire livré avec l’outil de développement utilisé, mais dans notre cas, il est fourni par le constructeur. Par contre il est spécifique aux environnements de développement Microsoft, donc l’utilisation d’un compilateur compatible est nécessaire. Le fichier LIB doit être copié dans le répertoire du projet de l'application et inséré dans le projet Qt. Nous détaillerons cette manipulation.

* ***HEADERS (.h)***

Les fichiers .h donnés par le constructeur définissent en langage C ANSI les prototypes des fonctions de l'interface logicielle. Le fichier principal est **CANPCEX.H**. Celui-ci inclus deux autres fichiers qui séparent la déclaration des structures et des constantes (**CANDEFEX.H**) de la déclaration des prototypes des fonctions (CANPROEX.H). Ces trois fichiers doivent être copiés dans le répertoire du projet de l'application et le fichier CANPCEX.H doit être inclus dans le code source par la directive suivante :

#include "Canpcex.h"

Les fichiers .h de l’interface logicielle contiennent des directives de compilation dont la syntaxe est spécifique aux outils Microsoft. Celles-ci doivent impérativement être modifiées en fonction du compilateur utilisé.

**INSTALLATION DE MSVC SOUS Qt**

* ***Installation du compilateur***

Cette section décrit l’installation du compilateur C++ de Microsoft nommé MSVC (Microsoft Visual Compiler) et son intégration au sein de l’environnement de développement de Qt.

Le moyen le plus simple d’ajouter ce compilateur à Windows est d’installer Microsoft Visual Studio 2010, pour avoir la version 10 du compilateur ou Microsoft Visual Studio 2012 pour avoir la version 11. A notre niveau, les différences entre ces deux versions n’ont guère d’importance.

Une fois l’environnement de développement de Microsoft installé, le fichier d’initialisation du compilateur est installé par défaut dans :

*C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\vcvarsall.bat x86*

Le type de compilateur peut-être ajouté en argument :

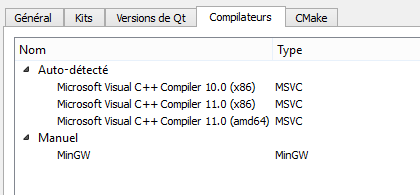
* "x86" (valeur par défaut)
* "amd64"
* "x86\_amd64"
* ***Intégration du compilateur sur Qt***

Maintenant il faut ajouter un Kit sous Qt. Les kits permettent de paramétrer un environnement de compilation et de choisir un compilateur, un debugger, le type de périphérique, la version de Qt à utiliser… etc

Plusieurs kits peuvent être définis pour un même projet et l’utilisateur peut switcher d’un kit à l’autre à n’importe quel moment.

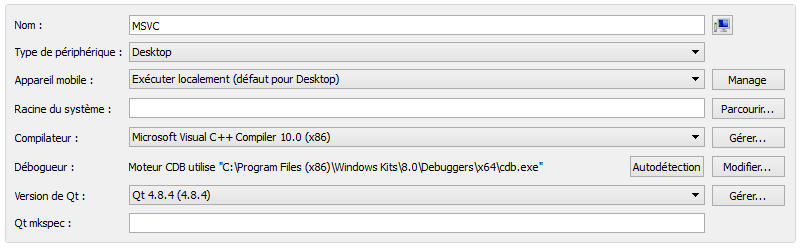
On va d’abord vérifier que le compilateur est bien reconnu par Qt

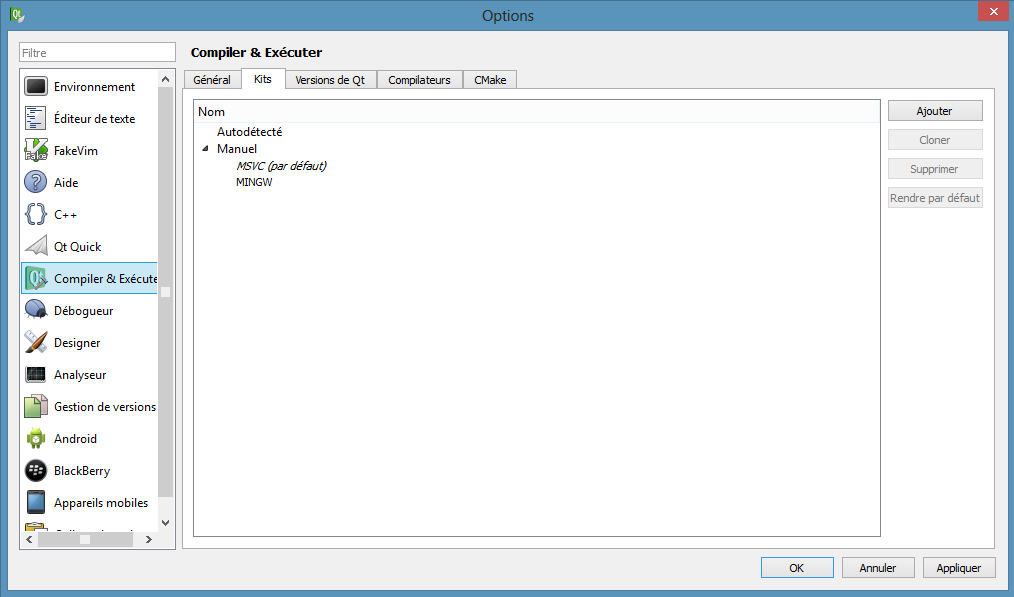
*Sous Qt > Outils > Options > Compiler et exécuter, onglet compilateurs :*



Normalement le compilateur devrait être automatiquement auto-détecté par Qt. Il faut vérifier que le chemin d’initialisation est le même que le précédent avec le bon argument.

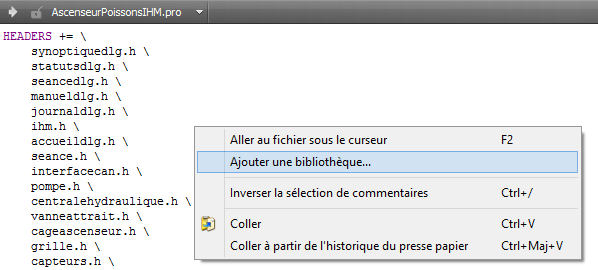
Maintenant il faut configurer le kit de développement. Il doit être défini comme ceci :



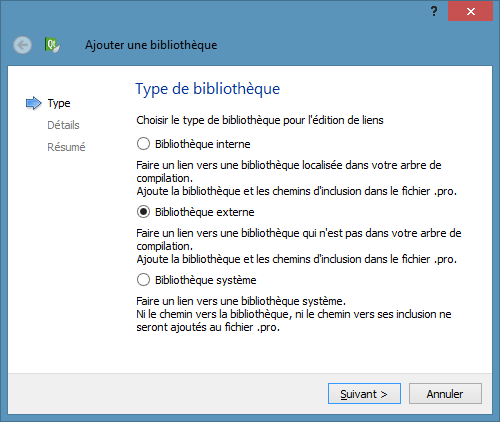
Le kit est maintenant configuré, il apparaît dans la liste des kits disponibles pour le développement :

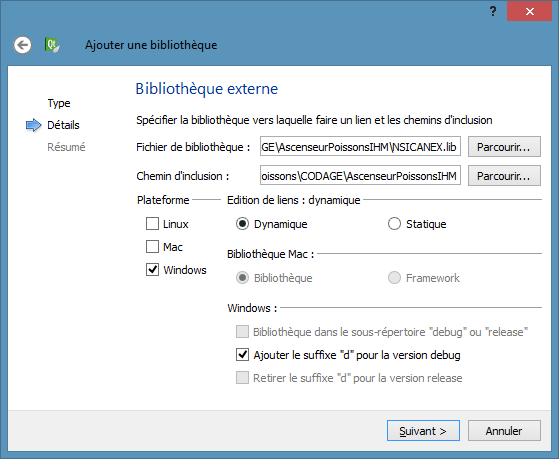
**INTEGRATION DE LA LIBRAIRIE SOUS QT**

Pour ajouter une librairie externe à un projet sous Qt, il faut le faire manuellement à partir d’un menu accessible via le fichier .pro du projet :



Ensuite il faut choisir le type de bibliothèque que l’on veut intégrer. Dans notre cas il s’agit d’une librairie externe :





Une fois le type de bibliothèque choisi, il faut spécifier le répertoire où se trouve la librairie pour que Qt aille la chercher, dans notre cas, elle se trouve à la racine du projet.

Une fois la librairie ajoutée, l’utilitaire de Qt ajoute le code suivant dans le fichier principal du projet :

win32: LIBS += -L$$PWD/ -lNSICANEX

INCLUDEPATH += $$PWD/

DEPENDPATH += $$PWD/

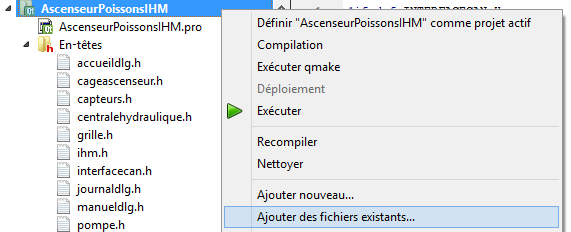
**PWD** désigne le répertoire courant et **NSICANEX** est la librairie utilisée.

* ***Ajout des fichiers d’en-têtes***

Maintenant que la librairie est ajoutée au projet, il faut insérer les 3 headers principaux de la librairie au sein du projet. Les 3 fichiers suivants doivent être mis à la racine du projet :

* CANDEFEX.H
* CANPCEX.H
* CANPROEX.H

Ensuite, on doit ajouter manuellement les fichiers dans l’arborescence du projet :



Puis inclure quatres fichiers d’en-têtes dans notre classe d’interface matérielle :

#include <windows.h>

#include <conio.h>

#include <stdio.h>

#include <Canpcex.h>

* ***Description des fichiers d’en-têtes***

**WINDOWS.H**

Windows.h est un header spécifique au système exploitation Windows pour la programmation en langage C contenant toutes les déclarations des fonctions de l’API Win32 de Windows, les macros couramment utilisés ainsi que des types de données exclusifs à cette API comme **\_\_int32** et **\_\_int64**.

**CONIO.H & STDIO.H**

Conio.h & stdio.h sont deux en-têtes standards pour la gestion des opérations d’entrées/sorties. La première est spécifique au système windows, plus particulièrement pour la console alors que la seconde fait partie de la bibliothèque standard C.

**CANDEFEX.H**

Ce fichier contient les définitions des structures de données utilisées par les fonctions de l'interface CAN, les valeurs constantes utilisées comme paramètres, et les codes de retour des fonctions de l'interface CAN.

**CANPCEX.H**

Le symbole \_CANAPI définit la convention d'appel des fonctions de la DLL. Il est utilisé dans le fichier CANPROEX.H. Ce symbole spécifie que les fonctions doivent être importées d'une DLL et qu'elles utilisent la convention d'appel C.

#define \_CANAPI \_\_declspec(dllimport) \_\_stdcall

#define \_CANAPI \_\_declspec(dllexport) \_\_stdcall

Cette directive est spécifique au compilateur de Microsoft, elle ne peut donc pas être utilisée avec un autre compilateur.

**CANPROEX.H**

Ce fichier contient les définitions des fonctions de l'interface CAN exportées par **NSICANEX.DLL**. Les fonctions sont prototypées de cette manière :

#ifdef \_\_cplusplus

extern "C" {

#endif

TYPE \_CANAPI fonction(…);

TYPE \_CANAPI fonction2(…);

…

#ifdef \_\_cplusplus

}

#endif

Le symbole **\_CANAPI** est défini dans le fichier **CANPCEX.H**.

**extern "C"** permet d'indiquer que les fonctions déclarées à l'intérieur utiliseront une convention d'appel C. Le but est de pouvoir appeler une bibliothèque C depuis du C++.