CMPE1250 – Using Visual Studio Code as an Editor

CodeWarrior natively supports edit, build, debug, and target load operations. Because of this, it may be used as the sole tool for creating embedded projects on your micro. It is, sadly, a particularly poor code editor.

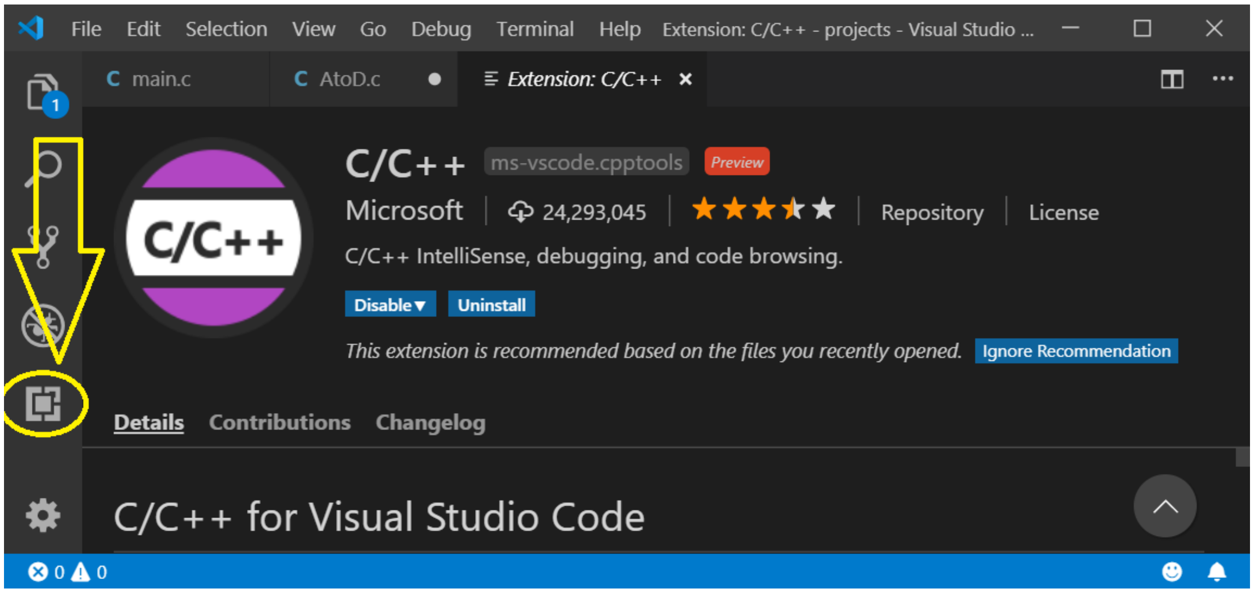
The derivative file that was created to support your chip contains thousands of definitions that could assist you in code development, but because CodeWarrior refuses to reliably provide popup support for included libraries, much of this boon is lost.

You may wish to use an alternate code editor – one capable of providing the assistive tools that you are accustomed to, as you have seen in C#, for example. Visual Studio could be such an editor, but it is a little heavy-handed. Instead, consider using Visual Studio Code. Code is a lightweight editor that provides the desired editing capability, without the heavy footprint.

Installing

Visual Studio Code can be downloaded from Microsoft for free: https://code.visualstudio.com/
It is suggested that you install the 'User', not 'System' version, unless you are the only user of the system you are installing it on. Version 1.47.3 shown at the time of print - things may look a little different for you.

Code supports many different languages, so you will need to install the language support you require, in this case, the C language. Such extensions are downloaded and installed from within Code, using the Extensions option. You should grab the C/C++ extension from Microsoft (Version 0.29.0 at the time of print):



C/C++ IntelliSense, debugging, and code browsing.

This extension is recommended based on the files you recently opened.

C/C++ for Visual Studio Code

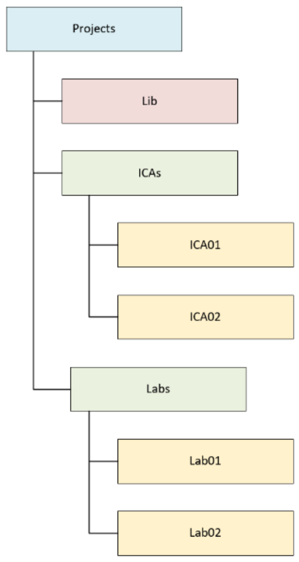
Other instructors may provide similar requests for other languages (especially in Web Technologies).

You will now have the ability to edit C code with all the benefits of popups and syntax highlighting.

Organization

When you set out to create projects for your micro, you should consider a few things. Each project will contain many files, contained in a folder for the project. Name your project appropriately, so that you know what the project is. Be considerate of future you.

You will be developing libraries (compilation units) that will be shared among various projects. To be visible to all projects they should be placed in a separate folder, probably at the same level or higher than the projects themselves, which means all of your work should be in an über folder. The overall directory structure should look something like this:



The *Projects* folder contains everything that you have and will code (root).

The *Lib* folder contains compilation units that are the shared libraries between projects. You only want one copy of these to save version control nightmares. You will be creating several library compilation units.

Optionally you may create a *ICAs* and *Labs* folders for organizational purposes, but with proper project naming, this is unnecessary.

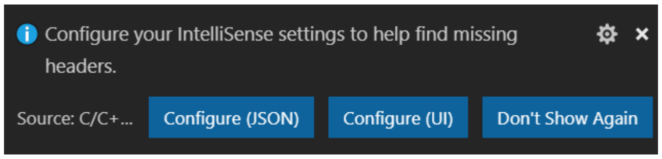
Each project will reside in its own folder (created by CodeWarrior) and will refer to libraries up the folder tree. Your instructor will eventually show you how to add a compilation unit in a separate folder to your project.

Additionally, you may be required by your instructor to source control your project tree. If this is the case, you will be given additional instructions.

You should create this folder structure now and stick to it. You should always treat the *entire* folder structure as a unit (meaning that if you copy/move/source control it, you do it to the entire folder, from the root).

Sadly, after first installation, Code will not be aware of the file locations for headers and libraries that you will wish to include in your projects. You will need to add a search path to include the location of the derivative file and standard include files.

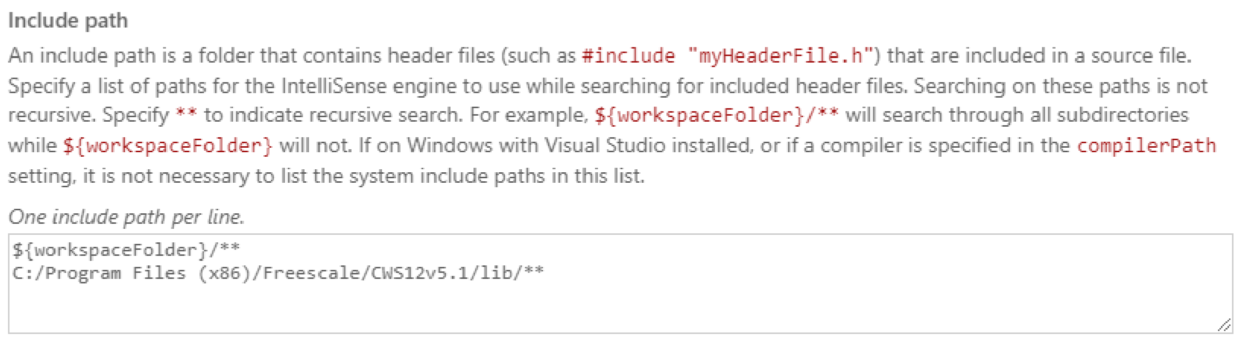
Note: if you fail to correct the search path, the following popup will typically appear when Code attempts to resolve symbol names and references:


Configure your IntelliSense settings to help find missing headers.

Source: C/C+...

To prevent this, use the menu 'View \ Command Palette' and select the 'C/C++: Edit Configurations (UI)' option.

Under the 'Include path' option, add the following paths (where not already present):



Include path

An include path is a folder that contains header files (such as `#include "myHeaderFile.h"`) that are included in a source file. Specify a list of paths for the IntelliSense engine to use while searching for included header files. Searching on these paths is not recursive. Specify `**` to indicate recursive search. For example, `${workspaceFolder}/**` will search through all subdirectories while `${workspaceFolder}` will not. If on Windows with Visual Studio installed, or if a compiler is specified in the `compilerPath` setting, it is not necessary to list the system include paths in this list.

One include path per line.

```
${workspaceFolder}/**
C:/Program Files (x86)/Freescale/CWS12v5.1/lib/**
```

The bottom line is necessary to include the system includes and the derivative files. Edit the paths as necessary to reflect your install location for CodeWarrior (the above is the default). You may include other locations as well. Recursively searching for files from these two locations should (eventually) locate all files you are referencing. VSCode works on folders (you use the 'File \ Open Folder' option), so in Code you would open the entire *Projects* folder. In fact, you will likely need to open a folder before this entire operation is possible (and setting are saved per folder, in the target folder).

Using

As stated earlier, Code is a good editor. It is not configured to compile, debug, or load your projects. For this, we will continue to use CodeWarrior. In fact, even with the correct paths resolved, **Code will still generate errors for some elements in your code**, and we cannot easily get around this – you will need to learn what to pay attention to, and what to ignore.

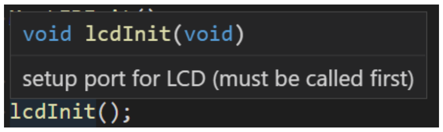
Project creation and management will all occur within CodeWarrior. Once this is done you can edit the files in Code. If you save the files in Code, CodeWarrior will see this and ask to reload them. It is a little clumsy switching between the two applications, but it will become natural, and the benefits are worth it. Your instructor will demonstrate the workflow, so you will get a chance to see the procedure.

Advantage

So what do you get for all of the trouble of setting this up and having to switch between editors?

- The VSCode editor is much newer and provides better syntax highlighting
- The VSCode editor will successfully autocomplete and provide popups in a familiar way (based on all available headers)
- The VSCode editor will apply comments as evidence in popups

This may not seem like much, but you will need every advantage you can get for the lab exams. Comments on prototypes will show up in the editor:

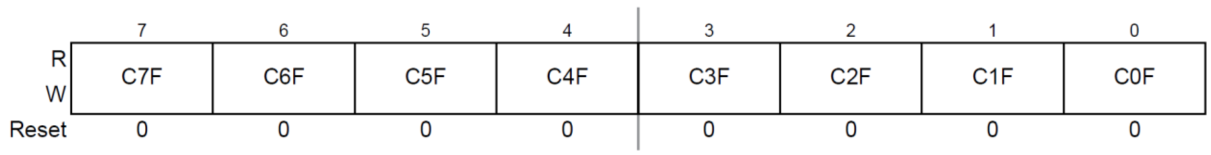


```
void lcdInit(void)
setup port for LCD (must be called first)
lcdInit();
```

If you document your code well, you will be jumping around a lot less in your files, attempting to figure out what you did weeks ago, or what that weird parameter is for.

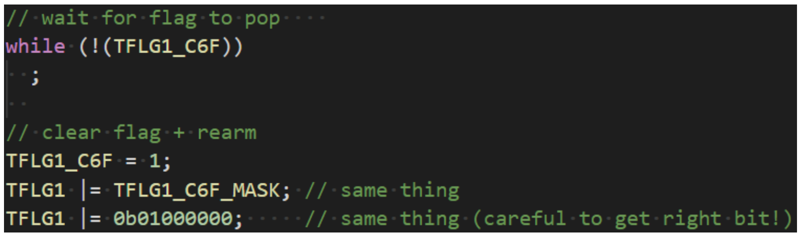
The biggest advantage and one that will not mean anything to you yet, is the fact that every register and every bit in that register is named in the derivative file, as described in Big Pink. If you look up a module's behavior in Big Pink, you need little effort to translate that into code:

7.3.2.12 Main Timer Interrupt Flag 1 (TFLG1)



	7	6	5	4	3	2	1	0
R W	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F
Reset	0	0	0	0	0	0	0	0

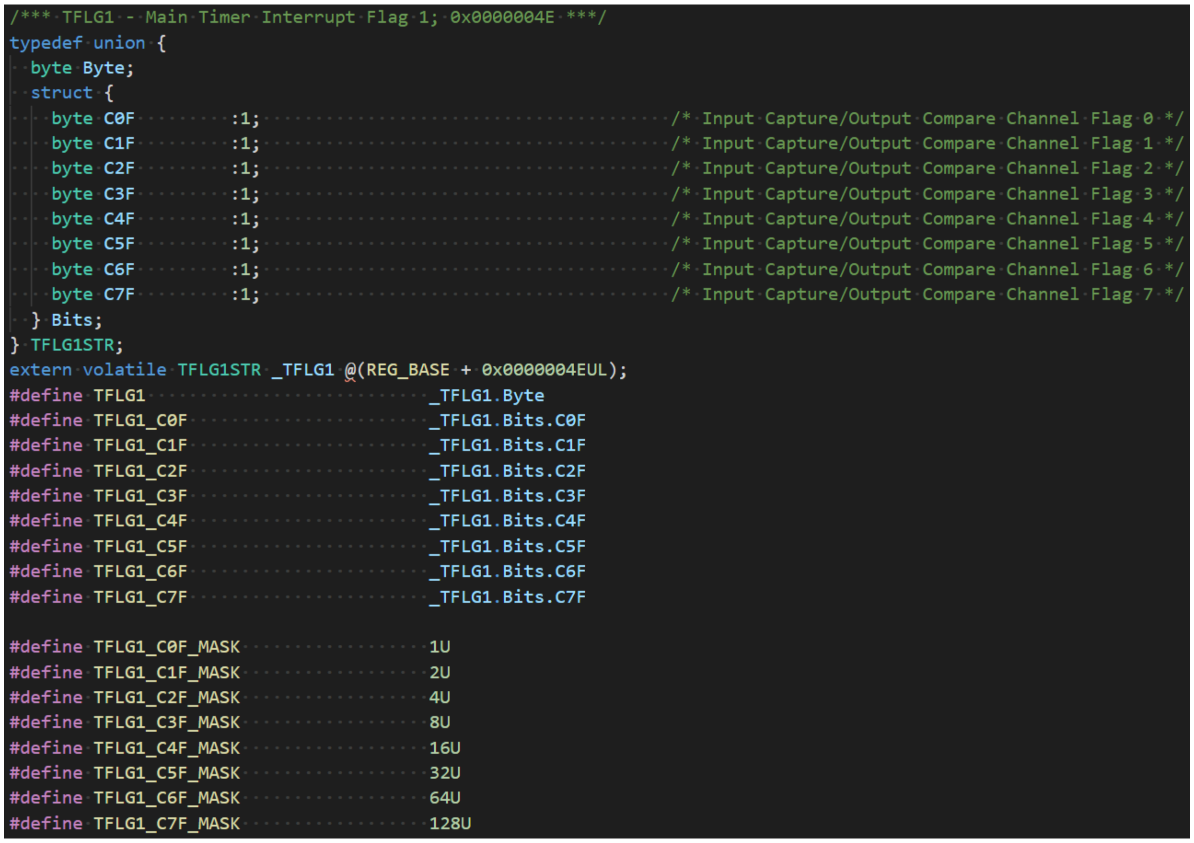
Figure 7-17. Main Timer Interrupt Flag 1 (TFLG1)



```
// wait for flag to pop
while (!(TFLG1_C6F))
  ;

// clear flag + rearm
TFLG1_C6F = 1;
TFLG1 |= TFLG1_C6F_MASK; // same thing
TFLG1 |= 0b01000000;     // same thing (careful to get right bit!)
```

There are masks defined for each bit, and unions of byte over single bit structures for the bits in each register. These shenanigans add up to an ultra-low overhead mechanism to read and alter individual bits, by name, in your code. Without autocomplete and popups though, this would not be so easy.



```
/*** TFLG1 - Main Timer Interrupt Flag 1; 0x0000004E ***/
typedef union {
  byte Byte;
  struct {
    byte C0F         :1;                                      /* Input Capture/Output Compare Channel Flag 0 */
    byte C1F         :1;                                      /* Input Capture/Output Compare Channel Flag 1 */
    byte C2F         :1;                                      /* Input Capture/Output Compare Channel Flag 2 */
    byte C3F         :1;                                      /* Input Capture/Output Compare Channel Flag 3 */
    byte C4F         :1;                                      /* Input Capture/Output Compare Channel Flag 4 */
    byte C5F         :1;                                      /* Input Capture/Output Compare Channel Flag 5 */
    byte C6F         :1;                                      /* Input Capture/Output Compare Channel Flag 6 */
    byte C7F         :1;                                      /* Input Capture/Output Compare Channel Flag 7 */
  } Bits;
} TFLG1STR;
extern volatile TFLG1STR _TFLG1 @(REG_BASE + 0x0000004EUL);
#define TFLG1                           _TFLG1.Byte
#define TFLG1_C0F                       _TFLG1.Bits.C0F
#define TFLG1_C1F                       _TFLG1.Bits.C1F
#define TFLG1_C2F                       _TFLG1.Bits.C2F
#define TFLG1_C3F                       _TFLG1.Bits.C3F
#define TFLG1_C4F                       _TFLG1.Bits.C4F
#define TFLG1_C5F                       _TFLG1.Bits.C5F
#define TFLG1_C6F                       _TFLG1.Bits.C6F
#define TFLG1_C7F                       _TFLG1.Bits.C7F

#define TFLG1_C0F_MASK                  1U
#define TFLG1_C1F_MASK                  2U
#define TFLG1_C2F_MASK                  4U
#define TFLG1_C3F_MASK                  8U
#define TFLG1_C4F_MASK                  16U
#define TFLG1_C5F_MASK                  32U
#define TFLG1_C6F_MASK                  64U
#define TFLG1_C7F_MASK                  128U
```

Again, future you will be very interested in this. Present you is probably confused. Not to worry, your instructor will demo these elements to you when the time is right.