

## Using Multiple Compilation Units Guide

### Files Included in Project Example:

main.c	Primary file for the first compilation unit.
filter.c	Primary file for the second compilation unit.
report.c	Primary file for the third compilation unit.
project.h	Include file with project wide definitions that should be included by all units.
filter.h	Include file with external definitions for filter that should be included by all units that use the filter unit.
report.h	Include file with external definitions for report that should be included by all units that use the report unit.
project.c	Import file used to list the units in the project for the linker.bat file.
project.pjt	Project file used to list the units in the project for the build.bat file.
build.bat	Batch file that re-compiles files that need compiling and linking.
buildall.bat	Batch file that compiles and links all units.
linker.bat	Batch file that compiles and links all units using a script.

### File Overview:

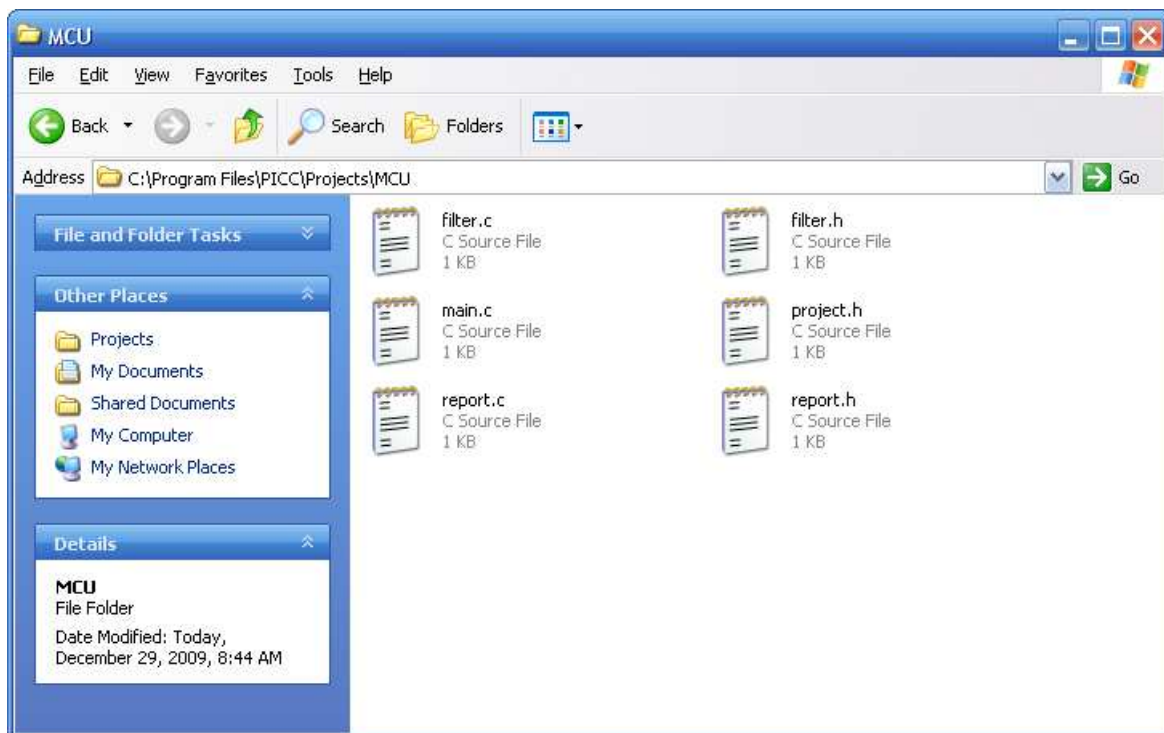
MAIN.C	FILTER.C	REPORT.C
<pre>#include:     • project.h     • filter.h     • report.h  Definitions:     • main( ) program  Uses:     • clear_data( )     • filter_data( )     • report_data_line( )     • report_line_number( )</pre>	<pre>#include:     • project.h     • report.h  Public Definitions:     • clear_data( )     • filter_data( )  Uses:     • report_error( )</pre>	<pre>#include:     • project.h  Public Definitions:     • report_data_line( )     • report_line_number( )     • report_error( )</pre>

## Compilation Files:

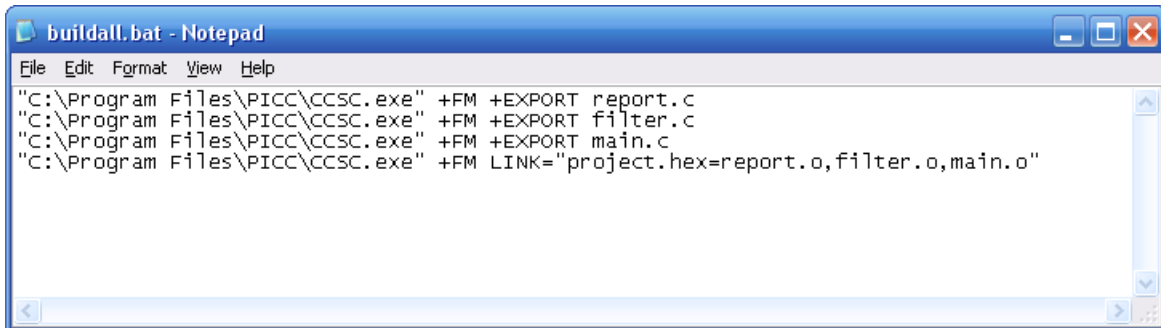
*.o	Relocatable object file that is generated by each unit.
*.err	Error file that is generated by each unit.
*.osym	Unit symbol file that is generated by each unit.
project.hex	Final load image file generated by the project.
project.lst	C and ASM listing file generated by the project.
project.sym	Project symbols file generated by the project.
project.cof	Debugger file generated by the project.

## Using Command Line to Build a Project:

- Move all of the source files for the project into a single directory.



- Using a text editor, create the file **buildall.bat**, based off of the following example in order to compile the files and build the project.
  - The path should point to the **CCSC.exe** file in the PIC-C installation directory.
  - Add any additional compiler options.
  - Use the EXPORT option to include the necessary \*.c files.
  - Use the LINK option to generate a \*.hex file.

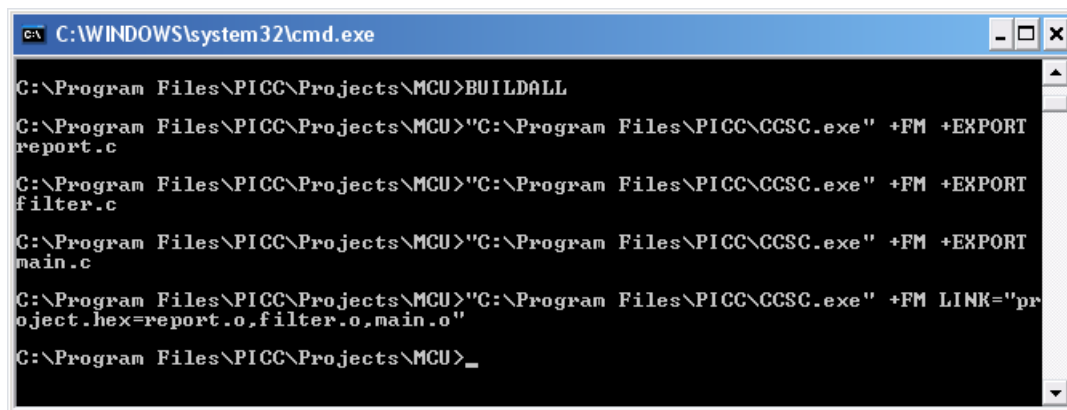


```

"C:\Program Files\PICC\CCSC.exe" +FM +EXPORT report.c
"C:\Program Files\PICC\CCSC.exe" +FM +EXPORT filter.c
"C:\Program Files\PICC\CCSC.exe" +FM +EXPORT main.c
"C:\Program Files\PICC\CCSC.exe" +FM LINK="project.hex=report.o,filter.o,main.o"

```

- Double-click on the **buildall.bat** file created earlier or use a command prompt by changing the default directory to the project directory. Then use the command BUILDALL to build the project using all of the files.



```

C:\WINDOWS\system32\cmd.exe

C:\Program Files\PICC\Projects\MCU>BUILDALL

C:\Program Files\PICC\Projects\MCU>"C:\Program Files\PICC\CCSC.exe" +FM +EXPORT
report.c

C:\Program Files\PICC\Projects\MCU>"C:\Program Files\PICC\CCSC.exe" +FM +EXPORT
filter.c

C:\Program Files\PICC\Projects\MCU>"C:\Program Files\PICC\CCSC.exe" +FM +EXPORT
main.c

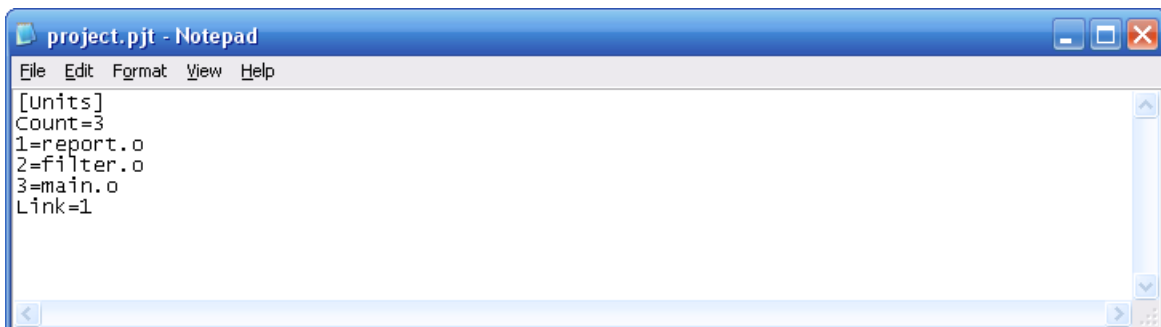
C:\Program Files\PICC\Projects\MCU>"C:\Program Files\PICC\CCSC.exe" +FM LINK="pr
oject.hex=report.o,filter.o,main.o"

C:\Program Files\PICC\Projects\MCU>_

```

### Using Command Line to Re-Build Changed Files in a Project:

- Using a text editor, create the file **project.pjt** based off of the following example in order to include the files that need to be linked for the project.

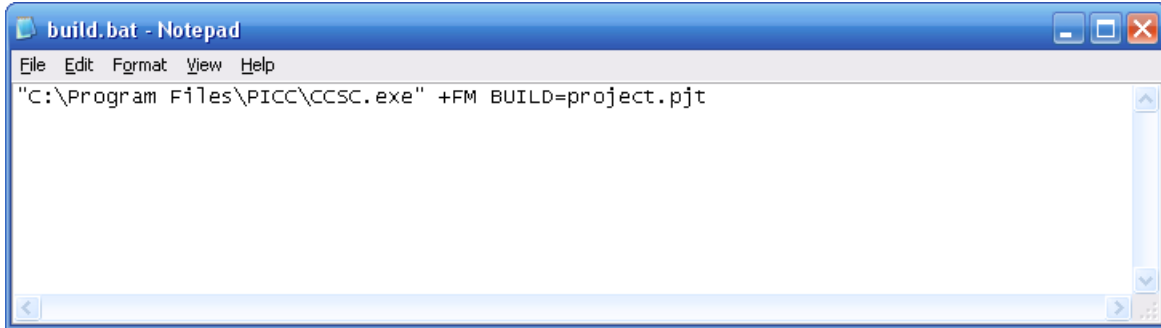


```

[Units]
Count=3
1=report.o
2=filter.o
3=main.o
Link=1

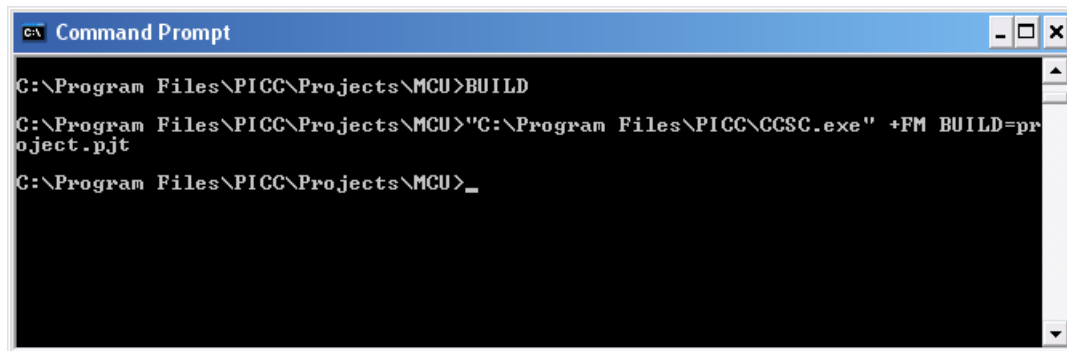
```

- Using a text editor, create the file **build.bat** based off of the following example in order to compile only the files that changed and re-build the project.
  - The path should point to the **CCSC.exe** file in the PIC-C installation directory.
  - Add any additional compiler options.
  - Use the BUILD option to specify the \*.pjt file.



```
build.bat - Notepad
File Edit Format View Help
"C:\Program Files\PICC\CCSC.exe" +FM BUILD=project.pjt
```

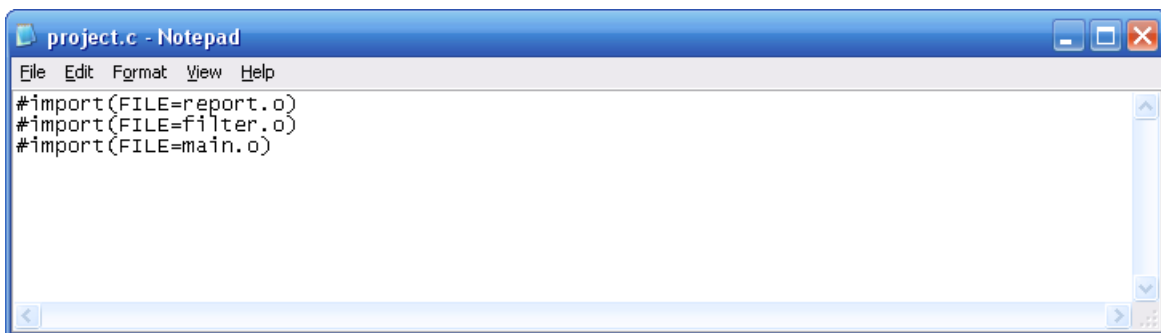
- Double-click on the **build.bat** file created earlier or use a command prompt by changing the default directory to the project directory and then use the command BUILD to re-build the project using only the necessary files that changed.



```
C:\ Command Prompt
C:\Program Files\PICC\Projects\MCU>BUILD
C:\Program Files\PICC\Projects\MCU>"C:\Program Files\PICC\CCSC.exe" +FM BUILD=project.pjt
C:\Program Files\PICC\Projects\MCU>_
```

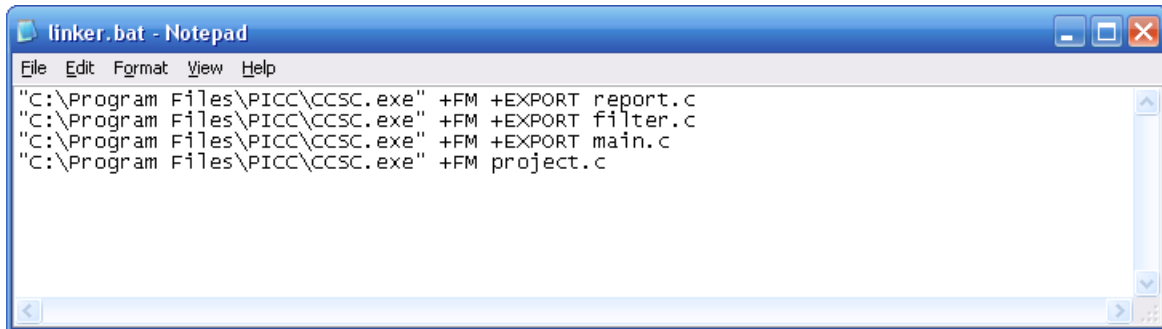
### Using a Linker Script:

- Using a text editor, create the file **project.c** based off of the following example in order to include the files that need to be linked for the project.



```
project.c - Notepad
File Edit Format View Help
#import(FILE=report.o)
#import(FILE=filter.o)
#import(FILE=main.o)
```

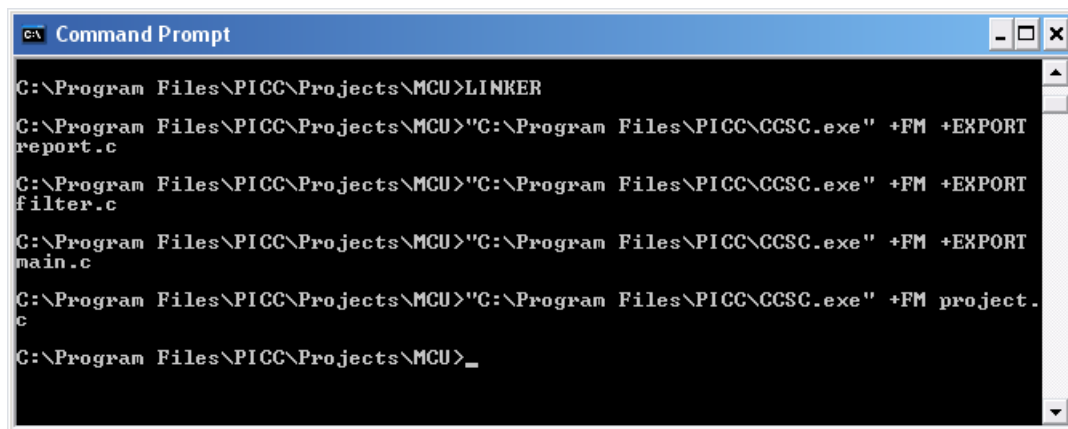
- Using a text editor, create the file **linker.bat** based off of the following example in order to compile the files and build the project.
  - The path should point to the **CCSC.exe** file in the PIC-C installation directory.
  - Add any additional compiler options.
  - Use the EXPORT option to include the necessary \*.c files.
  - The LINK option is replaced with the \*.c file containing the #import commands.



```

linker.bat - Notepad
File Edit Format View Help
"C:\Program Files\PICC\CCSC.exe" +FM +EXPORT report.c
"C:\Program Files\PICC\CCSC.exe" +FM +EXPORT filter.c
"C:\Program Files\PICC\CCSC.exe" +FM +EXPORT main.c
"C:\Program Files\PICC\CCSC.exe" +FM project.c
  
```

- Double-click on the **linker.bat** file created earlier or use a command prompt by changing the default directory to the project directory and then use the command LINKER to build the project using all of the files.

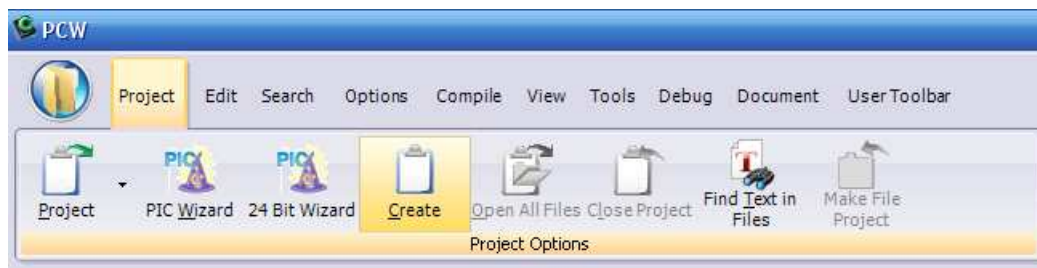


```

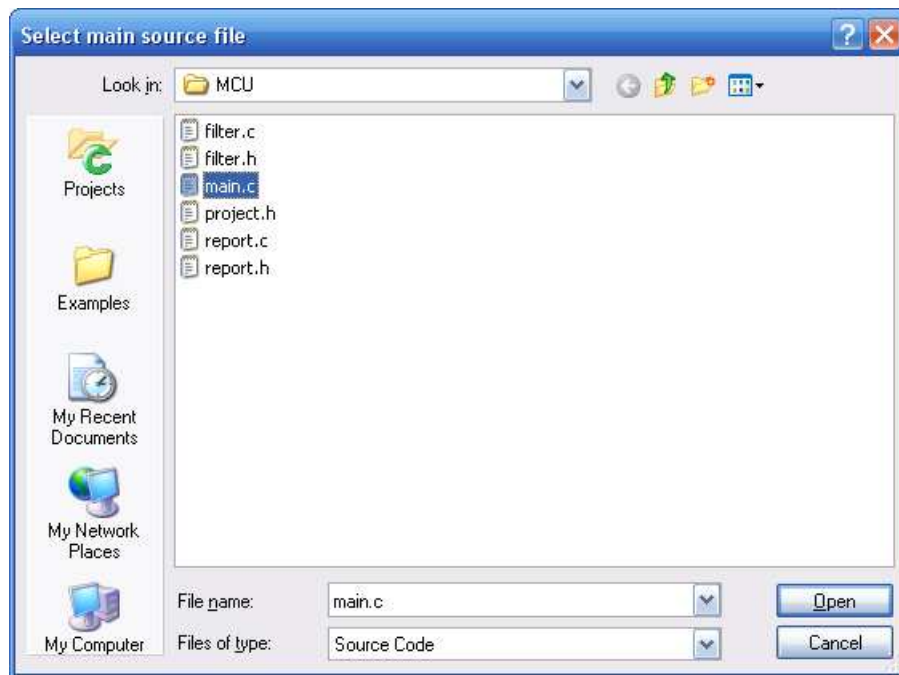
C:\ Command Prompt
C:\Program Files\PICC\Projects\MCU>LINKER
C:\Program Files\PICC\Projects\MCU>"C:\Program Files\PICC\CCSC.exe" +FM +EXPORT
report.c
C:\Program Files\PICC\Projects\MCU>"C:\Program Files\PICC\CCSC.exe" +FM +EXPORT
filter.c
C:\Program Files\PICC\Projects\MCU>"C:\Program Files\PICC\CCSC.exe" +FM +EXPORT
main.c
C:\Program Files\PICC\Projects\MCU>"C:\Program Files\PICC\CCSC.exe" +FM project.
c
C:\Program Files\PICC\Projects\MCU>_
  
```

### Using the CCS PCW IDE with Multiple Compilation Units:

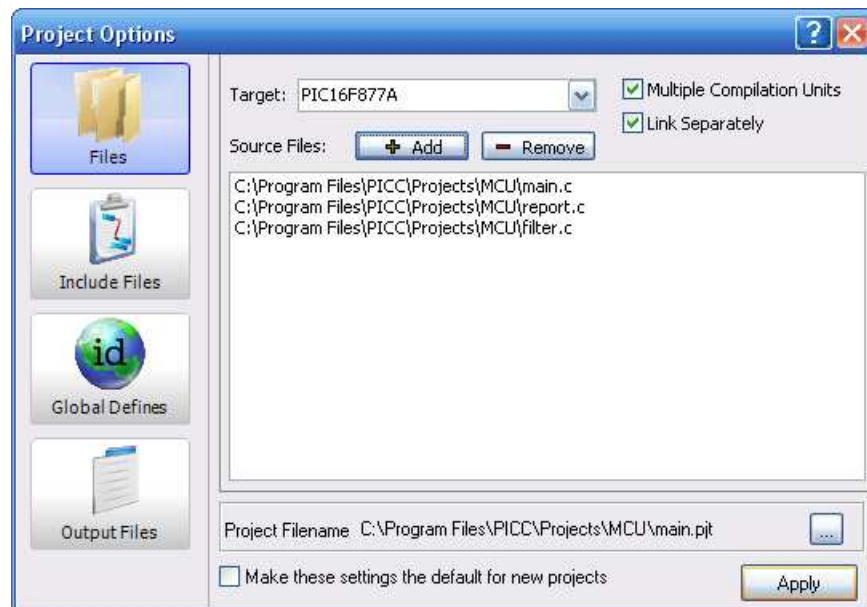
- Open the PCW IDE and select the **Project** tab in the ribbon along the top of the main window or in the menu bar if the IDE view style has been changed, then select the **Create** option.



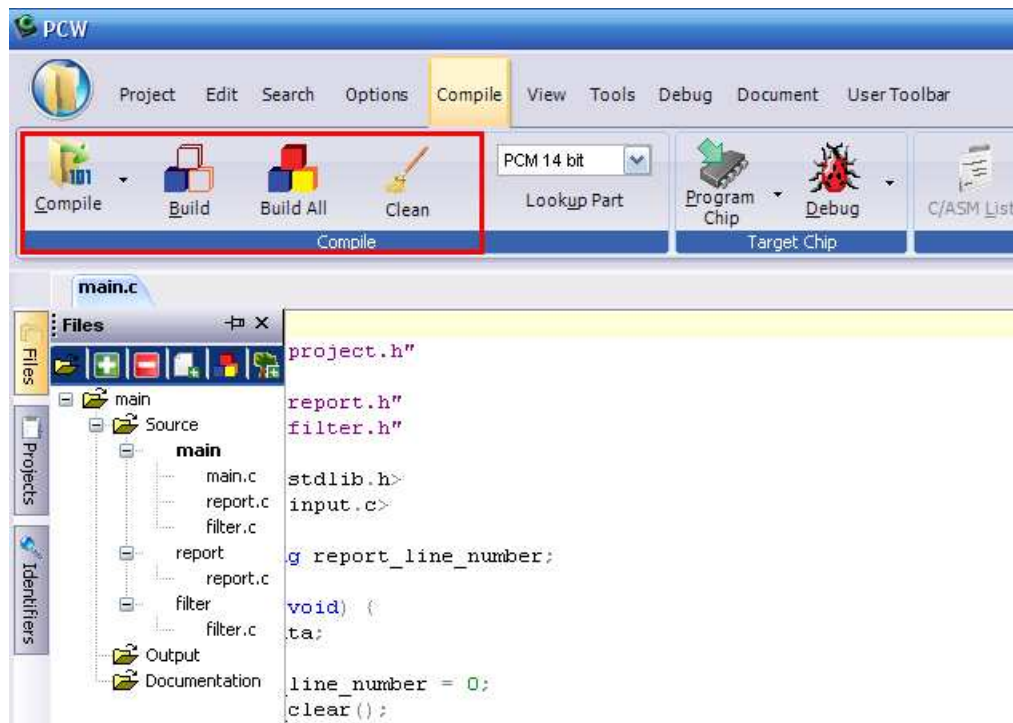
- A window will be displayed asking to select the main source file of the project.



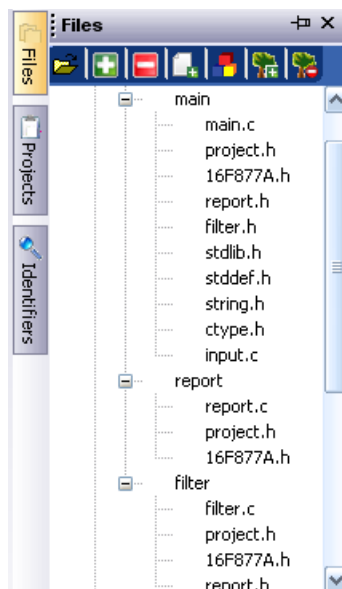
- After selecting the main source file the **Project Options** window will appear. In this window, select the type of chip being used in the project. Then, check the boxes next to the **Multiple Compilation Units** and **Link Separately** options. This will allow additional source files to be added. Click the **Add** button and select the other source files used in the project to add them to the list. Click the **Apply** button to create the project.



- To compile the files in a project and build the project itself, select either the **Compile** tab in the ribbon along the top of the main window, in the menu bar if the IDE view style has been changed, or right-click on the files in the **Files** pane along the left side of the editor window.
  - Compile: Compiles all the units in the current project or a single unit selected from the drop-down menu.
  - Build: Compiles units that have changed since the last compile and rebuilds the project.
  - Build All: Compiles all the units and builds the project.
  - Clean: Deletes the output files for the project.

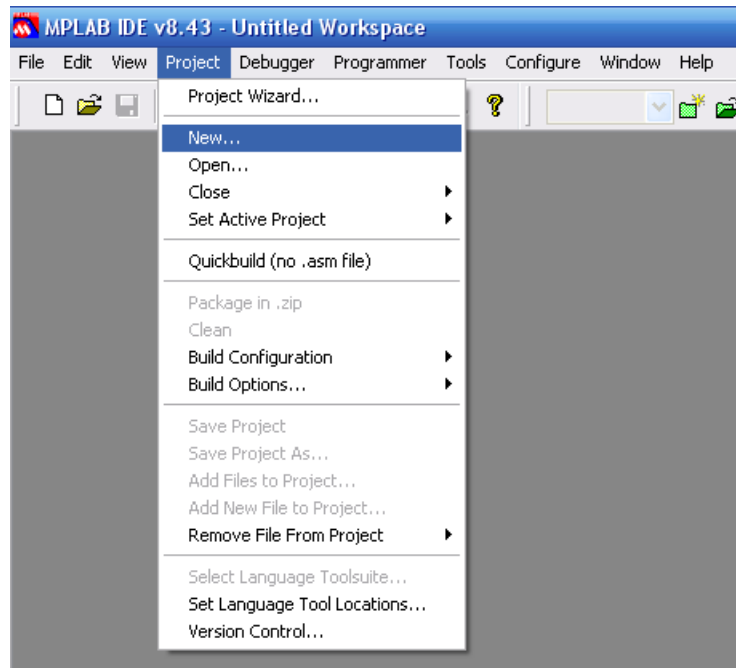


- After a file has been compiled, the files used during the compilation will appear under the unit's name in the **Files** pane.

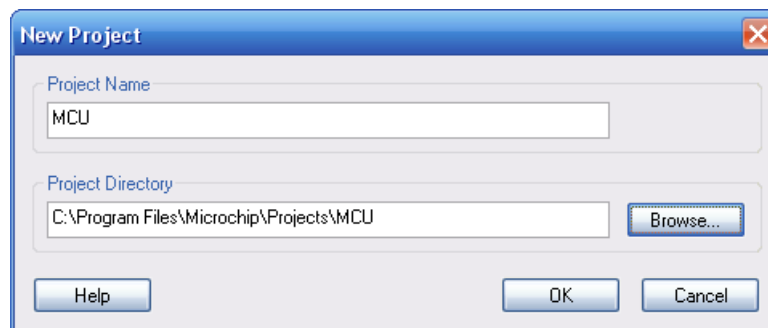


## Using the MPLAB IDE with Multiple Compilation Units:

- Open the MPLAB IDE, select the **Project** tab in the menu bar and select the **New** option.

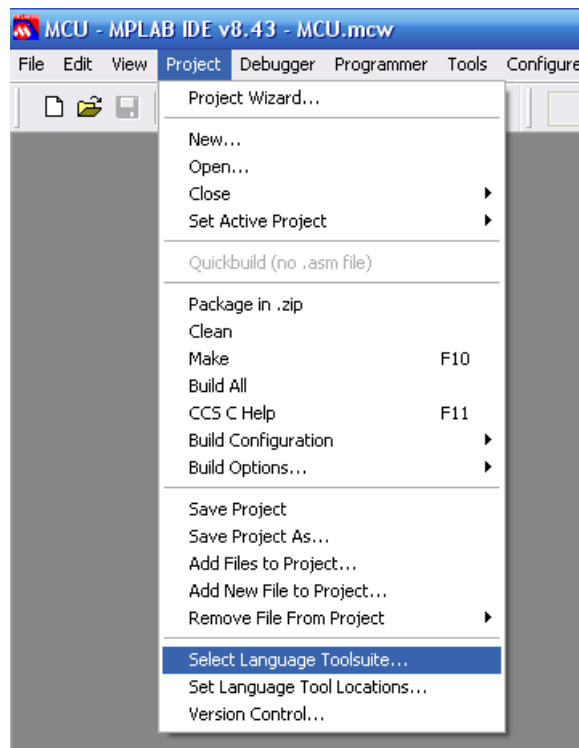


- A window will be displayed asking to select the main source file of the project.

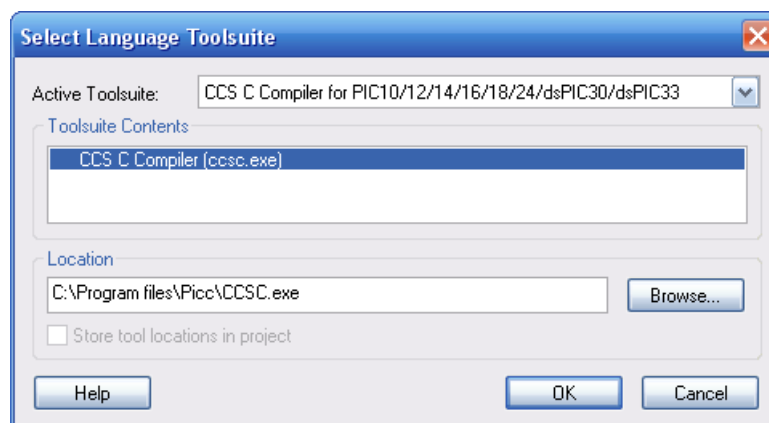




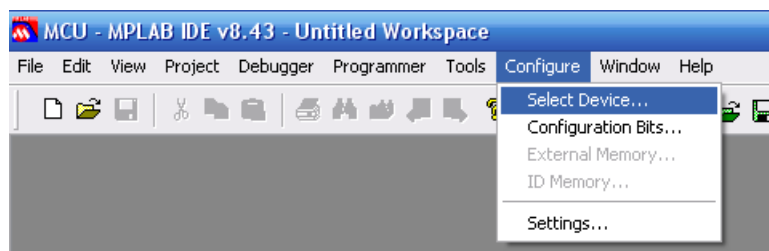
- Select the **Project** tab in the menu bar and select the **Select Language Toolsuite** option.



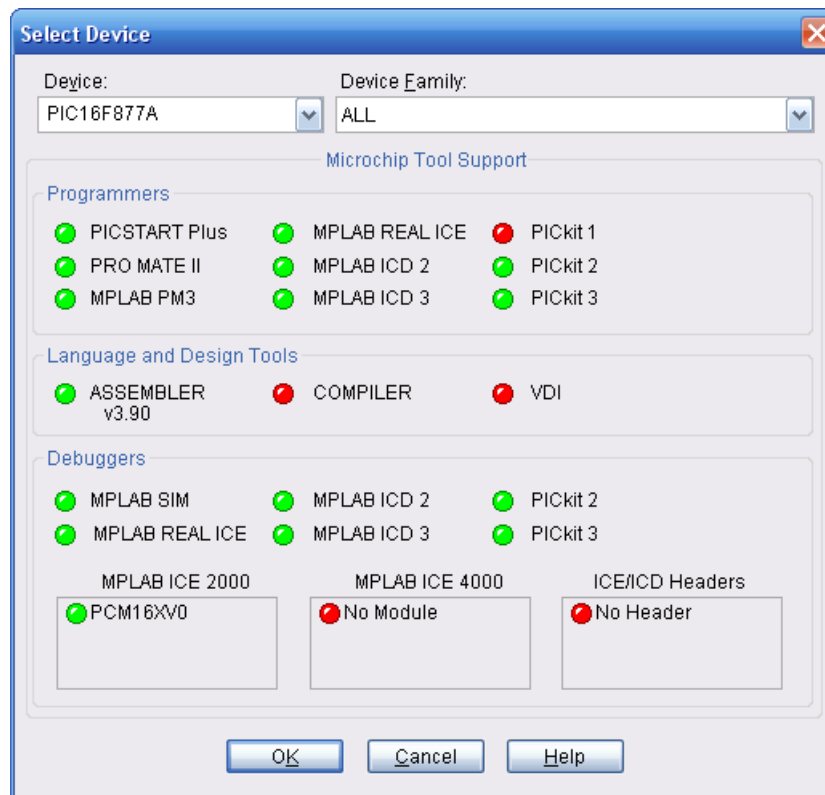
- A window will be displayed, select the **CCS C Compiler** from the drop-down list in the **Active Toolsuite** field. Make sure the correct directory location is displayed for the compiler.



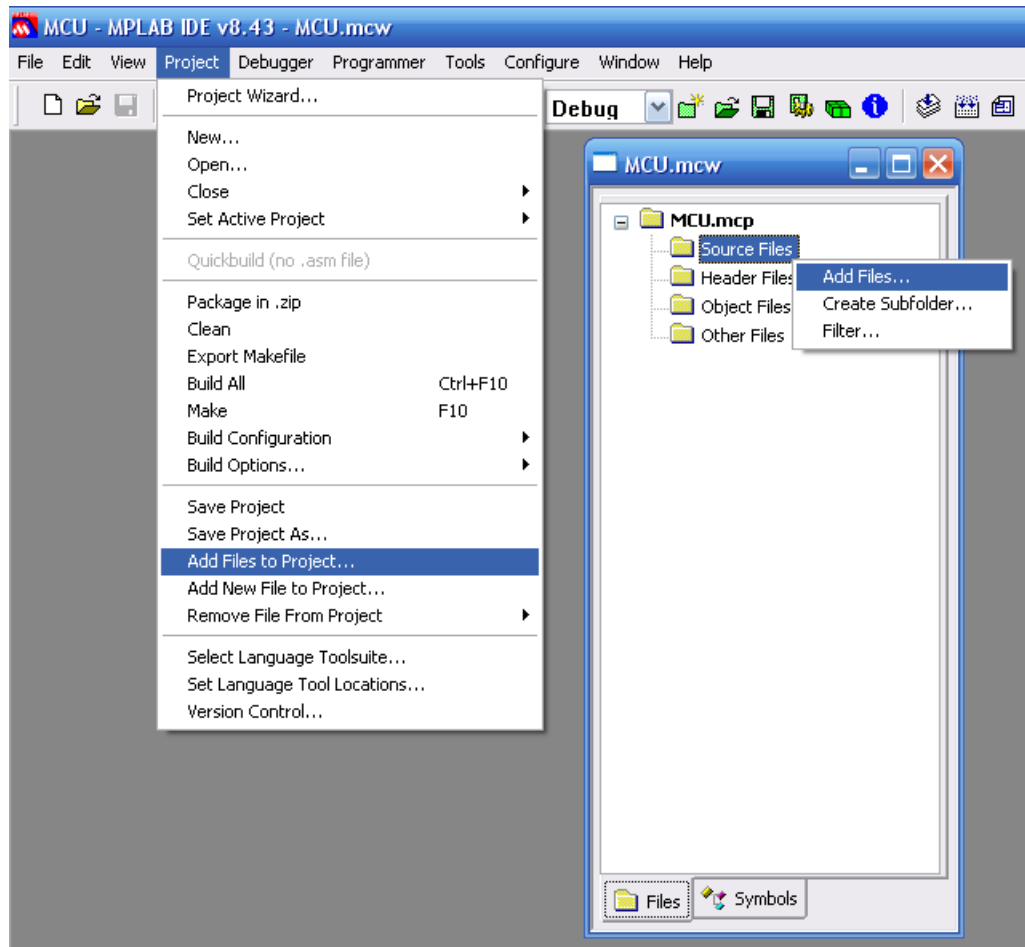
- Select the **Configure** tab in the menu bar and select the **Select Device** option.



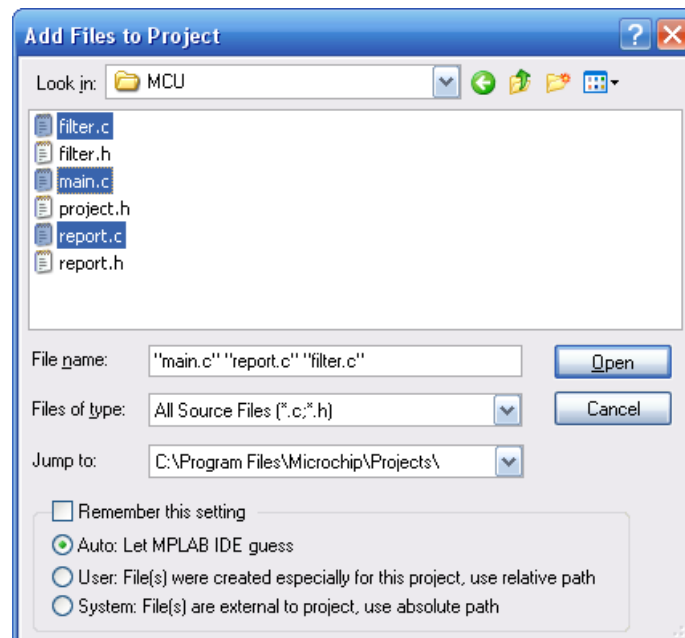
- A window will be displayed, select the correct PIC from the list provided.



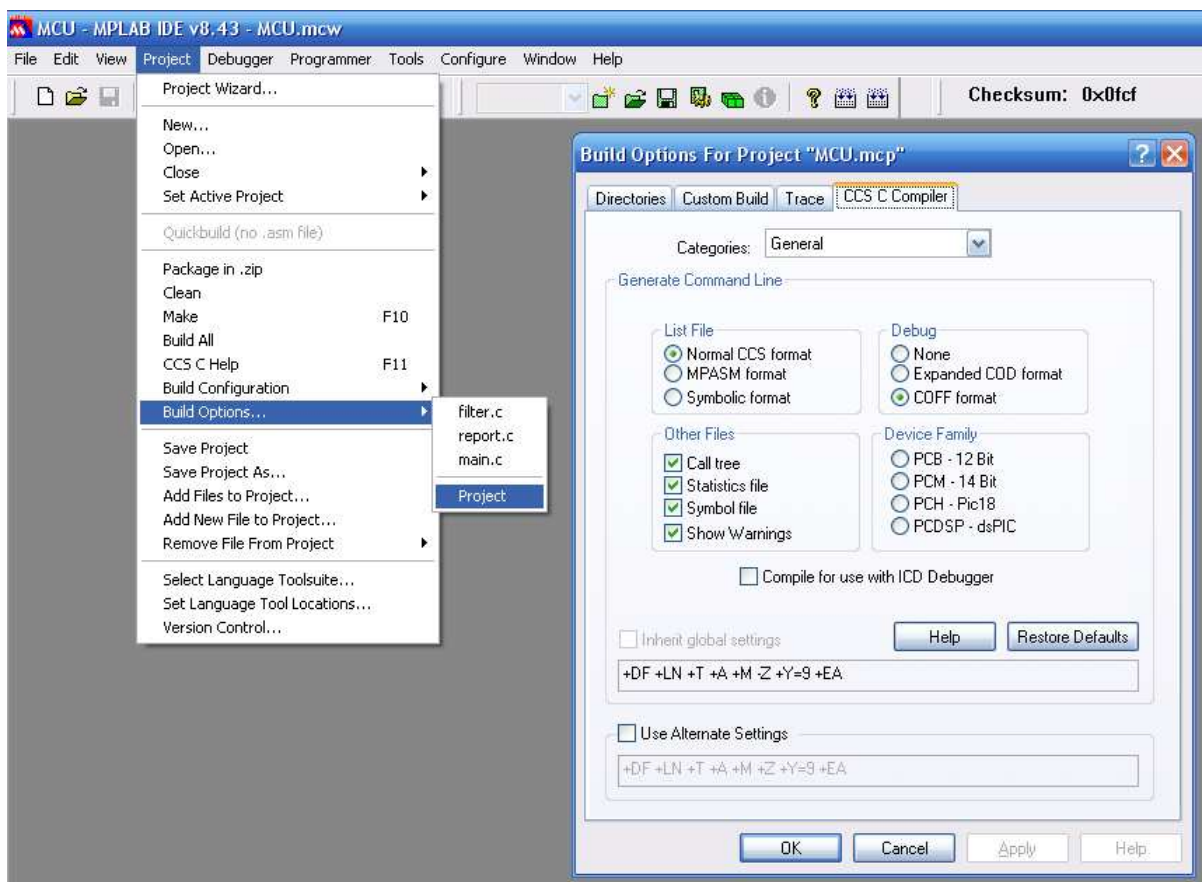
- Add source files to the project by either selecting the **Project** tab in the menu bar and then the **Add File to Project** option or by right-clicking on the **Source Files** folder in the project window and selecting **Add Files**.



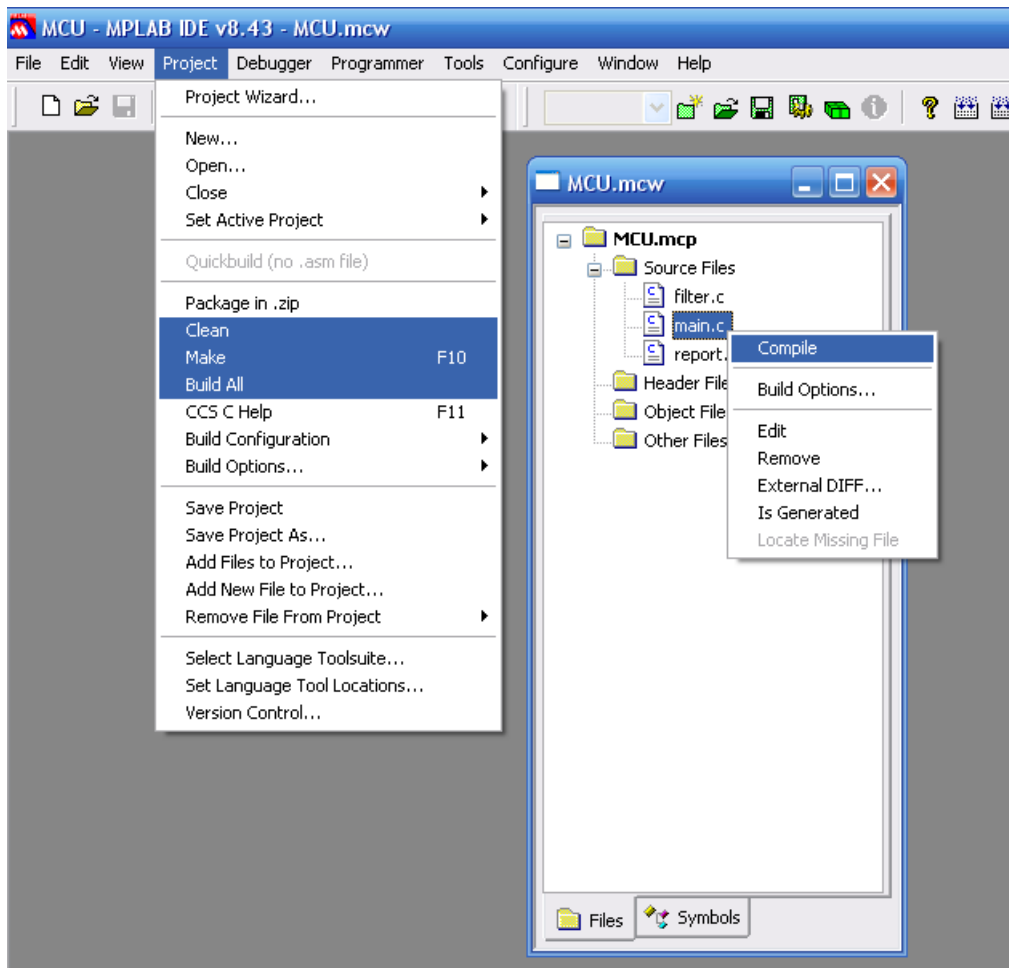
- A window will be displayed, select the source files to add to the project.



- Select the **Project** tab in the menu bar and select **Build Options**. This will allow changes to be made to the output directory, include directories, output files generated, etc for the entire project or just individual units.



- To compile the files in a project and build the project itself, select either the **Project** tab in the menu bar or right-click on the files in the **Project** window.
  - **Compile:** Compiles the selected unit and will not re-link the project after compilation.
  - **Make:** Compiles units that have changed since the last compile and rebuilds the project.
  - **Build All:** Compiles all the units, deletes intermediate files, and builds the project.
  - **Clean:** Deletes the output files for the project.



- **Additional Note:** If there is only one source file in the project, it will be compiled and linked in one step, a \*.o file will not be created. A \*.o file, that has already been compiled can be added to the project and linked during the make / build process.

## Additional Notes

- To make a variable or function private to a single unit, use the keyword **static**. By default, variables declared outside a function at the unit level are visible to all other units in the project. If the **static** keyword is used on a function or variable that is accessed outside of the local unit, a link time error will occur.
- If two units have a function or a unit level variable of the same name, an error will occur unless one of the following conditions is true:
  - The identifier is qualified with the keyword **static**.
  - The argument list is different for both functions, allowing them to co-exist according to normal overload rules.
  - The contents of the functions are identical, such as when the same \*.h file is included in multiple files, then the linker will delete the duplicate functions.
- For a project with multiple compilation units, it is best to include a file such as **project.h** which includes the #includes, #defines, pre-processor directives, and any other compiler settings that are the same for all the units in a project.
  - When a setting such as a pre-processor directive is included in the main include file between the units, a library is created in each of the units. The linker is able to determine that the libraries are duplicates and removes them during the final linking process.
- When building a project, each unit being used in the project has its own error file. When using a \*.bat file to do the unit compilations, it may be useful to terminate the process on the first error. Using the +CC command option, the compiler will return an error code if the compilation fails.