

Utilisation de Visual Studio pour un projet qui n'est pas fait à la base pour Visual Studio

But

Le but de ce document est de décrire comment on peut utiliser Visual Studio pour travailler sur du code qui n'est à la base pas compilable sous Visual Studio. L'exemple qui va être traité est celui du programme embarqué dans la carte Navi-Ctrl du quadrirotor de chez Mikrokopter, qui est basée sur un processeur de type ARM. Attention, voici les prérequis principaux de la méthode présentée :

- _ On doit disposer d'un fichier Makefile ou d'un script similaire qui lance les commandes/programmes nécessaires à la compilation du programme embarqué.
- _ Il faut que les commandes/programmes utilisés soient disponibles sous Windows (ou puissent être installés).

A la fin de ce tutoriel, le projet créé devrait permettre de :

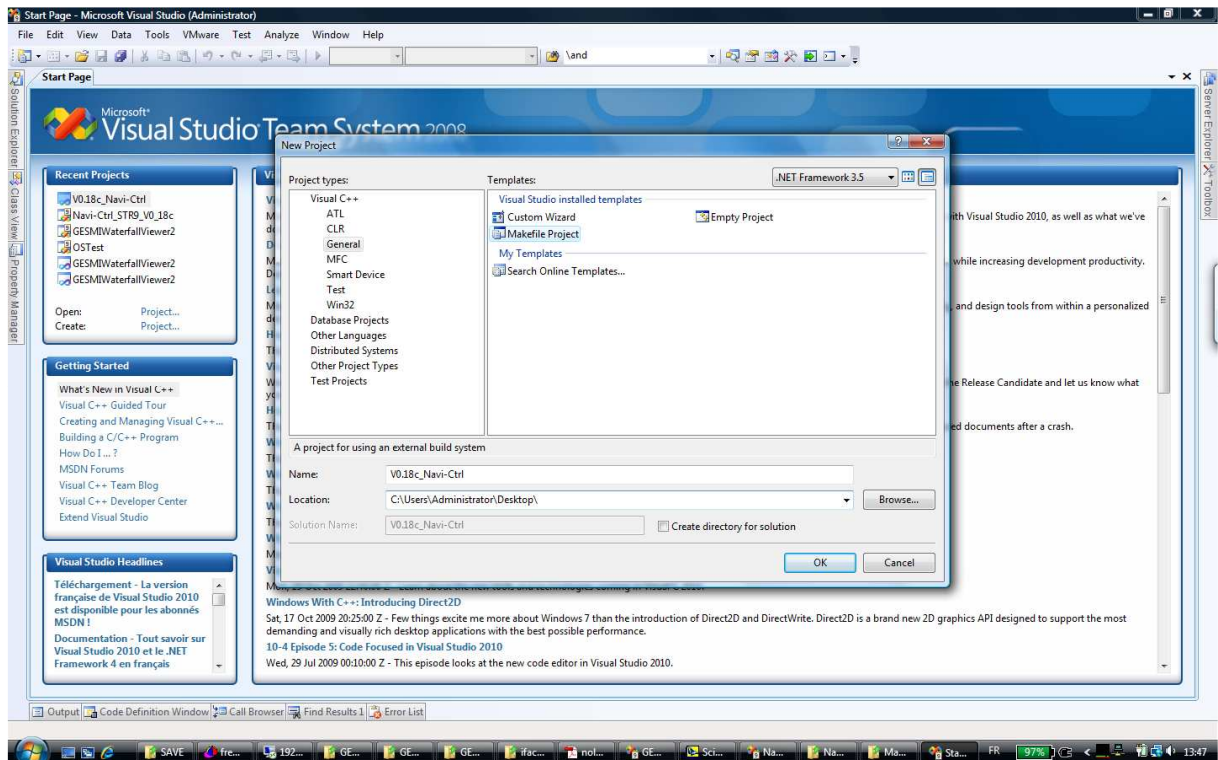
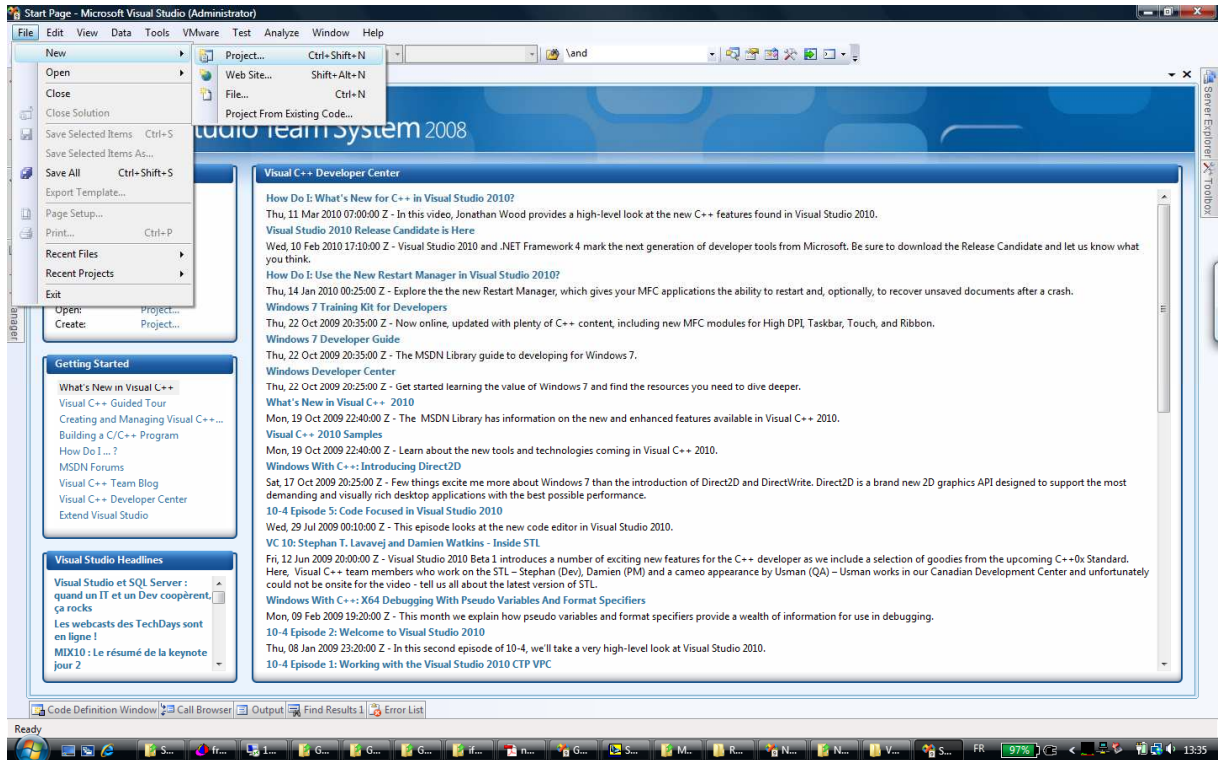
- _ Editer le code C embarqué dans Visual Studio comme on le ferait pour un programme C/C++ pour PC habituel. On bénéficiera donc de la coloration syntaxique, l'auto complétion, les info-bulles donnant des informations sur le code sous la souris, la possibilité d'atteindre l'endroit dans le code où est définie une variable ou une fonction (clic droit puis Go to définition)... L'ensemble de ces fonctionnalités de Visual Studio sont souvent regroupée sous le nom Intellisense.
- _ Compiler le code en utilisant les Build\Rebuild Solution et les raccourcis associés.

En revanche, voici ce que le projet ne supportera pas (a priori) :

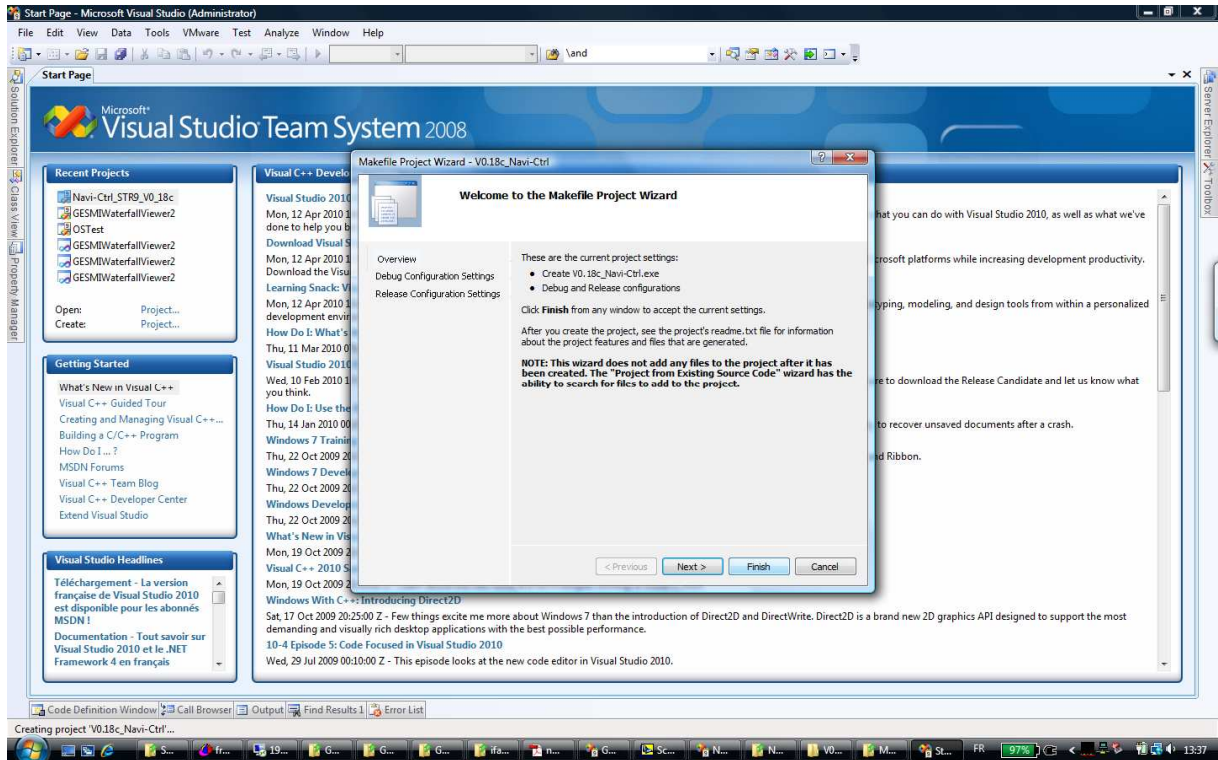
- _ L'exécution et débogage de l'application dans Visual Studio. En effet, dans notre exemple, le programme compilé est fait pour une carte embarquée avec un processeur ARM et ne fonctionnera donc pas sur un PC sous Windows.
- _ L'ajout/suppression/renommage de fichiers au projet Visual Studio comme on le fait pour un projet normal n'a pas d'influence sur le programme qu'on cherche à compiler. Il faut a priori modifier le Makefile.

Création du projet

Le type de projet à créer est Makefile project.

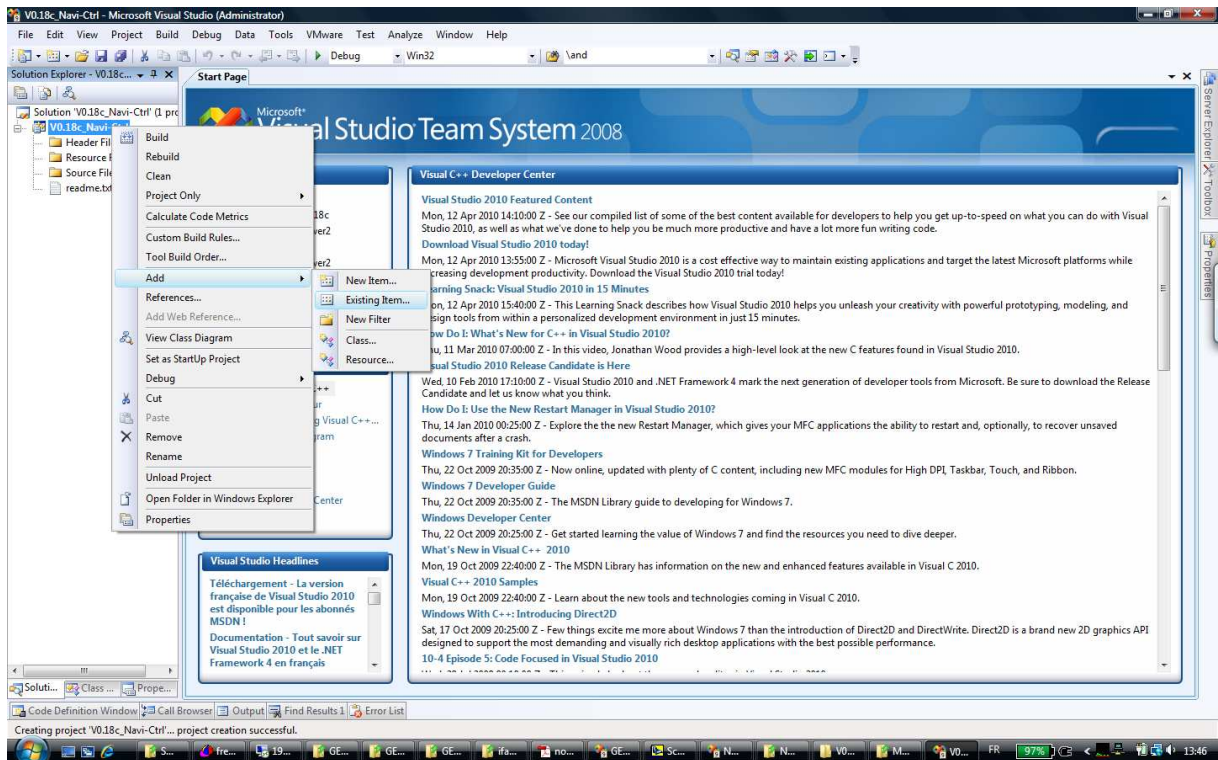


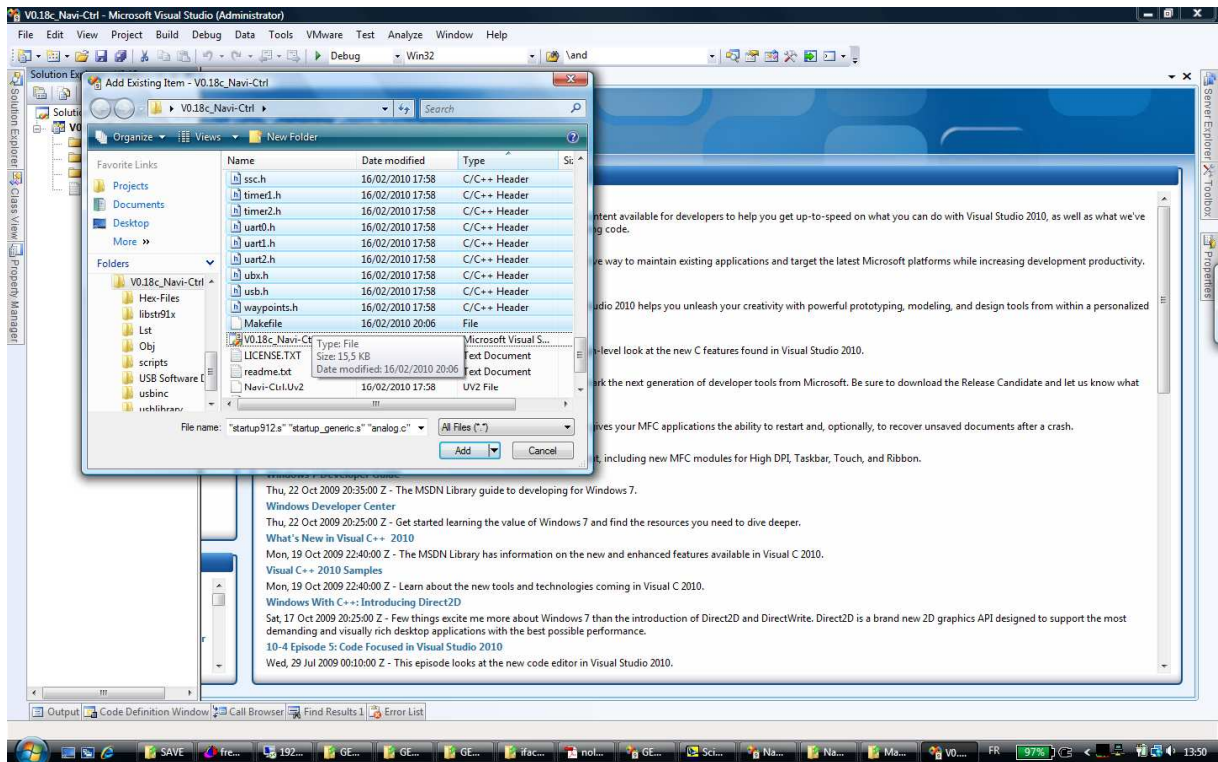
Les options proposées dans cette page seront configurées plus tard.



Ajout des fichiers au projet

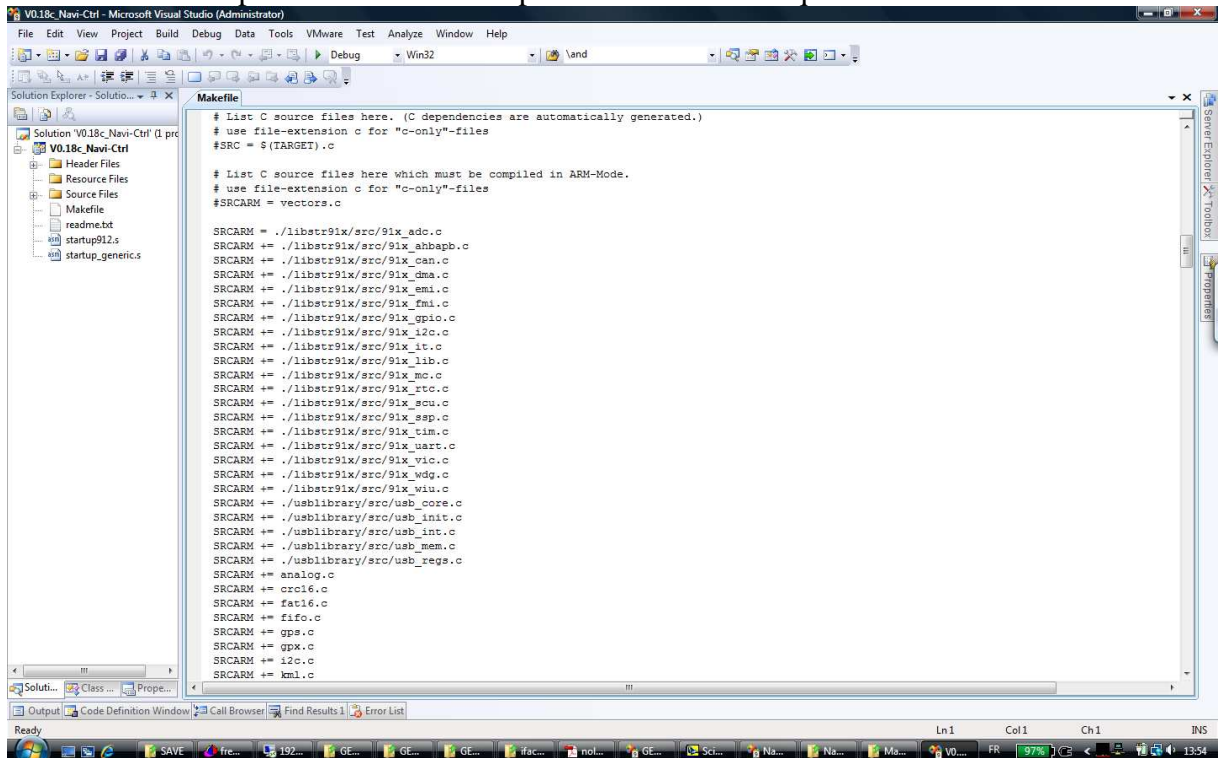
Nous devons ajouter les fichiers qui nous intéressent au projet Visual Studio. Cependant, il faut garder à l'esprit que cette étape n'a aucune influence sur les fichiers qui seront compilés, seul ce qui est dans le Makefile compte.





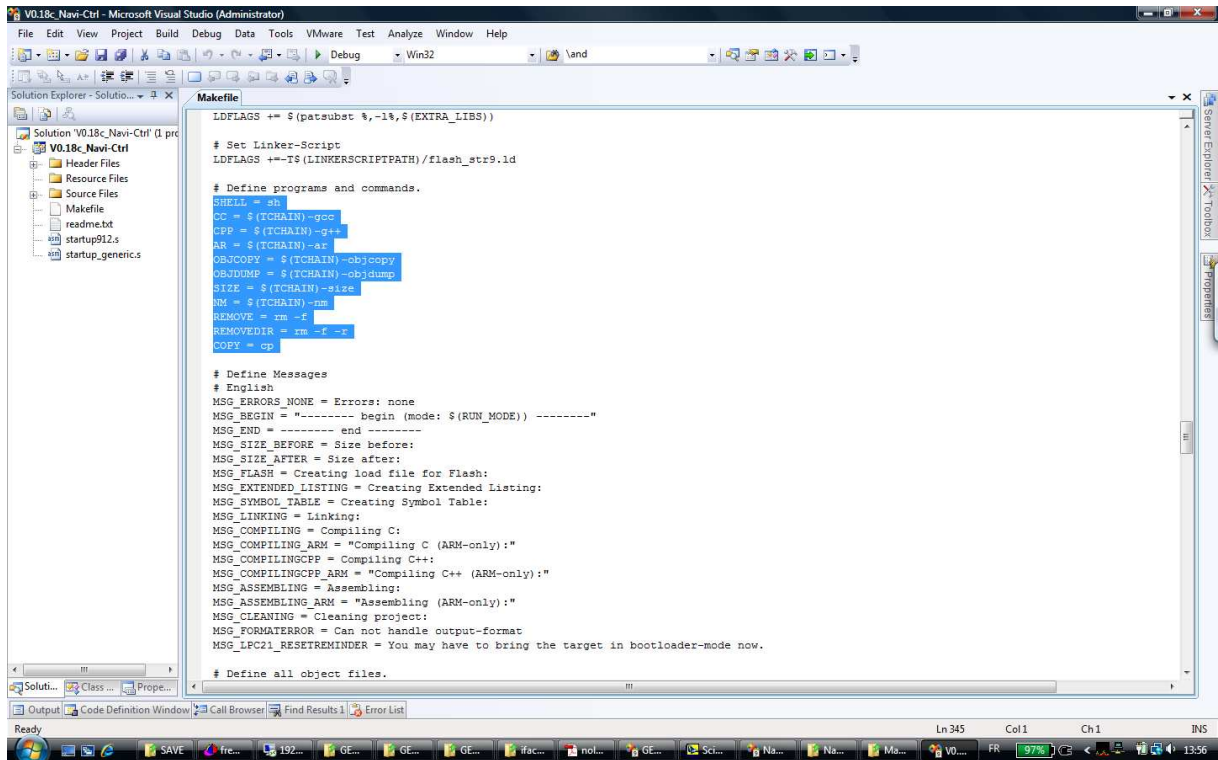
Analyse du Makefile

Il est conseillé de parcourir le Makefile pour voir les fichiers qui sont utilisés.



Il faut aussi regarder dans le Makefile les commandes/programmes qui sont utilisés. On voit notamment que des commandes telles que rm et cp sont utilisées. Ces commandes n'étant pas disponibles sous Windows par défaut, il faut installer MinGW et MSYS pour les avoir.

MinGW est aussi nécessaire pour avoir la commande make, qui lancera la compilation en lisant le Makefile.



```
LDLFLAGS += $(patsubst %, -l%, $(EXTRA_LIBS))

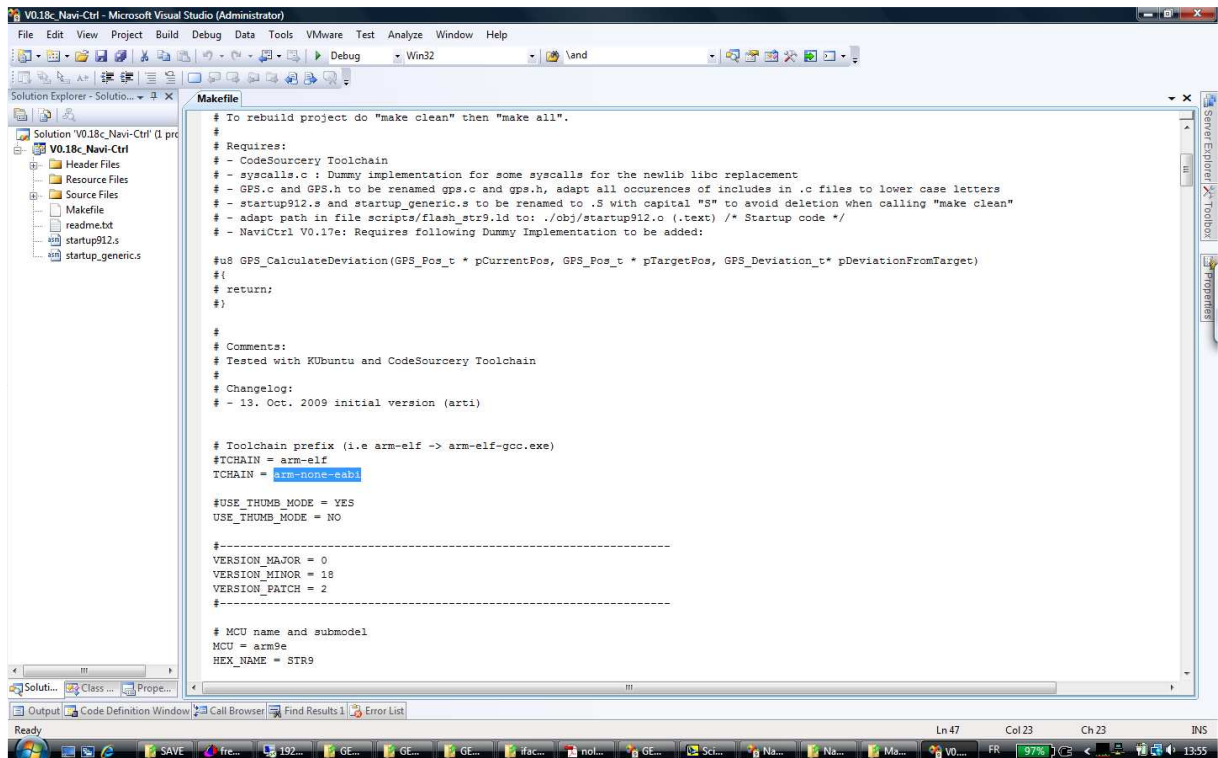
# Set Linker-Script
LDLFLAGS +=-T$(LINKERSRIPTPATH)/flash_str9.ld

# Define programs and commands.
SHELL = sh
CC = $(TCHAIN)-gcc
CXX = $(TCHAIN)-g++
AR = $(TCHAIN)-ar
OBJCOPY = $(TCHAIN)-objcopy
OBJDUMP = $(TCHAIN)-objdump
SIZE = $(TCHAIN)-size
RM = $(TCHAIN)-rm
REMOVE = rm -f
REMOVEDIR = rm -rf
COPY = cp

# Define Messages
# English
MSG_ERRORS_NONE = Errors: none
MSG_BEGIN = "----- begin (mode: $(RUN_MODE)) -----"
MSG_END = "----- end -----"
MSG_SIZE_BEFORE = Size before:
MSG_SIZE_AFTER = Size after:
MSG_FLASH = Creating load file for Flash:
MSG_EXTENDED_LISTING = Creating Extended Listing:
MSG_SYMBOL_TABLE = Creating Symbol Table:
MSG_LINKING = Linking:
MSG_COMPILING = Compiling C:
MSG_COMPILING_ARM = "Compiling C (ARM-only):"
MSG_COMPILINGCXX = Compiling C++:
MSG_COMPILINGCXX_ARM = "Compiling C++ (ARM-only):"
MSG_ASSEMBLING = Assembling:
MSG_ASSEMBLING_ARM = "Assembling (ARM-only):"
MSG_CLEANING = Cleaning project:
MSG_FORMATERROR = Can not handle output-format
MSG_IPC21_RESETRINDER = You may have to bring the target in bootloader-mode now.

# Define all object files.
```

De plus, on voit que le compilateur utilisé est celui de CodeSourcery et que la commande pour appeler le compilateur est arm-none-eabi-gcc. Il nous faudra donc installer ce compilateur aussi.



```
# To rebuild project do "make clean" then "make all".

# Requires:
# - CodeSourcery Toolchain
# - syscalls.c : Dummy implementation for some syscalls for the newlib libc replacement
# - GPS.c and GPS.h to be renamed gps.c and gps.h, adapt all occurrences of includes in .c files to lower case letters
# - startup912.s and startup_generic.s to be renamed to .S with capital "S" to avoid deletion when calling "make clean"
# - adapt path in file scripts/flash_str9.ld to ./obj/startup912.o (.text) /* Startup code */
# - NavCtrl1 V0.17e: Requires following Dummy Implementation to be added:

# GPS_CalculateDeviation(GPS_Pos_t * pCurrentPos, GPS_Pos_t * pTargetPos, GPS_Deviation_t* pDeviationFromTarget)
#{
# return:
#}

# Comments:
# Tested with KUbuntu and CodeSourcery Toolchain
#
# ChangeLog:
# - 13. Oct. 2009 initial version (arti)

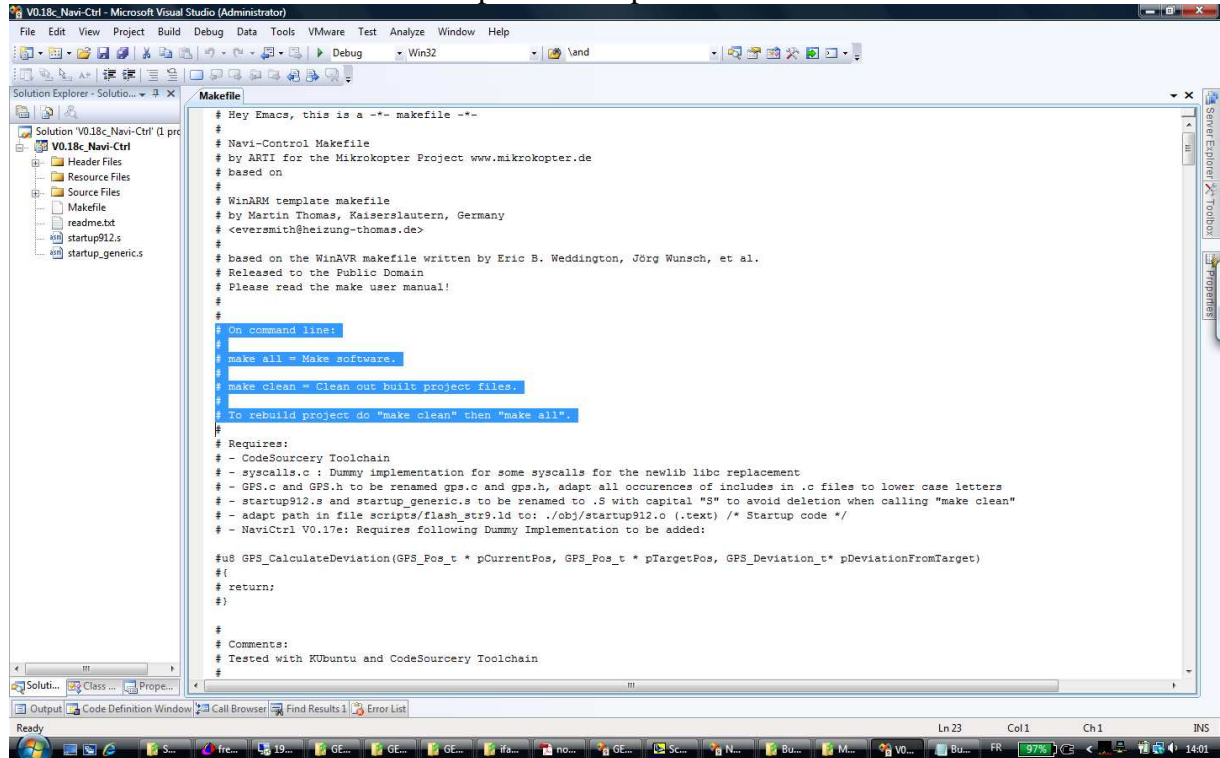
# Toolchain prefix (i.e arm-elf -> arm-elf-gcc.exe)
#TCHAIN = arm-elf
TCHAIN = arm-none-eabi

#USE_THUMB_MODE = YES
USE_THUMB_MODE = NO

#-----
VERSION_MAJOR = 0
VERSION_MINOR = 18
VERSION_PATCH = 2
#-----

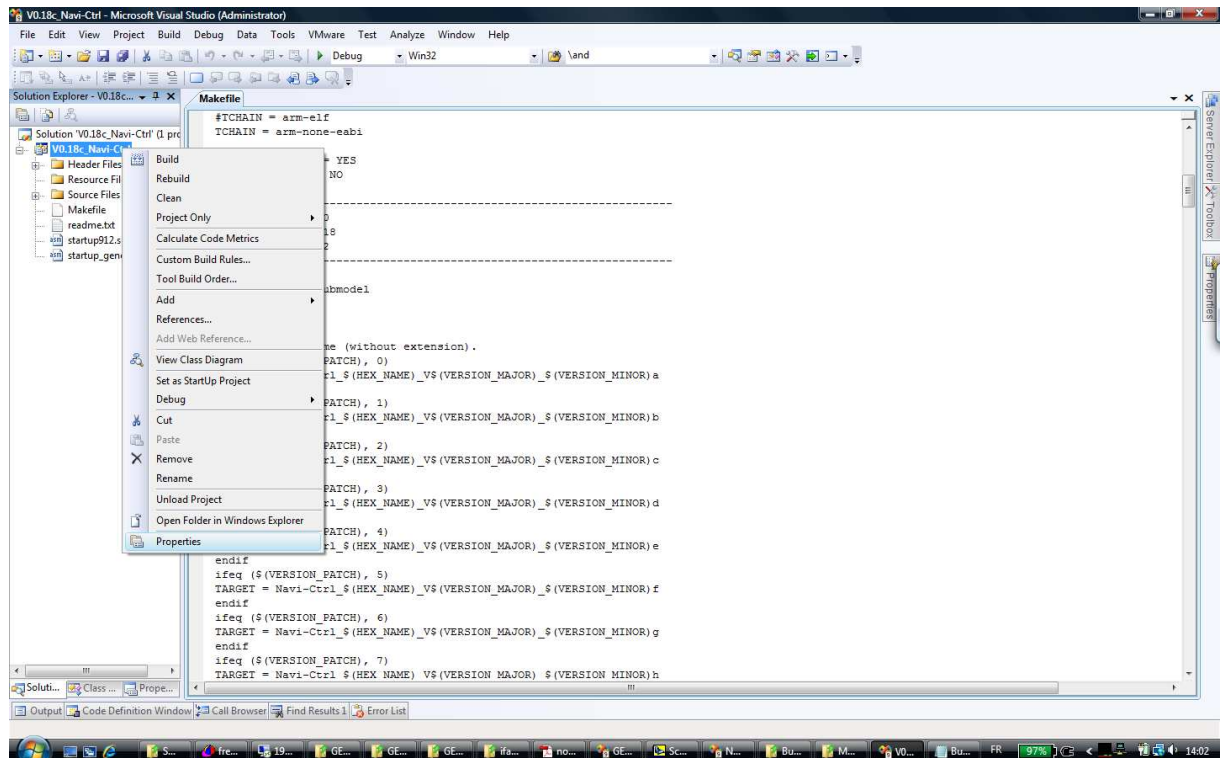
# MCU name and submodel
MCU = stm9e
HEX_NAME = STR9
```

On voit ici les commandes à utiliser pour la compilation.



Réglage des propriétés du projet

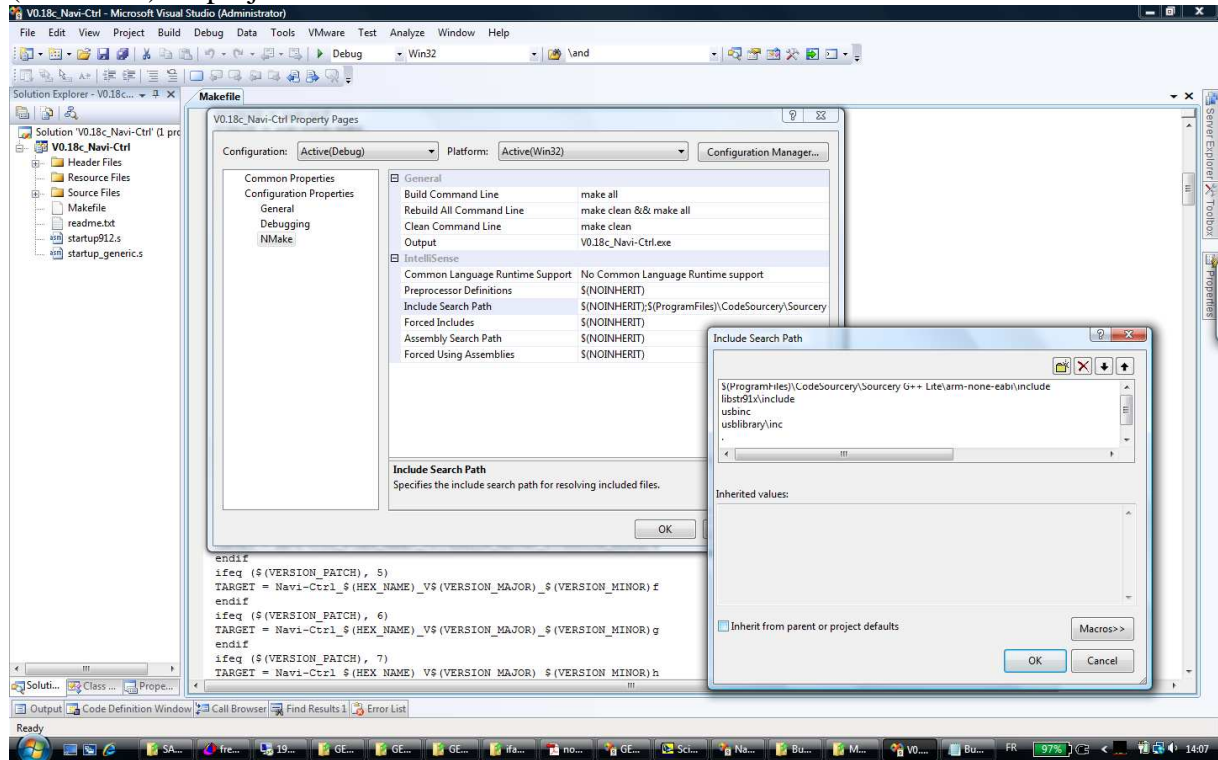
En fonction de ce qu'on a vu dans le Makefile, il va nous falloir régler les propriétés de notre projet Visual Studio.



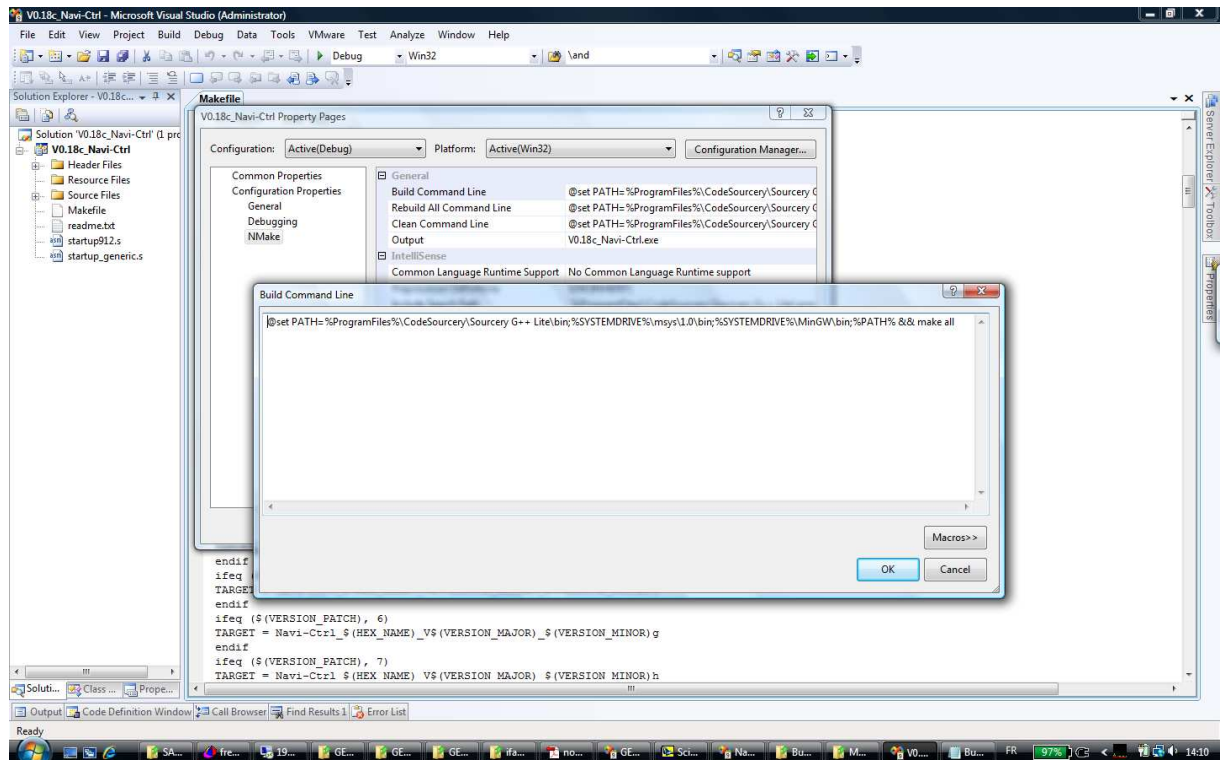
Dans un 1^{er} temps, il faut régler les commandes qui seront appelées lorsque l'on cliquera sur les menus Build\Build Solution, Build\Rebuild Solution et Build\Clean comme on l'a vu dans le Makefile. Cependant, nous allons compliquer ces valeurs dans la suite.

Le paramètre Output n'est pas forcément indispensable.

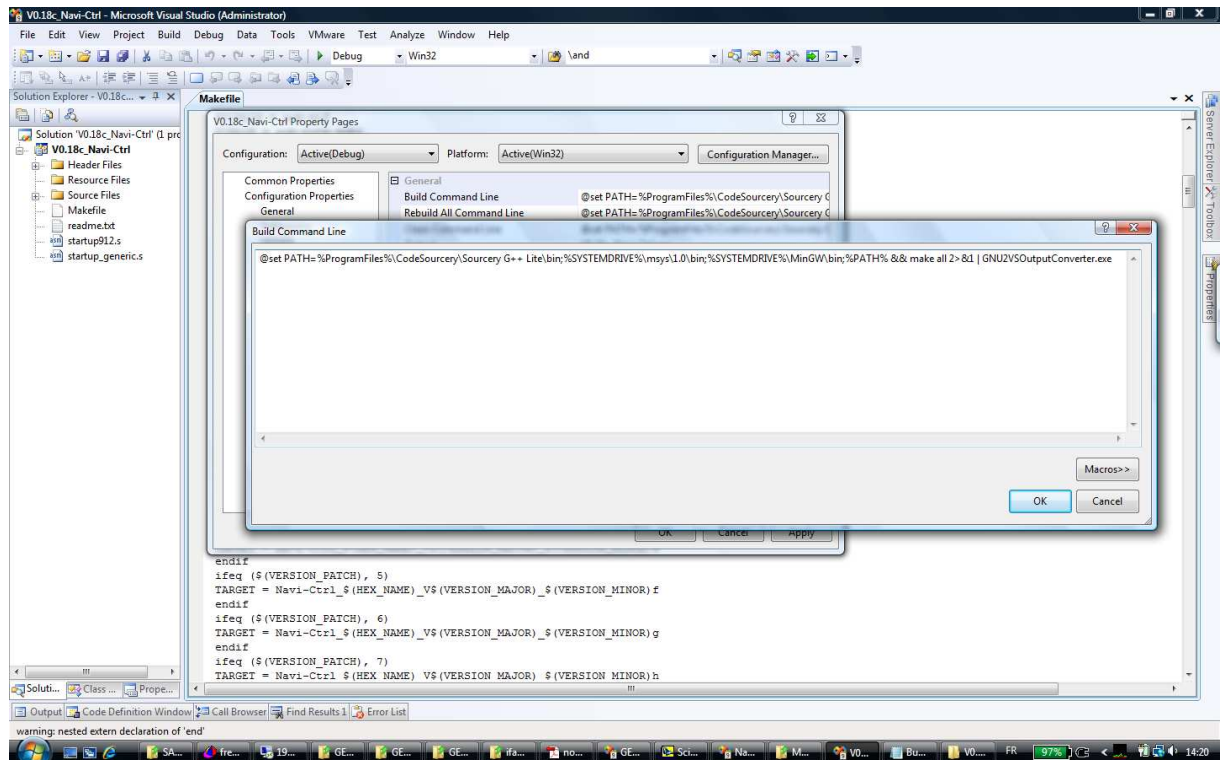
Le paramètre Include Search Path est important pour que toutes les fonctions d'Intellisense fonctionnent correctement. Il faut y indiquer les dossiers où se trouvent les fichiers headers (fichiers .h) du projet.



En fait, il peut y avoir un problème pour les paramètres Build, Rebuild et Clean lorsque toutes les commandes appelées par le Makefile ou la commande make ne sont pas dans un dossier listé dans la variable système PATH de Windows. Pour en être sûr, nous pouvons modifier les paramètres Build, Rebuild et Clean comme suit :

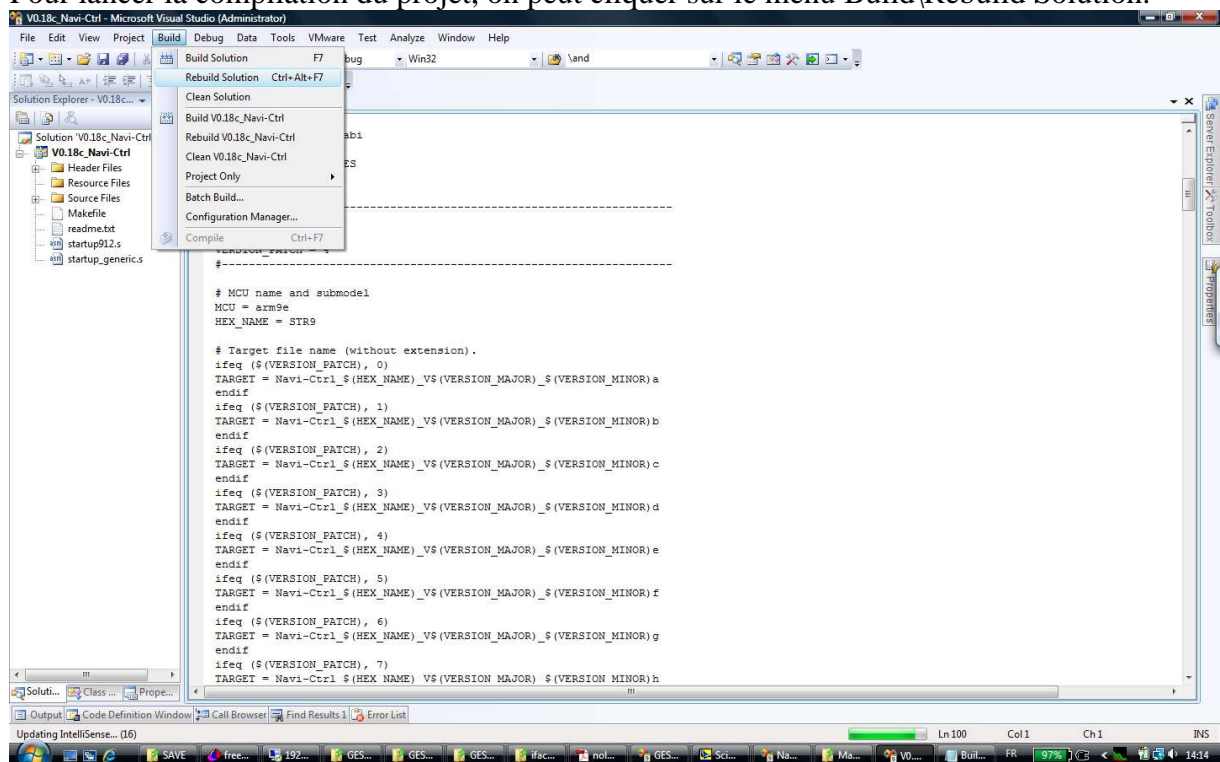


Une autre modification peut être faite en plus pour ces paramètres. Cette dernière modification requiert que le programme GNU2VSOuputConverter.exe soit placé dans le dossier du projet et que le compilateur (dans notre cas celui de CodeSourcery) se comporte comme GCC (compilateur habituel sous Linux). En effet, les messages d'erreurs et d'avertissement de ces compilateurs ne sont pas bien interprétés par Visual Studio. GNU2VSOuputConverter.exe sert à convertir les messages du compilateur en des messages compréhensibles par Visual Studio. Notez que cette étape n'est pas critique, elle permet juste de pouvoir double-cliquer sur le message d'erreur ou d'avertissement pour atteindre l'endroit dans le code correspondant.

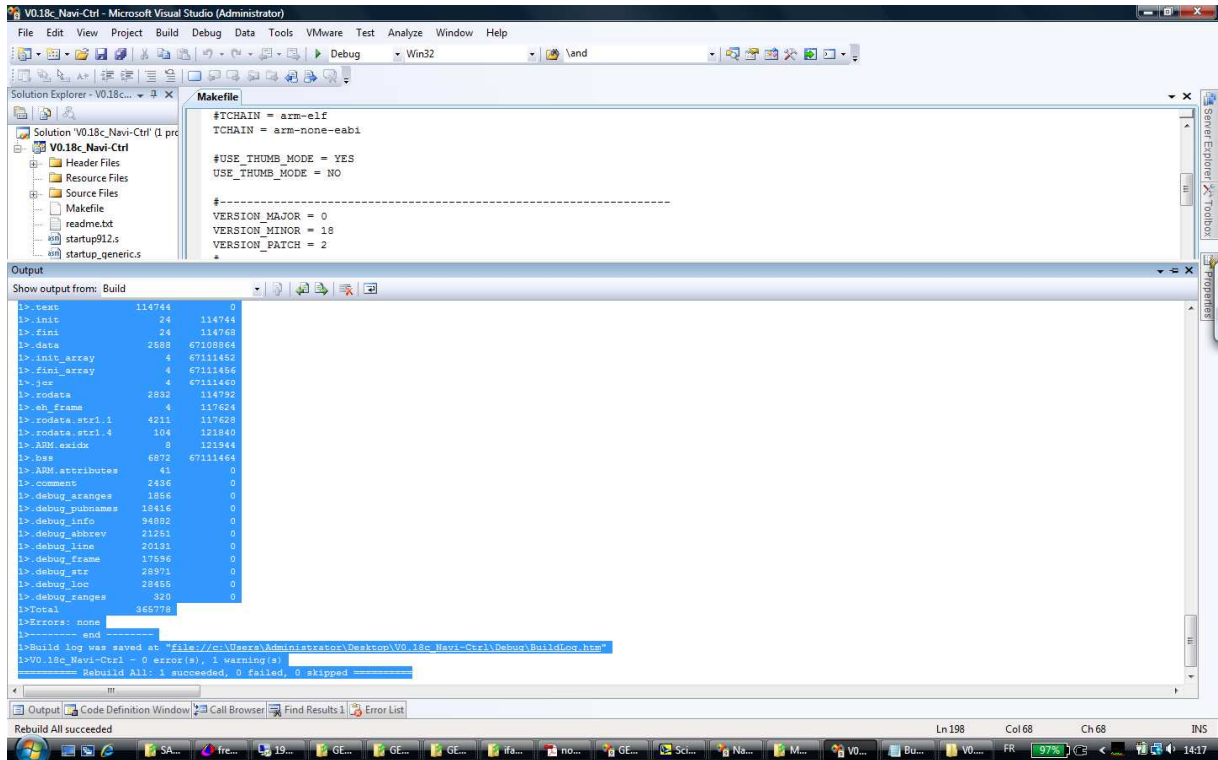


Compilation

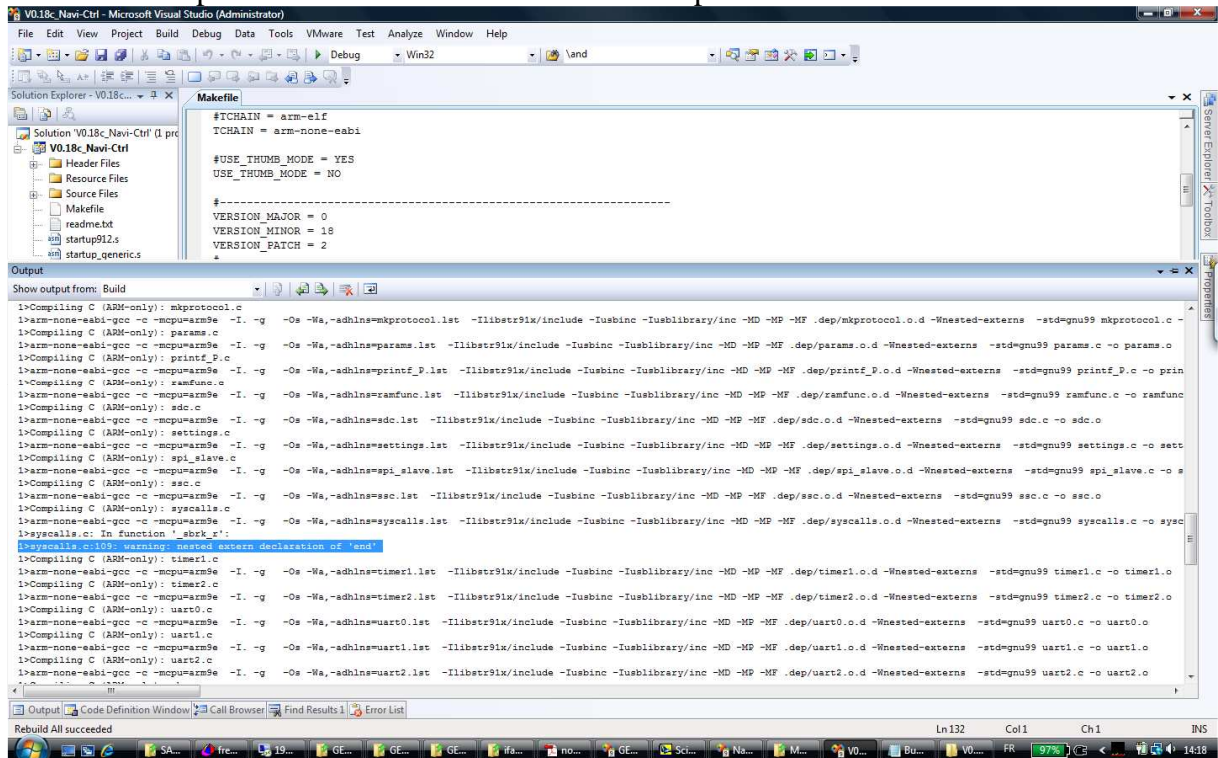
Pour lancer la compilation du projet, on peut cliquer sur le menu Build\Rebuild Solution.



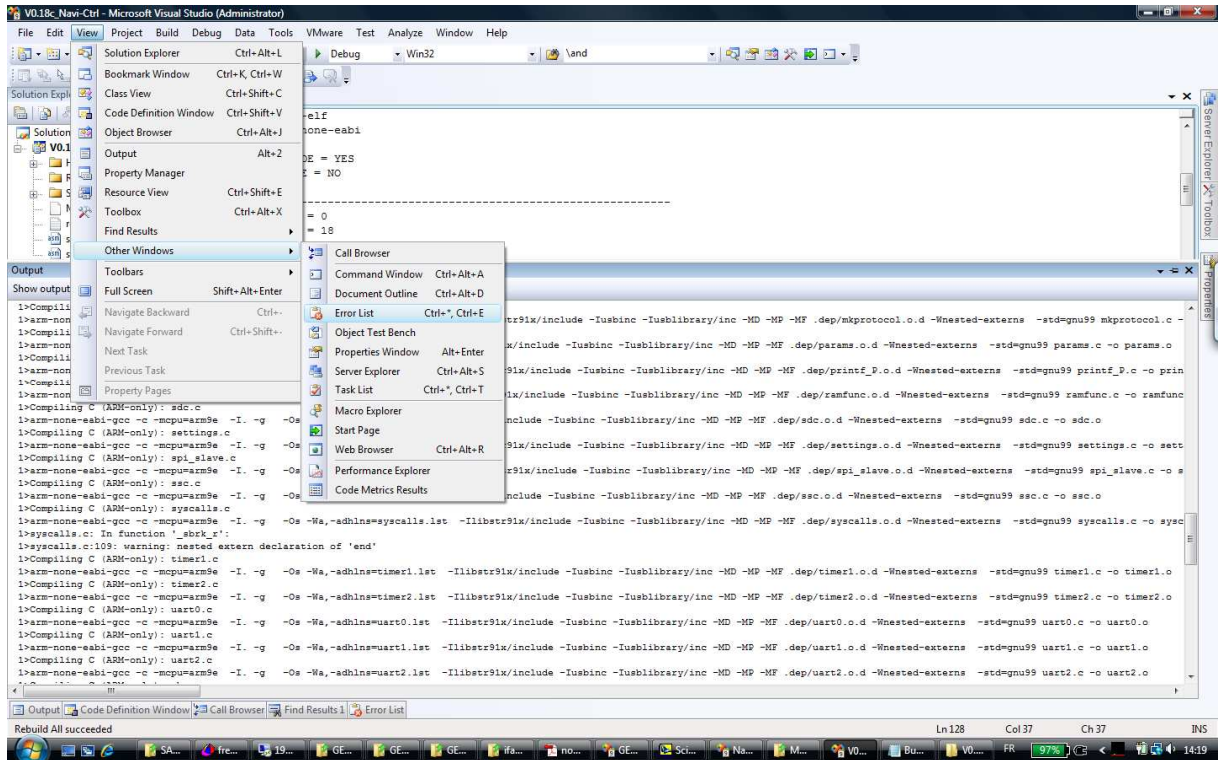
On peut voir dans la fenêtre Output de Visual Studio les commandes exécutées par le Makefile.



Voici un exemple d'avertissement au cours de la compilation.



On voit ici qu'il est bien listé dans la fenêtre Error List.



Si on double-clique sur le message, le code provoquant le message est affiché dans la fenêtre principale.

