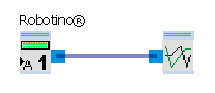
**LECTURE DES CAPTEURS ANALOGIQUES/DIGITAUX (à distance)**

1. **Tester les capteurs**

Afin de savoir si les capteurs fonctionnent, il faut utiliser le logiciel RobotinoVIEW qui va nous permettre de tester toutes les entrées-sorties analogiques/numériques de la carte de Robotino.

Voici comment nous procéder :

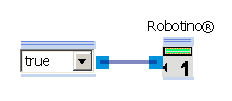
***Récupérer la valeur d’une entrée analogique :***



Pour se faire, la première box (correspondant à l’entrée analogique 1) se trouve dans la bibliothèque à droite de l’écran, rubrique : **Robotino -> Connecteur d’E/S -> Entrée analogique #1**

La deuxième box (correspondant à un oscilloscope virtuel) se trouve dans la rubrique : **Bibliothèque de blocs de fonction -> Affichages -> Oscilloscope**

***Forcer la valeur d’un des sorties TOR :***



La première box correspond à une constante, se situant dans la rubrique : **Bibliothèque de blocs de fonction -> Générateur -> Constante** (Par défaut la constante est un nombre (float) mais en la reliant à la box elle devient un booléen).

La deuxième box correspond à la sortie TOR 1, se situant dans la rubrique : **Robotino -> Connecteur d’E/S -> Sortie TOR #1**

Le fichier de démo « *Test E-S carte* » réalisant le test de la sortie digitale 1 et l’entrée analogique 1 est situé dans le répertoire APP2018/Arnaud/ ExempleLectureCapteurAnalogique sur les robotino.

1. **Interfaçage C en Java**

La solution pour pouvoir utiliser les capteurs de Robotino est d’utiliser le langage C. Mais pour éviter de refaire tout le travail effectué en Java, la solution consiste à créer une interface C que l’on mettra dans une bibliothèque et qui sera appelé dans le programme Java. C’est donc cela que je vais expliquer. Je me suis aider du tutoriel sur Développons en Java :

(<https://www.jmdoudoux.fr/java/dej/chap-jni.htm>)

Dans un premier temps, il faut déclarer la méthode native. C’est assez simple, il suffit, dans le code source Java, de déclarer la signature de la méthode avec la définition de la bibliothèque native.

Ce code est disponible sur Robotino dans le dossier : APP2018/Arnaud/ ExempleLectureCapteurAnalogiqueWindows/CptIndW.java

Voici le programme :

**public** **class** CptIndW {

//Déclaration de la méthode native en JAVA, elle contient le numéro de Pin où l’on souhaite lire la valeur et les données de connexion dans la chaine de caractère hostname

**public** **native** **float** valeurCptIndW(**int** numPin, String hostname);

**static** {

//Appel de la librairie nécessaire pour utiliser les objets de Robotino

System.*loadLibrary*("rec\_robotino\_com\_c");

System.*loadLibrary*("CptIndW"); //Notre librairie pour appeler la méthode en C qui nous retournera la valeur des capteurs à Induction

}

}

Puis nous déposons générons le fichier d’en-tête avec l’outil javah sous windows (présent dans le dossier jdk1.6). Il faut faire cette génération sur le PC en utilisant la même version Java que Robotino afin d’éviter tout problème de version par la suite.

La version Java sur Robotino peut s’obtenir avec la commande suivante (en l’exécutant en local à partir de putty):

java -version

On peut voir que la version est la 1.6.0.0

On compile donc le fichier puis nous générons le code du fichier d’en-tête avec les commandes en se plaçant dans le dossier contenant le fichier java:

javac *nomClasse.java*

javah -jni *nomClasse*

Le fichier d’en-tête (ici : CptIndW.h) généré est le suivant :

/\* DO NOT EDIT THIS FILE - it is machine generated \*/

#include <jni.h>

/\* Header for class CptIndW \*/

#ifndef \_Included\_CptIndW

#define \_Included\_CptIndW

#ifdef \_\_cplusplus

extern "C" {

#endif

/\*

\* Class: CptIndW

\* Method: valeurCptIndW

\* Signature: (ILjava/lang/String;)F

\*/

JNIEXPORT jfloat JNICALL Java\_CptIndW\_valeurCptIndW

(JNIEnv \*, jobject, jint, jstring);

#ifdef \_\_cplusplus

}

#endif

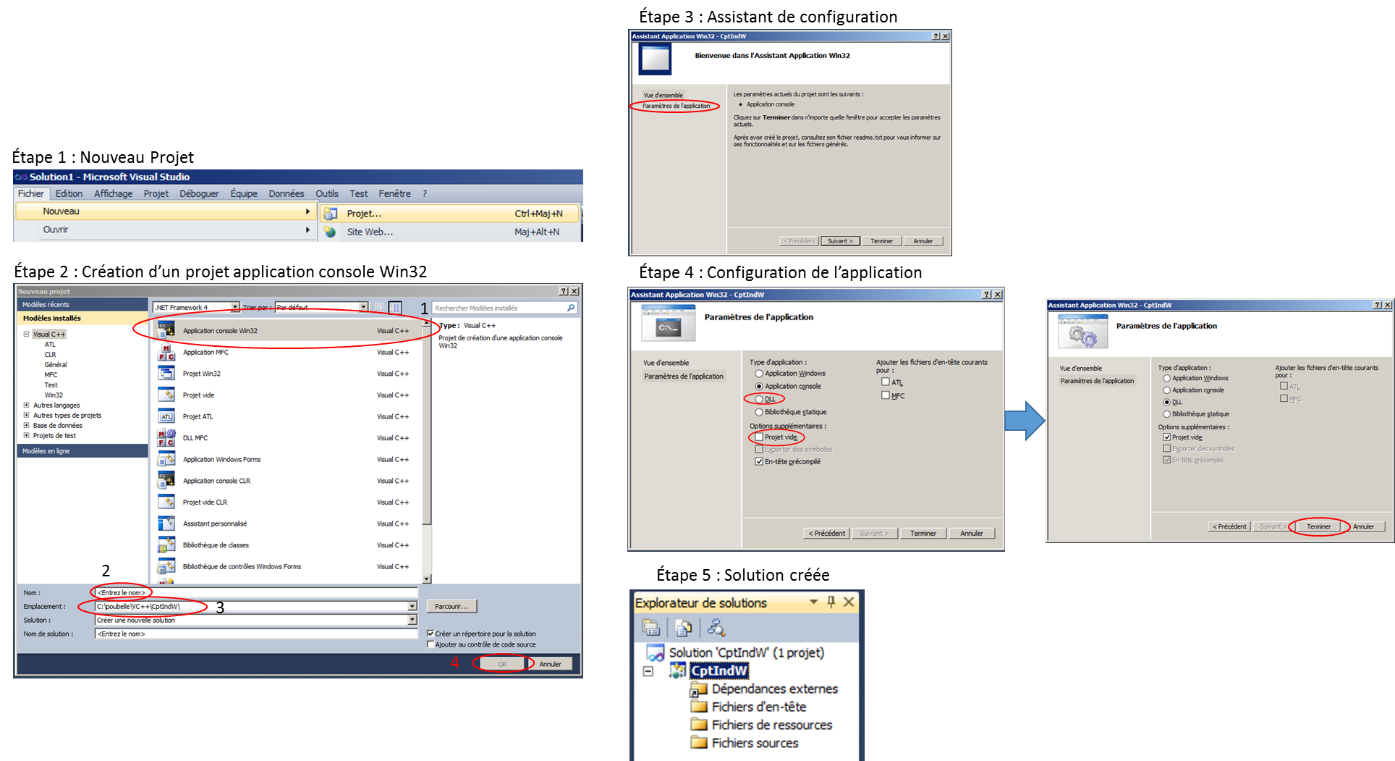
#endif

Maintenant, il faut réaliser l’écriture du code natif en C. La fonction est la suivante :

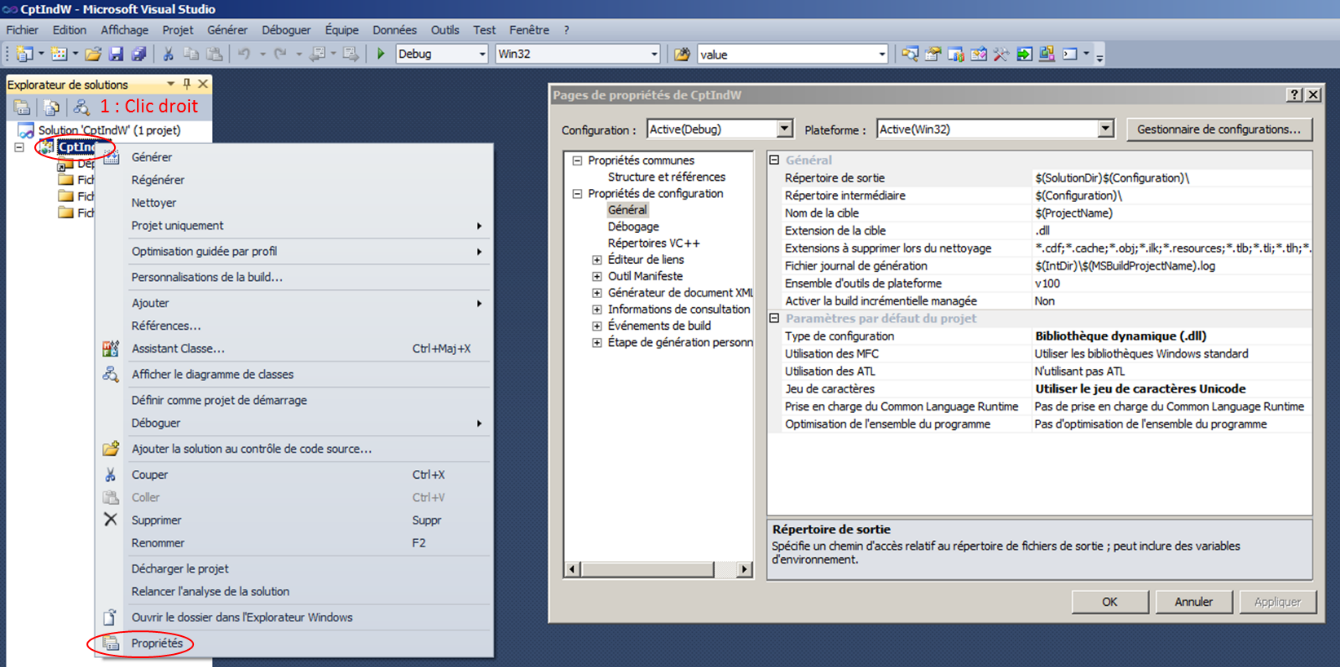
Il est placé dans le dossier APP2018/Arnaud/ ExempleLectureCapteurAnalogiqueWindows/CptIndW.h

Puis il faut écrire notre fonction en C avec le logiciel visual studio. Pour cela on créer un nouveau projet et il faut choisir une « application console Win32 » en C++. Vous choisissez un nom à votre solution, dans notre cas, nous avons choisi CptIndW et attention, l’emplacement du fichier doit être dans « C:\poubelle\VC++\*NomSolution*\ » pour des raisons de droits d’écriture et d’accès.

Vous validez et il s’affiche ensuite un panneau d’aide de configuration. Allez sur « **Paramètre de l’application** » et choisissez **DLL** et **projet vide** car nous souhaitons développer une librairie Windows.

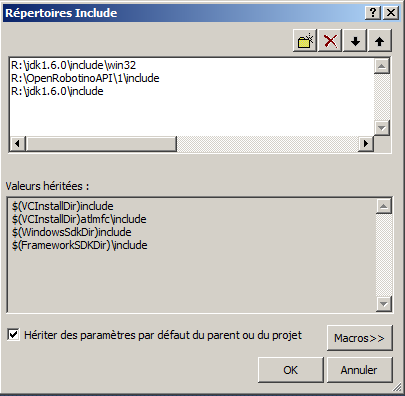
Enfin cliquer sur Terminer. En résumé voici les images pour vous aider à suivre les étapes précédentes :

Maintenant que la solution est créée, il faut configurer correctement la solution de telle sorte qu’à la compilation du fichier, nous ayons bien toutes les références et les librairies dont il a besoin.

Pour cela, clic droit sur votre solution et propriété :

Un panneau s’ouvre avec les différentes configurations possibles.

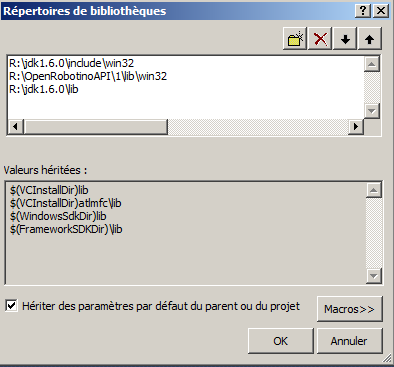
Voici les chemins que l’on doit ajouter dans les différents sous-menus des propriétés de configuration.

Tout d’abord, on va dans les répertoires VC++ et on ajoute dans les répertoires Include, les chemins suivant :

R:\jdk1.6.0\include\win32;

R:\OpenRobotinoAPI\1\include;

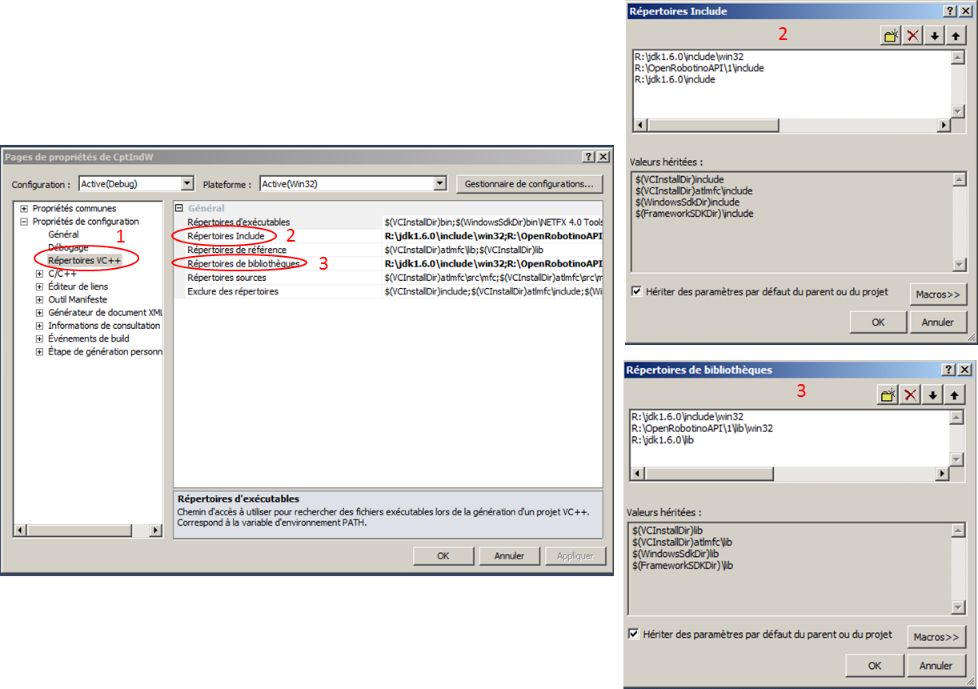
R:\jdk1.6.0\include;

Et dans les répertoires de Bibliothèques on ajoute :

R:\jdk1.6.0\include\win32;

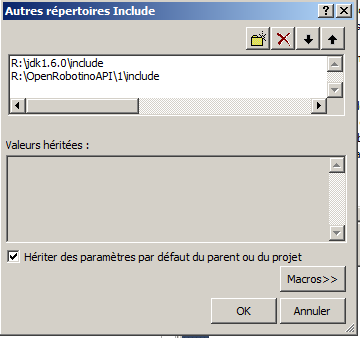
R:\OpenRobotinoAPI\1\lib\win32;

R:\jdk1.6.0\lib;

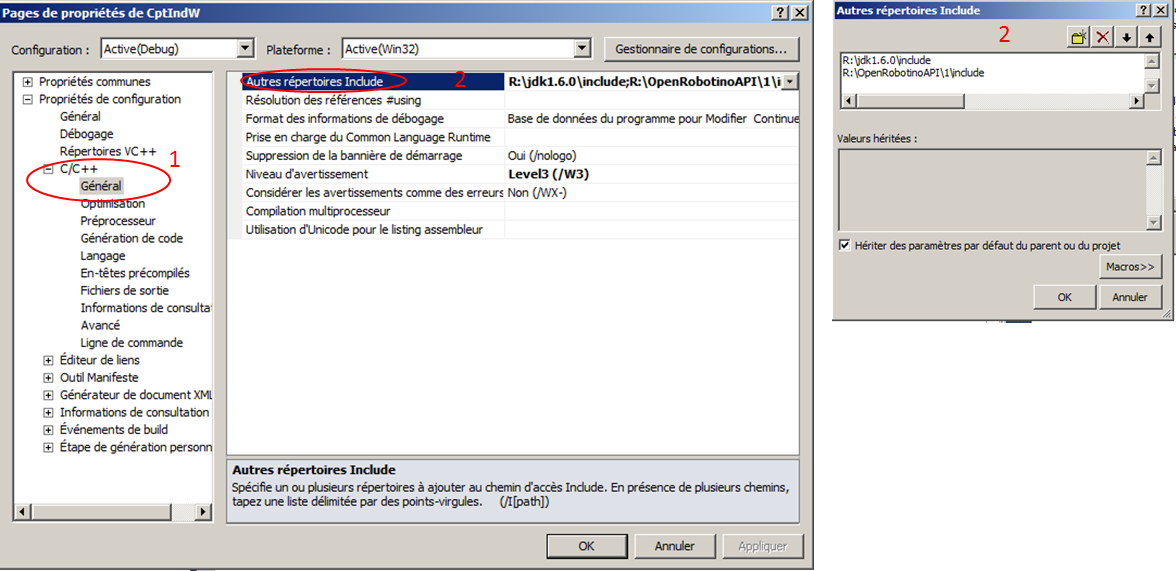
Donc en résumé dans la page de configuration des répertoires VC++ nous avons :

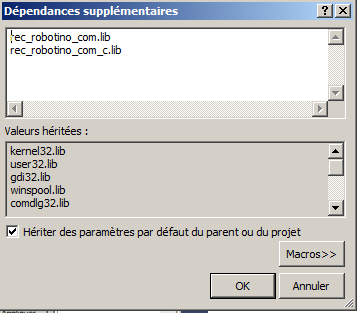
Ensuite, il faut se rendre dans les propriétés des fichiers C/C++. Pour faire apparaître ce menu, il faut au préalable créer un nouveau fichier C/C++.

Puis dans l’onglet Général Autres répertoires Include, on ajoute les chemins suivants :

R:\jdk1.6.0\include;

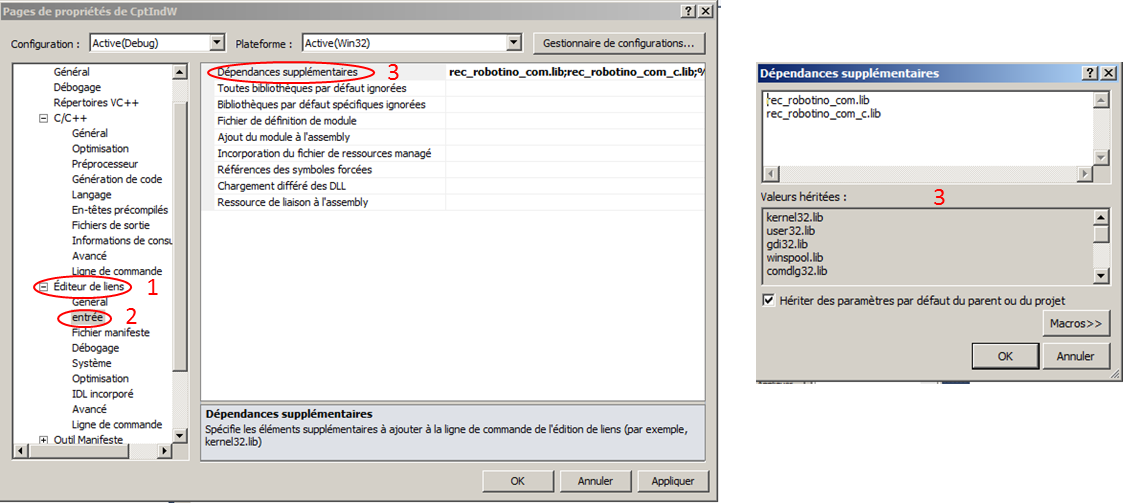
R:\OpenRobotinoAPI\1\include;

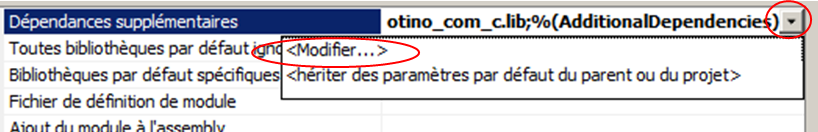
Soit en résumé sur cette page :

Enfin dans l’Éditeur de lien dans le sous menu entrée, on définit les dépendances supplémentaires qui sont :

rec\_robotino\_com.lib;

rec\_robotino\_com\_c.lib;

Soit :

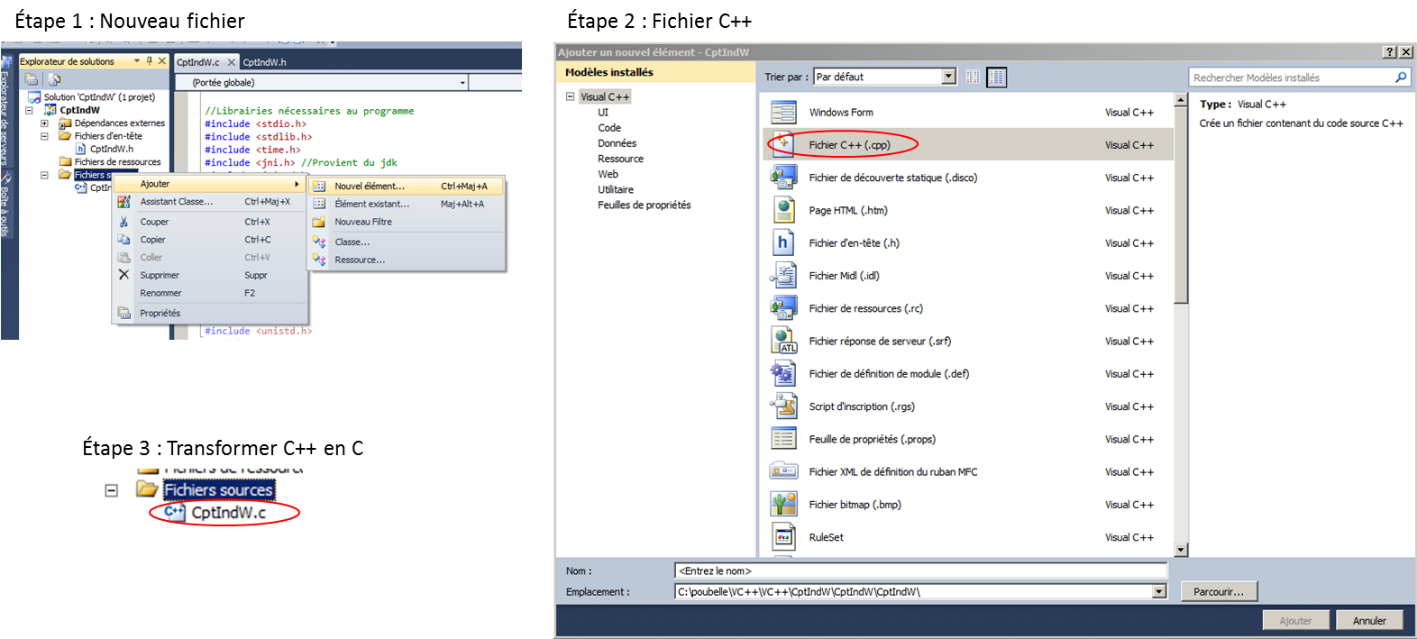
Nous avons oublié de préciser que pour pouvoir ajouter un chemin dans un des répertoires, il suffit de cliquer sur la petite flèche à droite et ensuite modifier, comme dans l’exemple ci-dessous :

Une fois que cela est effectué, il faut ajouter le fichier .h qui a été préalablement construit avec l’outil jni. Pour l’ajouter dans les fichiers d’en-tête, il suffit de faire un clic droit sur le dossier « Fichiers d’en-tête » puis Ajouter et Élément existant. Ensuite il faut aller chercher le fichier .h dans le bon dossier.

À présent, nous pouvons créer notre fichier .c dans les fichiers sources. Pour cela, clic droit sur « Fichiers sources » puis Ajouter et Nouvel élément. Choisir un Fichier C++ et on entre un nom de fichier. Dans notre cas CptIndW. Puis Ajouter.

Une fois que cela est fait, pour pouvoir écrire le fichier en C, l’astuce consiste à remplacer l’extension .cpp par .c.

En résumé voici les étapes en images :



Le fichier C pour notre application est le suivant :

// Copyright (C) 2004-2008, Robotics Equipment Corporation GmbH

#include "CptIndW.h" //Fichier d'entete genere

//Librairies nécessaires au programme

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <jni.h> //Provient du jdk

#include <jni\_md.h>

#include "rec/robotino/com/c/Com.h"

#ifdef WIN32

#include <windows.h>

// \_getch

#include <conio.h>

#else

// getchar

#include <stdio.h>

// usleep

#include <unistd.h>

#endif

float valeurCapteurInduction\_c(int iNumPin, const char\* hostname){

//Déclaration et initialisation des variables

ComId com;

AnalogInputId pinCapteur;

int i;

boolean connectSetCom = 0;

boolean connectPin = 0;

boolean tempsInit = 1;

float fvalue = -1.0;

int nbValueMoyenne = 10;

int tailleFenetreValue = 20; //Nombre de mesures à acquérir

float value[20]; //Ici on initialise notre tableau à la taille de la fenêtre (mettre le même nombre que tailleFenetreValue car la variable n'est pas reconnue dans les crochets)

//Connection à Robotino (via adresse IP)

com = Com\_construct();

Com\_setAddress(com,hostname);

if( FALSE == Com\_connect(com) )

{

fprintf(stderr, "Error on connect\n");

return -1.0;

}

else

{

char addressBuffer[256];

Com\_address( com, addressBuffer, 256 );

printf( "Connected to %s\n", addressBuffer );

}

//Construction de l'objet analogique : numero du pin allant de 0 à 7 (AIN1 -> AIN8)

pinCapteur = AnalogInput\_construct(iNumPin-1);

//On associe notre objet analogique à notre interface de communication

connectSetCom = AnalogInput\_setComId(pinCapteur, com);

if(connectSetCom == 0){

fprintf(stderr, "Error on connectSetComId pin\n");

return -1.0;

}

//On associe le numéro de pin à notre objet

connectPin = AnalogInput\_setInputNumber(pinCapteur, iNumPin-1);

//Initialisation capteur

while(tempsInit){

if(connectPin == 0){

printf( "Erreur Pin Connexion \n" );

break;

}

//Permet d'initialiser notre capteur : on sort de la boucle une fois que l'on lit une valeur > 0

if(AnalogInput\_value(pinCapteur) > 0.001){

tempsInit = 0;

}

}

//Récuperation valeur capteur

for(i=0;i<tailleFenetreValue;i++){

value[i] = AnalogInput\_value(pinCapteur);

}

//Moyenne des X dernières valeurs afin d'éliminer des erreurs de mesures

for(i=nbValueMoyenne;i<tailleFenetreValue;i++){

if(fvalue == -1.0){

fvalue = 0;

}

fvalue = fvalue + value[i];

}

//Valeur moyenne du capteur induction

fvalue = fvalue/(tailleFenetreValue-nbValueMoyenne);

//Si on souhaite afficher la valeur depuis le programme C

//fprintf(stdout, "Valeur Induction : %f \n", fvalue);

//On détruit les objets associés à notre interface de communication

AnalogInput\_destroy(pinCapteur);

Com\_destroy(com);

return fvalue;

}

//Fonction qui nous renvoie la valeur du capteur d'induction passé en paramètre d'entrée :

//Permet de "transferer"/"traduire" le code C en Java

//Définition du nom de la fonction : JNIEXPORT typeValeurRetournée JNICALL Java\_nomClasse\_nomMéthodeJava

JNIEXPORT jfloat JNICALL Java\_CptIndW\_valeurCptIndW

(JNIEnv \*env, jobject obj, jint iNumPin, jstring hostname){//Fonction en C qui va récupérer la valeur du capteur d'induction passé en paramètre d'entrée

//Conversion jstring en const char\*

const char\* nativeString = (\*env)->GetStringUTFChars(env, hostname, NULL);

if(nativeString == NULL){

return -1;

}

else{

return valeurCapteurInduction\_c(iNumPin, nativeString);

//Libération de la chaine de caractères

(\*env)->ReleaseStringUTFChars(env, hostname,nativeString);

}

}

Nous attirons l’attention sur le fait qu’en C il n’est pas possible d’utiliser le format STRING. Il faut donc effectuer la conversion en char. Cependant dans notre cas, il s’agit d’un pointeur Const Char\* car la fonction de connexion de Robotino (SetComAdress) demande ce type.

Pour cela, on doit donc utiliser la fonction GetStringUTFChars de jni.h pour convertir la chaine de caractère et il faut penser à la libérer en utilisant la ReleaseStringUTFChars car autrement nous avons un problème à la compilation.

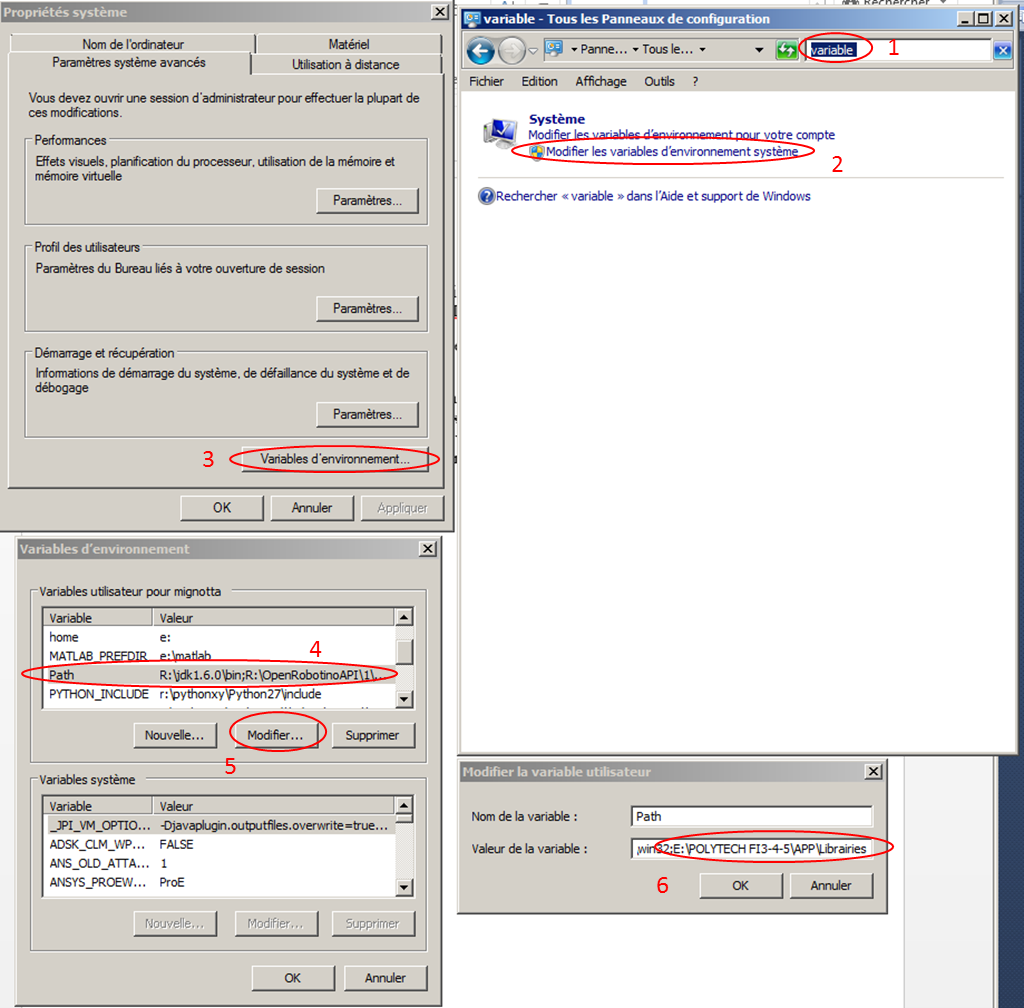
Nous plaçons ce fichier sur Robotino dans le dossier APP2018/Arnaud/ ExempleLectureCapteurAnalogiqueWindows/CptIndW.c

Ensuite il faut générer la solution et il nous créer automatiquement la librairie .dll dans le dossier C:\poubelle\VC++\ CptIndW\CptIndW\Debug.

Nous plaçons notre librairie dans le dossier sur Robotino afin que vous puissiez la retrouver dans le dossier ExempleLectureCapteurAnalogiqueWindows/Librairies/CptIndW.dll

On récupère donc cette librairie pour la mettre dans un autre dossier. Choisir un dossier qui va contenir toutes les librairies que l’on souhaite créer. Par exemple, nous l’avons appelé Librairies. Cela est très important car il faut ajouter ce dossier dans la variable d’environnement Path. Pour cela, il suffit d’accéder au panneau de configuration et de rechercher variable d’environnement puis de cliquer sur Variables d’environnement et chercher (ou créer si elle n’existe pas) la variable Path et ajouter le chemin du dossier.

Soit en image :



Nous pouvons maintenant utiliser notre librairie et la fonction C dans l’environnement Eclipse en appelant la méthode et la librairie de cette manière :

//Pour l'induction

**public** **native** **float** valeurCapteurInduction(**int** numPin, String hostname);

**static** {

System.*loadLibrary*("CptIndW");

}