

## Multiple Compilation Unit Example

### Files in project:

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, should be included by all units
filter.h	External definitions for filter, should be included by all units that use the filter unit
report.h	External definitions for report, should be included by all units that use report
buildall.bat	Batch file that compiles and links all units
build.bat	Batch file that recompiles files needing compiling and links
project.pjt	Used by build.bat to list project units



### Each unit:

- \*.o (relocatable object)
- \*.err (error file)
- \*.osym (unit symbols)

project.hex (final load image)  
project.lst (C and ASM listing)  
project.sym (project symbols)  
project.cof (debugger file)

### Building the project from the command line:

1. Move the project files into a directory.
2. Edit the buildall.bat file and make sure the path to CCSC.EXE is correct.
3. From a DOS prompt set the default directory to the project directory.
4. Enter: BUILDALL

```
"c:\program files\picc\ccsc" +FM +EXPORT report.c  
"c:\program files\picc\ccsc" +FM +EXPORT filter.c  
"c:\program files\picc\ccsc" +FM +EXPORT main.c  
"c:\program files\picc\ccsc" +FM LINK="project.hex=report.o,filter.o,main.o"
```

### Automatically building by recompiling needed files:

1. The required lines in the project.pjt file are:  
    [Units]  
    Count=3  
    1=filter.o  
    2=report.o  
    3=main.o  
    Link=1
2. From a DOS prompt set the default directory to the project directory.
3. Enter: BUILD

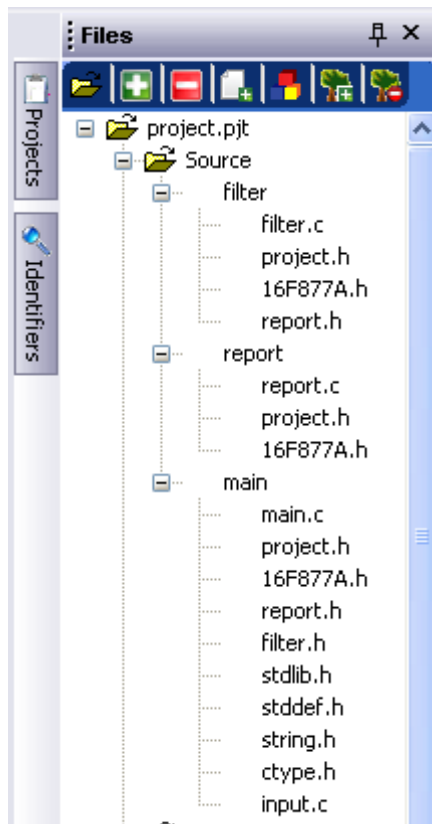
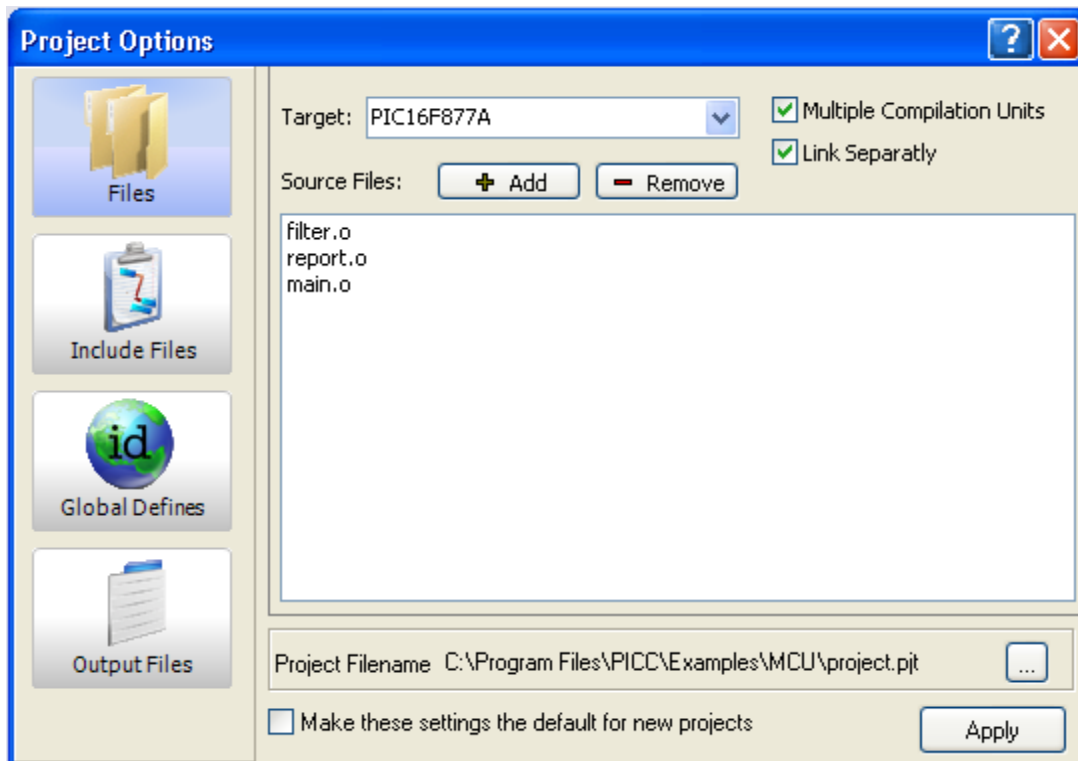
Note that after a project is linked if no .pjt file exists the linker will create one that may be used with the BUILD= option in the future.

```
"c:\program files\picc\ccsc" +FM BUILD=project.pjt
```

### Replacing the linker command line with a linker script:

1. Create a file named project.c with the following lines:  
    #import( report.o )  
    #import( filter.o )  
    #import( main.o )
2. Compile each unit (report, filter, main).
3. Compile project.c

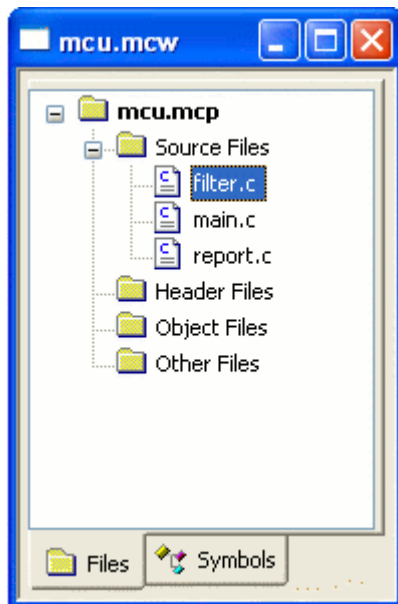
## Using the IDE to work with multiple compilation units:



- The above screen is from OPTIONS > PROJECT OPTIONS after loading the project.pjt file. If the file does not exist create the project manually and make a screen like the above.
- The pane to the left is the FILES slide out that is available from VIEW > PROJECT FILES.
- Right click on a unit name (like filter) and select COMPILE to compile just that unit.
- Click on the build icon (third from the right) to rebuild and link the whole project.
- This pane is helpful in managing each unit in the project. Review the right click options for the full range of options.

## Using MPLAB IDE to work with Multiple Compilation Units

- Create a new project by selecting “Project -> New” from the toolbar. Follow the dialog boxes to specify the project name and project path.
- Make sure MPLAB is configured for the proper chip, as the CCS C compiler uses this selection to determine which compiler to use (PCB, PCM, PCH, PCD, etc). The chip can be selected using “Configure -> Select Device” from the MPLAB toolbar.



*MPLAB Project window*

- Add source files by either a.) right clicking on 'Source Files' in the MPLAB Project window or b.) selecting “Project -> Add New File to Project..” from the MPLAB toolbar.
  - Performing a Make (hotkey is F10) or Build All will compile the source files separately, and link the .o files in the final step. Make only compiles files that have changed, Build All will delete all intermediate files first and then compile all files regardless if they have changed since last build.
  - An individual unit can be compiled by right clicking on the file in the MPLAB Project window and choosing 'Compile.' This will not re-link the project when it is done compiling this unit.
- An already compiled .o file can be added to the project, and will be linked during the Make/Build process.
  - If there is only one source in the project, it will be compiled and linked in one phase (no .o file will be created).
  - Many project build options (such as output directory, include directories, output files generated, etc) can be changed by selecting “Project -> Build Options” from the MPLAB toolbar.
  - If the compile fails with an error that says something like “Target chip not supported” or “Compiler not found” make sure that **a.)** you have the proper PIC selected (use “Configure -> Select Device” from the MPLAB toolbar), **b.)** the CCS C Toolsuite has been selected for this project (use “Project -> Set Language Toolsuite” from the MPLAB toolbar) and **c.)** the path for CCSC.EXE is configured correctly for your installation of the CCS C Compiler (use “Project -> Set Language Tool Locations” on the MPLAB toolbar)

## Notes

- By default variables declared at the unit level (outside a function) are visible to all other units. To make a variable private to the unit use the keyword **static**. Notice report.c defines the variable **report\_line\_number**. If the definition were changed to look as the following line then there would be a link time error since main.c attempts to use the variable.

```
static long report_line_number;
```

- This same rule applies to functions. Use **static** to make a function local to the unit.
- Should two units have a function or unit level variable with the same name an error is generated unless one of the following is true:
  - The identifier is qualified with **static**.
  - The argument list is different and two instances of the function can co-exist in the project in accordance with the normal overload rules.
  - The contents of the functions are absolutely identical. In this case the CCS linker simply deletes the duplicate function.
- The standard C libraries (like stdlib.h) are supplied with source code in the .h file. Because of the above rule these files may be #include'd in multiple units without taking up extra ROM and with no need to include these in the link command since they are not units.
- #define's are never exported to other units. If a #define needs to be shared between units put them in an include file that is #include'd by both units. Project wide defines in our example could go into project.h.
- It is best to have an include file like project.h that all units #include. This file should define the chip, speed, fuses and any other compiler settings that should be the same for all units in the project.
- In this example project a #USE RS232 is in the project.h file. This creates an RS232 library in each unit. The linker is able to determine the libraries are the same and the duplicates removed in the final link.
- Each unit has its own error file (like filter.err). When the compilations are done in a batch file it may be useful to terminate the batch run on the first error. The +CC command line option will cause the compiler to return a windows error code if the compilation fails. This can be tested in the batch file like this:

```
"c:\program files\picc\ccsc" +FM +CC +EXPORT report.c
if not errorlevel 1 goto abort
...
goto end
:abort
echo COMPILE ERROR
:end
```