A Debugging Tool for Arduino Code, version 3.0

By R. G. Sparber

Protected by Creative Commons.[1]

Publishing an idea for others to see can have extremely positive results. I wrote version 1.0 of 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. It has since evolved to the following.

During compilation, the string

```
__LINE__
```

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

```
Serial.println(__LINE__);
```

at 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.

Although the line number is unique, I have found that knowing the subroutine containing the code speeds up debugging.

```
__FUNCTION__
```

is converted by the compiler to the current subroutine's name.

[1] This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Especially when debugging code that is running on separate Arduinos in parallel, it is essential to know the relative time of events. It is easy to hold down the reset line on both processors and let go at the same time. As part of setup(), I initialize the variable StartForTimeStampULong to 0 on each device. This lets me time stamp each diagnostic print. And finally, I use #define, #ifdef, and #endif to turn a group of prints on and off. When not used, the code is not compiled. This minimizes memory usage and the impact on real-time.

At the top of the file are a series of #defines that I can comment out as needed. For example:

```
//#define DiagPrint1 1
//#define DiagPrint2 1
//#define DiagPrint3 1
//#define DiagPrint4 1
//#define DiagPrint5 1
//#define DiagPrint6 1
//#define DiagPrint7 1
//#define DiagPrint8 1
//#define DiagPrint9 1
//#define DiagPrint12 1
```

Each line controls a family of diagnostic print statements.

Then, sprinkled around the code is one of two blocks of code that I keep in macros. It just takes a few keystrokes to insert one of them.

When I just want to know I've been at a place in the code, I use:

```
#ifdef DiagPrint1
Serial.print(__FUNCTION__);
Serial.print(F("(): "));
Serial.print(__LINE__);
Serial.print(F(".    TS:  "));
Serial.println(millis() - StartForTimeStampULong);
#endif
```

where `DiagPrint1` is replaced by the print family's name.

It generates something like:

```
Foobar(): 123.   TS:456
```

which tells me I am in the Foobar subroutine, line 123 with a time stamp of 456 milliseconds since the processor started up.

If I want to see one or more variables, I use:

```
#ifdef DiagPrint1
Serial.print(__FUNCTION__);
Serial.print(F("(): "));
Serial.print(__LINE__);
Serial.print(F(".    TS:  "));
Serial.println(millis() - StartForTimeStampULong);
Serial.print(F(" tag "));
Serial.println( data );
#endif
```

"`tag`" and "`data`" are replaced with the variable's name. I replicate these two lines as needed in the block of code.

A sample output with tag replaced with "`qaz =`" and data replaced with the variable name "`qaz`":

```
Foobar(): 123.   TS:456
qaz = 789
```

I welcome your comments and questions.

If you wish to be contacted each time I publish an article, email me with just "Subscribe" in the subject line. If you are on this list and have had enough, email me "Unsubscribe" in the subject line.

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