# A Diagnostic Tool for Arduino Software, version 1.1

**By R. G. Sparber**

While developing software for the Arduino[2] it is inevitable that bugs will be introduced. In order to isolate and understand each of these bugs, it is essential to see what is going on.

A well equipped software development environment permits the user to be notified of program status as the code executes. The output is commonly displayed on a computer screen. But what if your environment is not so great or the system has already been deployed and no computer is attached?

A very old yet still useful technique is to seed the code with a diagnostic subroutine. It will report on status when turned on and be invisible when turned off. Reporting is done with whatever human/machine interfaces exist. In my case, I have a Liquid Crystal Display (LCD) and a pushbutton.

At any time, I can turn on all copies of my diagnostic subroutine by holding down the pushbutton and power cycling. Turning them all off requires power cycling without holding down the button. No special tools needed.

[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.
[2] For an overview of this hardware and software, see https://en.wikipedia.org/wiki/Arduino

The diagnostic subroutine prints a small amount of information to the LCD.

Here is an example of how I use the tool.

Say I wanted to know when I entere a given subroutine. I add this to the top of that subroutine

```
Diag(314,0,Pause);
```

When this tool is reached during normal program execution, it would first look at the DiagOn flag. If set to true, it would display on the LCD:

```
ID 314
Data 0
```

Since the control flag is "Pause", it will delay 10 seconds and then return control to the program.

What if I wanted time to check other parts of the system? I can use

```
Diag(314,0,Wait);
```

With the control flag set to "Wait", the tool first prints to the LCD. Then it starts watching the Pest pushbutton. When it see it had been pushed, the tool clears the display and waits for the button to be released. Then it returns control to the main line code.

I wanted to be sure the button was released before returning control in case the main line code was waiting for this button to be pushed.

My data does not have to be a fixed number, I can have this

```
MyShoeSize = 8;
      …
Diag(7, MyShoeSize,Wait);
```

which will display

```
ID 7
Data 8
```

This information would stay on the display until I pushed the Pest button. Program execution would resume when I released it.

In the startup() section of the code I put:

```
//Enable Diag() if Pest button was held down at power up. Default is Diag() off
if (DebouncedRead(PABPin) == Pushed)
   {
      DiagOn = true;
      lcd.clear();
      lcd.print (" Diagnostics");
      lcd.setCursor(0,1); //print next line on second row
      lcd.print ("  enabled.");
   }
WaitUntilPestButtonReleased:
if (DebouncedRead(PABPin) == Pushed) goto WaitUntilPestButtonReleased;
```

My pushbutton is called "Pest". I read it's state using a subroutine called DebouncedRead(). The constant "Pushed" equals the value returned by my read subroutine when the button is pushed (no surprise there…).

If Pest is held down while this code executes, I turn on a flag called DiagOn, display an acknowledgement message on the LCD, and wait until the button is released.

I hope you notice that I choose names that help me remember what is going on. This costs me no Arduino memory since the compiler strips all of this text off.

At the top of the code I define "Wait", "Pause", and the DiagOn flag:

```
const int Pause = 1; //used by Diag()
const int Wait = 2; //used by Diag()
boolean DiagOn = false;
```

The user might accidently specify a control value other than Wait or Pause so I do defend against this in the tool.

My diagnostic tool is called Diag() and is defined at the bottom of my file. It first reads the DiagOn flag and just returns if set to false. If true, the tool displays a user defined marker, a user defined bit of data, under user defined control. This control can be a 10 second pause or a wait until the Pest button is pushed and released.

Since each instance of this tool can be customized, I can use as many as desired with no chance of confusion.

Here is the diagnostic tool code.

When called, I pass the subroutine an integer that can uniquely define this instance of the tool: "marker". I also pass it whatever integer data is needed: "data". And finally, I pass it an integer that tells the tool to either wait 10 seconds or to respond to the pushing of the Pest button: "control". If the user does not pass a valid control value, the tool notifies them.

```
void Diag(int marker, int data, int control) //diagnostic tool; control can be Pause or Wait. Pause is for 10 seconds. Wait is until Pest pushed and released
{
if (DiagOn == false) return; //default is all instances of this diagnostic tool off. Turn them on by holding down only the Pest button at power up.
/*
Inputs:
marker - identify of this instance of Diag()
data - an integer to be displayed
control - what to do when data is displayed
   * Pause - delay 10 seconds and then return
   * Wait - wait until the Pest button is pushed and released. Then return.
*/
lcd.clear();
lcd.print ("ID ");
lcd.print(marker);
lcd.setCursor(0,1); //print next line on second row
lcd.print ("Data ");
lcd.print (data);

if (control == Pause)
   {
      delay(10000); //wait 10 seconds
```

```
        lcd.clear(); //clean up display
        return;
    }

if (control == Wait)
    {
        ScanPestButton:
        if(DebouncedRead(CABPin) == Pushed)
            {
            //clean up display; next wait for Pest to be pushed and then wait for it to be
released
            lcd.clear(); //when Pest is pushed, clear the display
            if (DebouncedRead(CABPin) != Pushed)return; //when Pest is then
released, return
            }
        goto ScanPestButton; //keep scanning for Pest to be pushed and released
    }
//control is not Pause or Wait so tell user they didn't specify a valid Control type,
wait 10 seconds, clean up, and return
lcd.clear();
lcd.print ("ID ");
lcd.print(marker);
lcd.setCursor(0,1); //print next line on second row
lcd.print ("Control?");
delay(10000); //wait 10 seconds
lcd.clear();
return;
} //end of Diag()
```

I welcome your comments and questions.

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