# Serial Port Utilities

Debugging Arduino applications most often means sending some text out through the serial port over a USB link to your computer, and then looking at it in the Arduino Serial Monitor app.

The Serial.print and Serial.println functions are OK when you first start but are a bit limited in that they offer no real formatting.

It’s relatively easy to provide your own formatting functions that call Serial.print so that you can have printf-like formatting. Note the the GCC libraries with the Arduino IDE do not support formatting of floats. So you can’t use the %f formatting instruction in sprintf (for example).

The Arduino code does include the String class which can format a float so we can use that with a little varargs magic to make some functions and macros that make debugging easier.

In my own code I sometimes want print statements through the serial port only to be turned on while I’m debugging the code so that the final version saves the memory required for all those debug strings. You can, of course, put your print statements inside #ifdef wrappers like this:

#ifdef DEBUG

Serial.println("Yes, it’s working.");

#endif

That’s nice but gets a bit tedious if we want to do this a lot. So in addition to a couple of formatting functions I also have a macro that turns on my debug statements in development builds and turns then off when I’ve finished debugging. It’s a little tricky to avoid the compiler complaining about unused variables when it finds the strings we’ve decided we don’t need in our final builds. We fix that by adding some GCC pragma statements. You can find a good discussion about suppressing GCC warnings here: <http://dbp-consulting.com/tutorials/SuppressingGCCWarnings.html>

What this all means is that you can write code like:

dbg("Counter: %d", counter\_value);

which only sends the message to the Serial Monitor when you have DEBUG defined in your code. If you want the message to always be sent out, I provided another macro to do that:

sout("Counter: %d", counter\_value);

The real reason I have the sout macro is that my typing is lousy and typing serial\_printf all the time gets tedious.

There are lots of ways to achieve what I've just described and there are certainly libraries that you can download that provide comprehensive debug support and more. The header and .ino (C++) file here just show one way of doing things. You can easily rename the macros, or add to the functionality as you need to. There are some limitations to the implementation. Read the code for explanations.

The example application shows how to use the print and debug functions. To use the same functionality in your own code you can just copy the serial\_utils.h and serial\_utils.ino files into your project or put them in your own library (changing the .ino extension to .cpp).

Happy coding.