Debugging Tool for Arduino Code, version 2.0

By R. G. Sparber

Publishing an idea for others to see can have extremely positive results. I wrote this article about a new way for me to debug Arduino code. To my delight, one of the readers, Dave Kellogg, pointed out a far better method. During compilation, the string

__LINE__

is converted to a number that equals the line number in the uncompiled code. By entering

Serial.println(__LINE__);

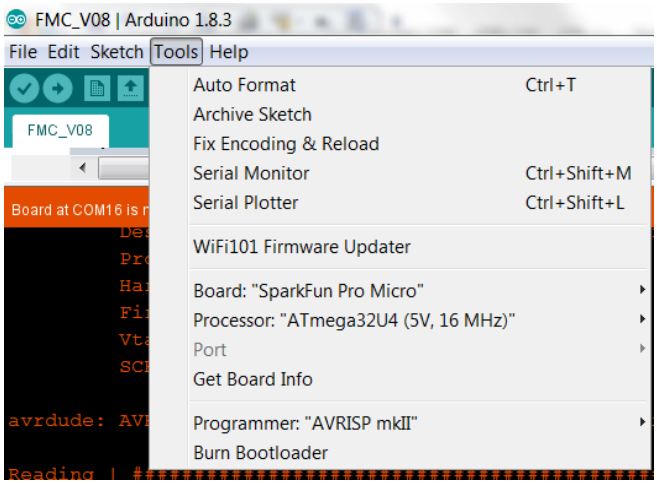
every place of interest, I can see if it got there. Since all line numbers are unique within a file, there is no ambiguity in finding the correct location. I now have a Key Text macro that spits out this line of code with a blank line above and below it.

If you want the ability to turn off all of these serial prints at the same time, conditional logic that operates during compilation can be used. Since I tend to turn these prints on and off individually, I prefer to just comment them out as needed. To turn them all off, I do a global substitution of "Serial" to "//Serial". Just be careful when turning one of them back on to also remove // from

Serial.begin(9600);

A word of warning. If you add enough print statements, you will run into program memory limitations which cause random program failures. In my case, all was fine, I added a few more print statements, and code that had worked fine for a long time started to fail. If in doubt, turn off all serial print statements and retest. Note that __LINE__ does not take up any dynamic memory.

My original method is not wrong, just not nearly as elegant. Others may still find some value in it so I have left it in:



A common technique for debugging Arduino code is to pepper it with serial print statements. You can then use the Arduino's Software Development Environment's serial monitor to spool up what is going on.

One problem with having a large number of

Serial.println("hello world");

statements is that the text string eats up space used for variables. A nice fix is to use

Serial.println(F("hello world"));

The F( ) subroutine keeps the text string in program store. But now there is even more to type. My solution was to call on my macro program, **Key Text**[2] which is an easy to use yet powerful program that lets me automate the debugging statement.

[2] See http://www.mjmsoft.com/

It isn't so important what I print but it must be unique so I can later search for it. With just a few keystrokes, I built a macro that generates a string of numbers unique for each minute. It is made up of the Day Month Year Hour Minutes using a 24 hour clock. After this series of numbers I add an index which increments from 1 to 26 and then back to 1. In this way I get a unique number as long as I do not trigger the macro more often than once every 3 seconds. This has not been a limitation since it takes me more than 3 seconds to think about where the next print should go.

I trigger this macro by holding down the Control and Alt keys while I press "=".

Say I want to add a serial print statement at line 901:

```
900     while(i<7){
901
902       MinorAlarm[i] = false;
```

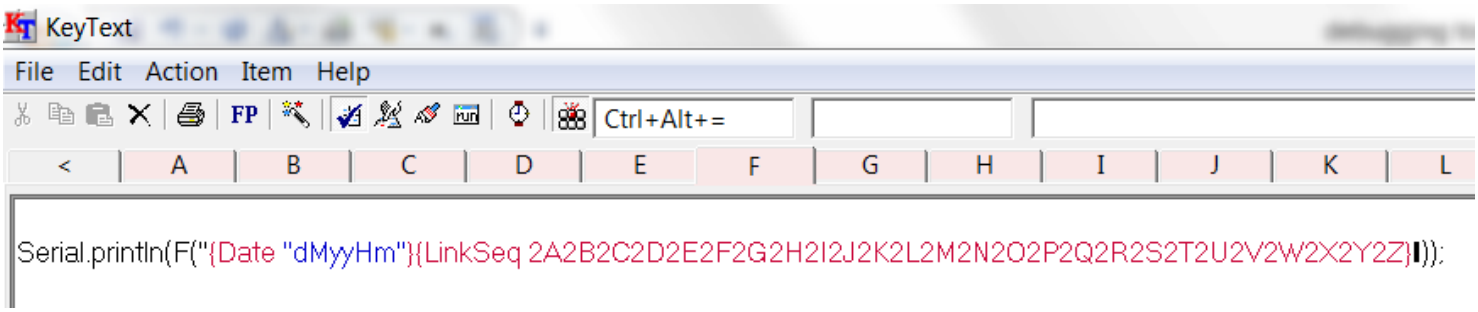
I first put the cursor on line 901. Then I press Control and Alt and "=" and see

```
900     while(i<7){
901
902 Serial.println(F("19121717125"));
903
904       MinorAlarm[i] = false;
```

What print out this time was 19121717125 which consists of the **19**th day of the **12**th month 20**17** at **17**:**12** followed by an index number of **5**. The next time I invoke this macro, the index number will be 6.

To make the code easier to see, I had the macro put a blank line above and below the print statement.

If you own Key Text, here is what the macro looks like:



The first line contains a carriage return. Then I wrote **Serial.println(** which will send the defined text out the Arduino's serial port. Next you see **F("** which is the start of the F( ) subroutine. After that is gets kind of cryptic but this was all generated by the Wizard and is the date and time. Next is a built in function called Link Sequence. On each call it picks up the contents of the next macro location. The first location is 2A and the last one is 2Z. Location 2A contains "1" and location 2Z contains "26". I end with that black rectangle followed by **));** and a second carriage return. The black rectangle is my second quote. I initially just typed the quote but it didn't print so I used a built in function that inserted special characters. With this tool I got my **"));** sequence.

At the top right you see the shortcut: Ctrl+Alt+=.

The serial monitor spills out a stream of these numbers. I copy them into a text file so they can be searched. If desired, I can add text using the find and replace function.

Assuming I placed my serial prints in useful places, I take each number, go to the code and search for that number, and know its location.

When I'm done, I can use the global search for "Serial.println" and replace it with "// Serial.println". That will logically remove it from the file without physically removing it. At a later date I might want to turn some of them back on.

If you wish to be contacted each time I publish an article, email me with just "Article Alias" in the subject line.

Rick Sparber
Rgsparber.ha@gmail.com
Rick.Sparber.org