# COMPILATEUR C CC5X POUR PIC UTILISÉ AVEC MPLAB

Adresse Internet du site de l'éditeur : http://www.bknd.com/cc5x/index.shtml

## 1) GÉNÉRALITÉS

Ce document se réfère à la version 3.2. Il est basé sur un autre document écrit pour une version antérieure. Il subsiste peut être quelques incohérences.
Une version gratuite du compilateur, avec quelques limitations, peut être téléchargée sur le site de l'éditeur.

### 1.1) UTILISATION

Le compilateur CC5x est constitué d'un exécutable CC5X.exe lancé par une ligne de commande. Cette ligne peut être générée par un environnement de développement intégré (ex : MPLAB). CC5X s'interface parfaitement avec MPLAB. Il y a interactivité entre la fenêtre des messages et la ou les fenêtres d'édition (un double clic sur un message d'erreur renvoie à la ligne de l'erreur dans la fenêtre d'édition).

Le compilateur CC5x peut être utilisé avec tous les PIC d'entrée de gamme et de milieu de gamme. Un compilateur spécifique est disponible pour les PIC haut de gamme de la série 18C (CC8E).

Le compilateur peut produire seul un fichier exécutable .hex à partir d'un seul fichier source en C (avec d'autres fichiers inclus éventuellement). Il n'y a pas de ce cas utilisation d'un éditeur de liens. Cette façon de travailler, peu courante pour un compilateur, a été retenue pour pouvoir produire un fichier exécutable compact et rapide à l'exécution.

Avec CC5X, il est aussi possible de réaliser des compilations séparées, avec un projet constitué de plusieurs fichiers source en C, suivies d'une édition de liens. Dans ce cas, l'édition de liens s'effectue avec l'éditeur de liens MPLINK fourni avec MPLAB. Le fichier exécutable produit est moins optimisé que lorsqu'il n'y a qu'un seul fichier source. De plus, il existe certaines limitations, notamment dans l'utilisation des fonctions.
*Le présent document ne traite pas de l'édition de liens avec MPLINK. Voir CC5X User's Manual §6.8 Linker support.*

Pour ce compilateur, la priorité de la conception n'a pas été d'assurer une conformité avec le C ANSI, mais d'obtenir un code compact et rapide en faisant le meilleur usage possible des ressources limitées du PIC. Certains types de données n'occupent pas en mémoire la taille définie par le C ANSI. *Voir §3.*
Le compilateur dispose d'extensions au C ANSI pour gagner de la place en mémoire (type bit pour les données) et pour s'adapter aux particularités du µC (possibilité de manipuler un bit d'un port avec une seule instruction machine, etc.).

## 1.2) PRINCIPALES LIMITATIONS PAR RAPPORT AU C ANSI

**Seuls les tableaux à une dimension sont acceptés**. Il est possible de remplacer les tableaux multidimensionnels par des structures, avec cependant un seul index pouvant être une variable (voir manuel de l'utilisateur § 2.2 Arrays, structures and unions).

**Les variables globales initialisées ne sont pas acceptées.**

**Le compilateur a des possibilités limitées pour allouer des variables temporaires** → A cause de cela les instructions complexes ont souvent à être réécrites en décomposant en instructions plus simples.

Par exemple, avec la version d'évaluation, il est impossible de compiler la ligne suivante :

```
ResultatCAN1+=ADRESH*256+ADRESL ;
```

La décomposition suivante est acceptée :

```
ResultatCAN1 = ADRESH*256;
ResultatCAN1 += ADRESL;
ResultatCumulCAN1+=ResultatCAN1; // 1 variable de plus nécessaire
```

Pour plus de détail, voir le § WRITING CODE THAT CAN BE COMPILED BY CC5X en fin du fichier ReadMe.txt dans le dossier d'installation du compilateur.

La version d'évaluation comporte des limitations supplémentaires :

- pas d'optimisation du code produit (mais évaluation du gain apporté par l'utilisation du compilateur non bridé)
- limitation de la taille du code produit (1024 mots max en mémoire programme par code objet résultat de la compilation d'un fichier source). *Le PIC16F84 ne dispose que de 1024 mots de mémoire programme.*

## 1.3) PRINCIPALES AMÉLIORATIONS PAR RAPPORT AU C ANSI

Le compilateur dispose de particularités pour minimiser l'occupation mémoire :

- fusion des données constantes pour diminuer l'occupation mémoire. Ceci est transparent à l'utilisateur. Voir User's Manual page 28. *Non décrit dans ce document*
- concaténation de 2 codes ASCII 7 bits pour enregistrement dans un mot mémoire de 14 bits. Voir User's § 2.5 Storing 14 bit data. *Non décrit dans ce document*
- types de données à virgule fixe (possibilité rare)
- nombreuses directives de compilation introduites avec #pragma

Pour une utilisation simple en lycée technique, la connaissance de ces améliorations n'est pas utile.

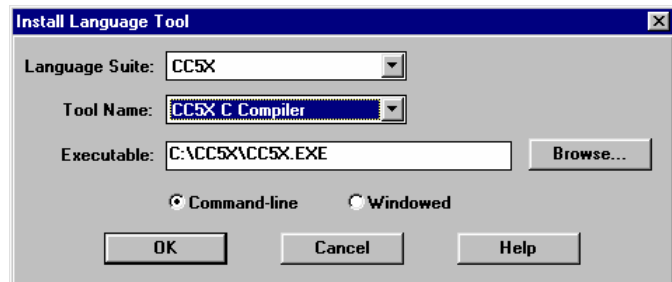
## 2) INSTALLATION & UTILISATION AVEC MPLAB 5.X

L'installation automatique du logiciel permet de décompacter le fichier d'installation et de placer tous les fichiers sur le disque dur. Il faut ensuite procéder aux opérations suivantes :

- copier les fichiers TLCC5X.INI et CC5X.MTC dans le dossier d'installation de MPLAB (d'autres fichiers .ini et .mtc existent déjà dans le dossier)
- configurer MPLAB

La configuration de MPLAB concerne la suite logicielle. *Voir aussi le document MPLAB 5.x §5.*

La configuration s'effectue avec la commande Project / Install Language Tool. Il suffit d'indiquer le nom de l'exécutable avec son chemin d'accès.




Pour une utilisation de plusieurs fichiers source en langage C, il faut utiliser l'éditeur de liens MPLINK de Microchip dont il faut donner le chemin d'accès (Tool Name : MPLINK) .Il est aussi possible de lier des fichiers objet créés avec CC5X et MPASM, l'assembleur de Microchip. Le chemin d'accès de ce dernier doit alors être défini.

## 3) CRÉATION ET CONFIGURATION D'UN PROJET AVEC MPLAB

### 3.1) DIFFÉRENCE ENTRE PROJETS AVEC ET SANS ÉDITION DE LIENS

La création et la configuration d'un projet sont différentes selon qu'on souhaite ou non une édition de liens.

L'édition de liens est utilisée que dans le cas où plusieurs fichiers source sont compilés séparément. L'assemblage (dernière phase de la compilation d'un fichier source) produit un fichier objet relogeable, sans adresses absolues. L'édition de liens est alors réalisée avec MPLINK.

Lorsqu'un seul fichier source est utilisé (éventuellement avec des fichiers inclus), la compilation + l'assemblage produit un fichier objet absolu directement utilisable pour la programmation et le débogage. *Cette absence d'utilisation de l'éditeur de liens est exceptionnelle. L'immense majorité des compilateurs utilise l'édition de liens, même avec un seul fichier source.*

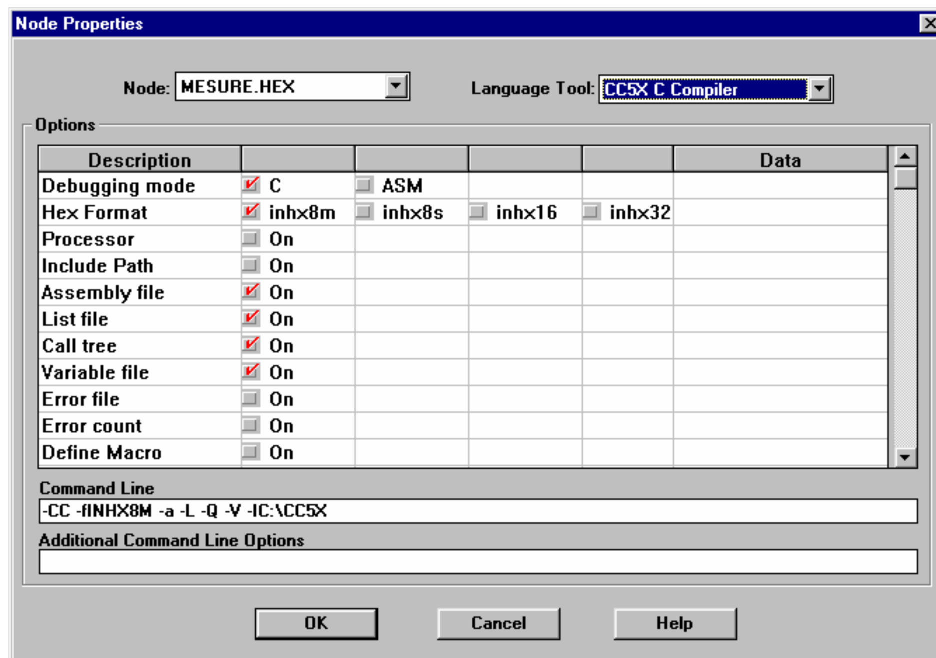
L'édition de liens n'est pas présentée dans ce document. Voir le manuel de l'utilisateur §6.8.

## 3.2) CRÉATION D'UN PROJET SANS ÉDITION DE LIENS

Pour créer un projet, il faut suivre les indications données § 6 du document « MPLAB 5.x de Microchip ».
Dans les propriétés du fichier exécutable, il faut choisir dans la rubrique « Language Tool » : « CC5X Compiler » pour indiquer qu'il n'y a pas d'édition de liens.

Il n'est pas possible d'éditer les propriétés du fichier source.

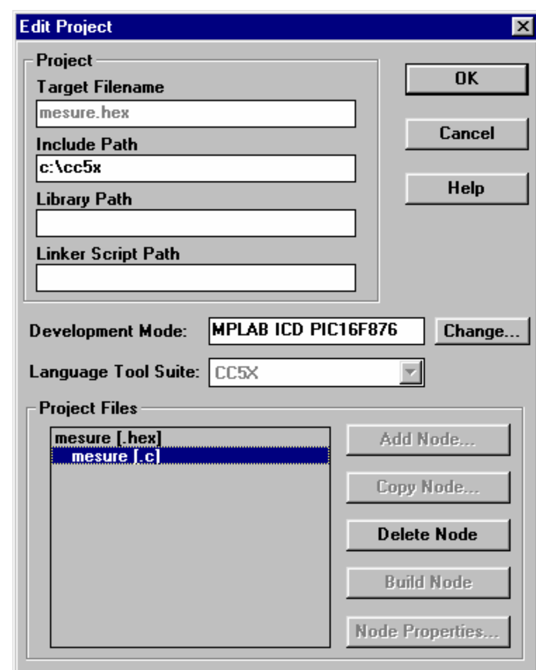
Les principales options sont fixées à partir de la boîte de dialogue ci-dessous.




**Assembly File** : fichier en langage d'assemblage uniquement sans les adresses et les codes des instructions. Extension asm
**List File** : Fichier en langage d'assemblage, avec les adresses et les codes des instructions. Extension lst.
**Call Tree** : Structure des appels de fonctions. Extension fcs.
**Variable File** : Liste des variables avec leurs adresses. Extension var.

Pour le détail de toutes les options, voir le manuel de l'utilisateur, pages 48 et suivantes.

La structure d'un projet sans édition de liens est toujours du type ci-contre, avec un seul fichier source.



# 4) EXTENSIONS AU C ANSI / PARTICULARITÉS DU COMPILATEUR

## 4.1) DIVERGENCE PAR RAPPORT AU C ANSI

Les fonctions ne peuvent être utilisées de façon récursive (limitation due aux faibles ressources du PIC, notamment une pile de quelques niveaux seulement).
Une fonction peut être appelée par le programme principal ou par le gestionnaire d'interruption.

Les variables globales initialisées ne sont pas supportées.

Le codage des certaines données n'est pas conforme au C ANSI (voir ci-dessous)

## 4.2) DONNÉES

Seuls les types char (char, signed char, etc.) sont codés selon le C ANSI.

Les types **int** et **long** sont codés comme suit (non conformes au C ANSI)

| Type | codage |
|---|---|
| int | 8 bits |
| long | 16 bits |

Pour les entiers, en plus des types de données du C ANSI, le compilateur dispose des types mentionnés ci-dessous :

| Type | codage | valeurs |
|---|---|---|
| **bit** | 1 bit | 0 ou 1 |
| **int8, s8** | 8 bits | -128 à 127 |
| **int16, s16** | 16 bits | -32768 à 32767 |
| **int24, s24** | 24 bits | -8388608 à 8388607 |
| **int32, s32** | 32 bits | -2147483648 à 2147483647 |
| **uns8, u8** | 8 bits | 0 à 255 |
| **uns16, u16** | 16 bits | 0 à 65535 |
| **uns24, u24** | 24 bits | 0 à 16777215 |
| **uns32, u32** | 32 bits | 0 à 4294967295 |

Pour les données à virgule flottante de types float ou double, le compilateur dispose des types mentionnés ci-dessous :

| Type | codage | valeurs extrêmes / Résolution |
|---|---|---|
| float16 | 16 bits | *voir manuel page 15* |
| float, float24 | 24 bits | *voir manuel page 15* |
| double, float32 | 32 bits | *voir manuel page 15* |

*Il est possible de convertir les float24 et 32 au format IEEE754. Voir manuel de l'utilisateur page 15.*

Le compilateur dispose aussi des types décimaux à virgule fixe. *Voir manuel page 16*. L'utilisation de ce type de données est rare.

## DONNÉE DE TYPE BIT

*Ce type de donnée permet d'économiser de la RAM, ce qui est très important avec les PIC d'entrée de gamme.*

Le type bit peut être utilisé comme une variable simple, comme paramètre d'une fonction, comme valeur retournée par une fonction.
Il ne semble pas qu'il soit possible de définir un pointeur sur type bit (de toute façon c'est sans intérêt).

| Déclaration | Utilisation |
|---|---|
| Comme n'importe quelle donnée. Ex :<br>bit Fonctionnement ; /*Fonctionnement ne peut valoir que 2 valeurs 0 ou 1*/ | Comme n'importe quelle donnée. Ex :<br>if (Fonctionnement == NORMAL)<br>{...} /* NORMAL vaut 1 par exemple */ |

Pour la déclaration d'un bit placé dans un registre avec adresse, voir ci-dessous.

### Utilisation avec MPLAB

Lors du déboguage, il est impossible de visualiser une donnée de type bit avec MPLAB.

## ACCÈS À UN BIT D'UNE DONNÉE

Il est possible d'accéder individuellement à chacun des bits d'une donnée, en mentionnant le numéro du bit précédé de . (point) après le nom de la variable.

Exemples : PORTA.0 = 1 ;
if (PORTB.4 == 1) ...

## CODAGE DES DONNÉES SUR PLUSIEURS OCTETS

L'octet de poids faible est rangé à l'adresse basse. C'est la convention petit boutiste (little endian).

## BASES POUR LES ENTIERS

En plus des bases du C ANSI, le compilateur CC5X accepte la base 2.
Notation : 0bnombre ou 0Bnombre. Ex : 0b01110011

Il est possible de séparer les bits par des points pour plus de visibilité.
Exemple : 0b1101.1001, 0b1.101.11.00

## CHAMPS DE BITS

Indiquer l'ordre de rangement des bits

## 4.3) DÉCLARATION D'UNE DONNÉE PLACÉE EN REGISTRE AVEC ADRESSE

L'utilisateur a peu à manipuler de telles déclarations. Toutes les déclarations pour les registres SFR sont dans un fichier en-tête spécifique à chaque PIC fourni.
Avec MPLAB, il n'y a aucune inclusion à placer dans le fichier source. L'inclusion d'une directive #include <nom_pic.h> est automatique lors de la compilation.

### DÉCLARATION POUR UNE DONNÉE D'UN OCTET

**syntaxe** : unsigned char <NOM_REGISTRE> @ <adresse_registre> ;

**exemple** : unsigned char TMR0 @ 0x01 ;

***remarques*** :
*1_ Il est inutile de spécifier la banque*
*2_ Il est aussi possible de rajouter #pragma devant la déclaration. Ceci n'a aucun intérêt, mais c'est ce qui est fait dans les fichiers en-tête fournis avec le compilateur.*

### DÉCLARATION POUR UNE DONNÉE DE TYPE BIT

**syntaxe** : bit <NOM_BIT> @ <NOM_REGISTRE>.<N°_BIT>
<NOM_REGISTRE> peut être remplacé par une adresse.

bit RC7 @ PORTC.7 /* PORTC doit avoir été déclaré */

***remarque*** *: Il est aussi possible de rajouter #pragma devant la déclaration. Ceci n'a aucun intérêt, mais c'est ce qui est fait dans les fichiers en-tête fournis avec le compilateur.*

## 4.4) PLACEMENT DES VARIABLES EN RAM

Le placement des variables dans une des banques en RAM est laissé au soin de l'utilisateur. Le choix de la banque peut s'effectuer avec une directive de compilation **#pragma rambank** placée avant la définition des variables ou avec le qualificateur **bank**x (x vaut de 0 à 1 ou à 3 selon le PIC).

### UTILISATION DE LA DIRECTIVE #PRAGMA RAMBANK

Les variables définies avant toute directive sont placées en banque 0.

| | |
|---|---|
| #pragma rambank 1<br>char a, b, c | a, b et c sont placées en banque 1 |
| #pragma rambank 0 | les variables qui suivent sont placées en banque 0 |

### QUALIFICATEUR « BANK »

Les données définies sans qualificateur bankx sont placées dans la banque 0.

Ex : bank1 char Tab[60];

## 4.5) POINTEURS

Le compilateur permet d'utiliser 2 tailles pour les pointeurs, 1 octet ou 2 octets, pour accéder à tout ou partie de la ROM et de la RAM.

Le choix de la taille des pointeurs peut être laissé à l'initiative du compilateur pour les pointeurs qui ne sont pas inclus dans des tableaux ou des structures.

L'utilisateur peut spécifier la taille d'un pointeur lors de sa définition à l'aide du qualificateur **size1** ou **size2**. Ces qualificateurs sont à utiliser essentiellement dans les tableaux de pointeurs ou avec des structures dont les champs sont des pointeurs.

Les différents types de pointeurs sont donnés dans le tableau suivant :

| Taille pointeur | mémoire accessible | zone accessible |
|---|---|---|
| 8 bits | RAM | 256 octets |
| 16 bits | RAM | différents segments de RAM jusqu'à 256 octets chacun |
| 8 bits | ROM | 256 octets |
| 16 bits | ROM | tout l'espace mémoire |
| 8 bits | RAM et ROM | 128 octets en RAM et 128 octets en ROM. le bit 7 est utilisé pour détecter un accès en RAM ou en ROM |
| 16 bits | RAM ou ROM | tout l'espace mémoire. le bit 15 est utilisé pour indiquer le type de mémoire |

Exemples de définition :
bank1 size2 char* PtVar1 ; /* définition d'un pointeur de 2 octets sur des entiers 8 bits. Le pointeur est placé en banque 1. Il permet d'accéder à tout l'espace RAM et ROM */
char* PtVar2 ; /* définition d'un pointeur sans précision de taille. Le compilateur fixera la taille en fonction des accès mémoire réalisés avec ce pointeur */

Le choix pour la taille d'un pointeur s'effectue selon les priorités suivantes :
1_ qualificateur
2_ choix automatique (pointeurs simples)
3_ choix d'après le modèle par défaut fixé par les options de la ligne de commande

*Pour plus de détail, voir la manuel de l'utilisateur pages 23 et suivantes.*

## 4.6) INTERRUPTION EN C

Le nom de la fonction doit être précédée de **interrupt**. L'en-tête de la fonction ne doit pas mentionner de valeur de retour et ne doit pas avoir de paramètre.

En tête : **interrupt TraitementInter(void)**

Le PIC ne sauvegarde automatiquement dans la pile que l'adresse de retour.
Le compilateur ne sauvegarde pas automatiquement les différents registres modifiés lors du traitement de l'interruption. C'est à l'utilisateur de le faire à l'aide de macros fournies dans le fichier int16cxx.h.

Avec un seul fichier source en C, sans utilisation de l'édition de liens, il faut fixer l'adresse de début du programme de traitement d'interruption à l'aide de la directive #pragma origin 4

exemple de programme

| | |
|---|---|
| #include <int16cxx.h> | nécessaire pour les macros utilisées pour la sauvegarde et la restitution des registres |
| #pragma origin 4 | les programmes d'interruption de tous les PIC commencent à l'adresse 4 |
| interrupt TraitIner (void) { | le nom importe peu |
| int_save_registers | macro (non suivie d'un ;) définie dans int16cxx.h |
| ... | |
| int_restore_registers | macro (non suivie d'un ;) définie dans int16cxx.h |
| } | |

## 4.7) INSERTION DE LIGNES EN LANGAGE D'ASSEMBLAGE DANS UN PROGRAMME SOURCE EN C

Des lignes en langage d'assemblage peuvent être insérées dans un programme source en C de 2 manières :

- entre les directives #asm et #endasm
- avec des fonctions « in line »

Les directives #asm et #endasm permettent d'intégrer plusieurs instructions.
**Attention** : les instructions en langage d'assemblage entrées avec les directives précédentes ne font pas partie de manière syntaxique du C et elles ne se conforment pas aux règles de contrôle de flux. Par exemple, on ne peut utiliser un bloc #asm dans une structure if.

Les fonctions in line les plus utilisées sont :

- **nop()**, **nop2()** (2 cycles machines d'attente, mais un seul mot en mémoire),
- **clrwdt()** (pour RàZ timer chien de garde).

*Pour plus de détail, voir le manuel de l'utilisateur page 63 et suivantes.*

**L'insertion de lignes en langage d'assemblage dans un programme source en C doit rester rare et utilisée avec beaucoup de précautions (pas d'interférence avec les registres utilisés par le compilateur, etc.)**

## 4.8) COMMENTAIRES

Le compilateur accepte les commentaires du style C++ ainsi que les commentaires imbriqués. Les commentaires imbriqués sont pratiques pour ne pas compiler une partie du programme qui contient des commentaires.

Exemples :

```
// Ceci est un commentaire sur une ligne

/* passage supprimé pour gagner quelques octets
if (BP2 == 0){
        AffichageMsgAccueil(); /* avec animation */
        }
*/
```

Comme les commentaires imbriqués sont acceptés, les commentaires de titres de la forme suivante ne sont pas acceptés :

```
/****************************************************************************/
/* Nom fonction : ReinitDureePhaseInst
/* Appelée par: ReglageDateHeure()
/* Appelle : -
/* Niveau de pile: +1
/****************************************************************************/
```

Il faut utiliser le style suivant :

```
/****************************************************************************/
// Nom fonction : ReinitDureePhaseInst
// Appelée par: ReglageDateHeure()
// Appelle : -
// Niveau de pile: +1
/****************************************************************************/
```

## 5) DÉFINITION DES REGISTRES

Certains des registres sont communs à tous les PIC et sont connus du compilateur. Il s'agit des registres INDF, TMR0, PCL, STATUS, FSR, PORTA, PORTB, TRISA, TRISB, OPTION (+ quelques autres) *Voir détail page 98 du manuel de l'utilisateur.*

Pour les autres registres, il faut utiliser les fichiers en-tête fournis

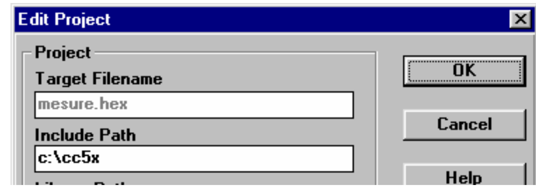
## 5.1) FICHIERS « EN-TÊTE » FOURNIS

De nombreux fichiers « en-tête » contenant les définition de tous les registres et de beaucoup de bits de contrôle / état sont fournis avec le compilateur.

Avec MPLAB, il est inutile d'inclure un fichier en-tête pour le PIC utilisé. La directive d'inclusion est effectuée automatiquement d'après la configuration du projet car le PIC cible est défini dans le projet.
Ex de ligne générée automatiquement : #include <16F877.h> pour un PIC16F877.

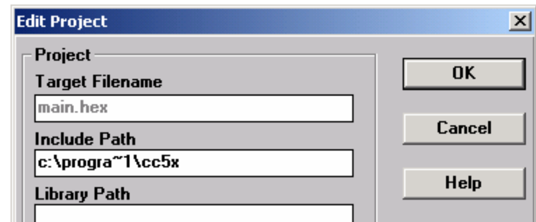
L'emplacement du fichier en-tête n'est pas à définir si on utilise les fichiers fournis. La ligne de commande automatiquement générée lors de la compilation donne l'emplacement.

Si on veut utiliser un autre dossier, on peut spécifier le chemin pour les fichiers « include » dans la boîte de paramétrage du projet.




Il est aussi possible de spécifier le chemin d'accès pour chaque nœud du projet.

Attention, si le dossier est dans Program Files, il faut utiliser le nom court de ce dossier progra~1.




## 5.2) NOM DES REGISTRES ET DES BITS

Le compilateur reprend les noms des registres définis dans la documentation Microchip, sauf pour quelques registres :

| Appellation Microchip | Appellation CC5X |
|---|---|
| OPTION_REG | OPTION A vérifier |
| *AD/$\overline{GONE}$* | GO |

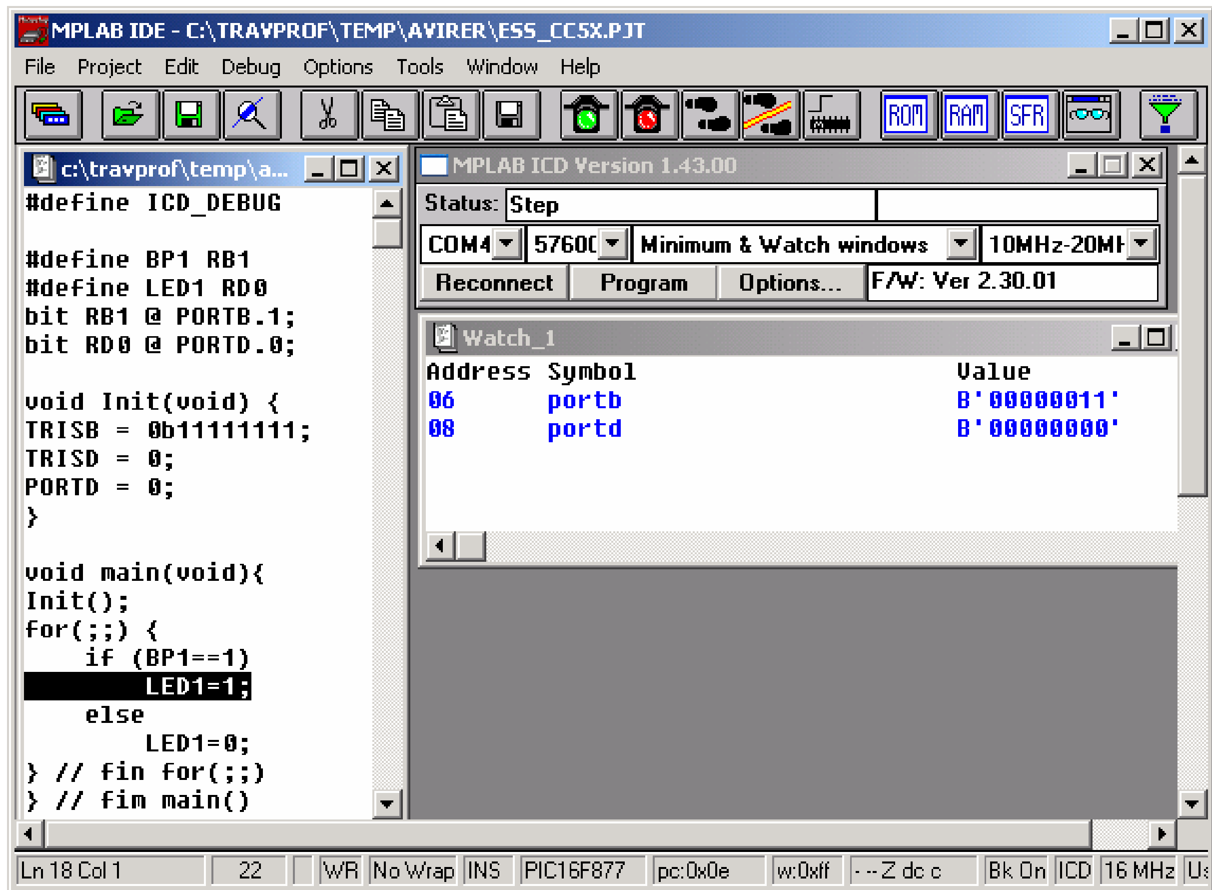
De nombreux bits de contrôle / état sont définis (avec le type bit –voir ci-dessus-).
Pour voir le détail des bits définis, éditer le fichier en-tête spécifique au PIC utilisé.

## 6) PROGRAMME DE DÉMARRAGE

**Avec CC5X, il n'y a pas de programme de démarrage** à proprement parler. Un programme de démarrage sert à mettre les variables globales non initialisées à 0 et à placer les valeurs initiales pour les variables globales initialisées.

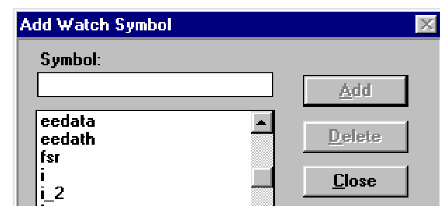
CC5X se contente de placer un « goto main » en début de programme.
**Les variables globales initialisées ne sont pas acceptées** → le programmeur doit leur affecter une valeur avant leur utilisation.
Les variables globales non initialisées sont dans une valeur inconnue en début de programme.

## 7) DÉBOGUAGE AVEC MPLAB




## VARIABLES LOCALES

**Les variables locales de même nom dans des blocs différents sont renommées**. Par exemple, avec 2 variables i, CC5X conserve ce nom pour la 1ère apparaissant dans le fichier source et renomme la 2ème i_2. Ce sont ces noms qui apparaissent dans la boîte de dialogue Edit Watch.




La difficulté est de faire la relation entre le nom apparaissant dans la boîte de dialogue et la bonne variable locale du programme.

**Le plus simple est d'utiliser des noms différents pour toutes les variables locales**. Pour une application pédagogique simple avec peu de variables cela est très facile à mettre en œuvre.

Si on conserve de noms identiques, il est possible d'éditer le fichier de listage pour voir comment ont été renommées les variables locales (en utilisant l'outil de recherche avec l'identificateur d'origine).

Ci-contre, on peut voir un extrait du fichier de listage.

```
               0070           ;void Tempo2(void)
               0071           ;{
               0072 Tempo2
               0073           ;char i;
               0074           ;for(i=0x80;--i;)
0019 3080      0075    MOVLW .128
001A 1283      0076    BCF   0x03,RP0
001B 1303      0077    BCF   0x03,RP1
```

| 001C 00A0 0078 MOVWF **i_2**

## 8) BIBLIOTHÈQUES

CC5X ne permet pas la création et l'utilisation de bibliothèques au sens habituel du terme, c'est-à-dire des fichiers qui contiennent des fonctions déjà compilées.

Cependant on peut utiliser un fichier .c comme une bibliothèque avec l'emploi de 2 directives particulières qui permettent de ne pas compiler les fonctions non utilisées.
Il s'agit de #pragma library1 et #pragma library 0. Les fonctions doivent être placées entre ces 2 directives.
Le fichier bibliothèque .c doit être inclus dans le fichier principal avec une directive #include.