

Arduino 'How-To' Series

ByVac BV4618 LCD Implementation Guide



Written by: *Sopwith*
Revision 1.0
September 28, 2014
sopwith@ismellsmoke.net

Introduction

This is another 'How-To' document in the continuing series of How-To's from the *Sopwith* library. Today we are going to explore the *ByVac 4618* LCD. *ByVac* is a company based in the UK that creates microcontrollers. On their [website](#), *ByVac* describes themselves as, "... a small manufacturers (sic) that aims to make electronics a bit easier; microcontrollers have a built in language and displays are either I2C or serial."

The *BV4618* LCD controller is a clever device that accepts VT100 ASCII commands to control a [Hitachi HD4470 LCD](#). Before the PC explosion in the 1980's, VT100 terminals were used to communicate with mainframes and micro-computers from IBM, Sperry, Rand, DEC, HP, and others. *Sopwith* remembers them well. Seems like just yesterday... The device has three interfaces: I2C, serial TTL and RS-232. The I2C and serial TTL interfaces make the device perfect for hacking on the Arduino or Raspberry PI.

The real strength of this device is its simplicity. It is easy to wire up and easy to manage the display. All the complexities of dealing with an LCD controller are handled by the *ByVac* controller. All you need to do is power it up, connect your interface of choice, and send ASCII commands to it.

Datasheets

If you plan on hacking any electronic device you should at least spend the time to understand its technical details. That is why *Sopwith* always starts a new project reading datasheets. *ByVac* has a very nice datasheet for the *BV4618* [here](#). The Hitachi *HD4470* LCD controller datasheet can be found [here](#). I suggest you closely study the *BV4618* datasheet so you understand how the device works and appreciate its cleverness.

You should also spend a few minutes reviewing the Hitachi datasheet for no other reason than get an appreciation for the amount of effort it takes to control a modern LCD device.

Step-by-Step

1) Wiring

To get started, let's wire up the *BV4618* device to your Arduino. In this illustration we will be using an Arduino UNO, although any of the devices in the Arduino family should work just fine. Figure 1 below shows how to connect the LCD to the Arduino UNO.

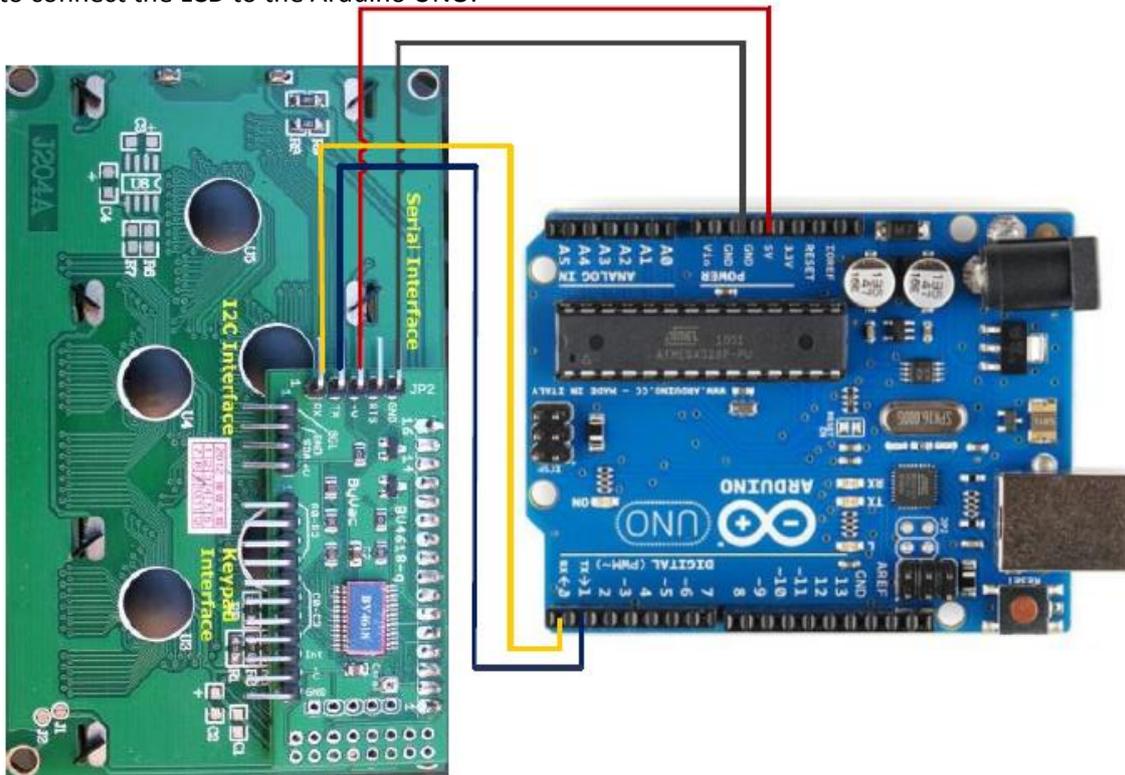


Figure 1- Serial TTL wiring

"If it works out of the box – what fun is that?"

As shown in the illustration connect the 5V+ and Gnd pins on the Arduino to the +V and GND pins on the LCD. Arduino Pin-0 (Rx) connects to the LCD Rx pin and Arduino Pin-1 (Tx) connects to the LCD Tx pin. Experienced serial interface gurus might be thinking the Rx and Tx connections should be crossed, but this is not the case in this instance. Use 'straight-through' connections or the LCD will not work.

Power up your Arduino and you should see the LCD light up. If you wired the device correctly the LCD should show "Press CR_" as shown on the cover page. The *BV4618* has a feature that allows it to automatically detect the serial connection baud rate. It accomplishes this by "listening" for a <CR> (ASCII 13) character when first powered up. This means you must send a <CR> command before you can begin writing stuff on the display.

2) Arduino IDE

If you are an experienced Arduino coder you can skip this section and go on to Section 5). If you are new to the Arduino, here we walk through the process of getting the Arduino IDE up and running. First thing, head out to the Arduino website and [download](#) the latest IDE.



Figure 2 – Arduino IDE 1.0.6 download

In this case, we will download IDE version 1.0.6. Once downloaded, install the IDE. The following screen shots show the installation process.

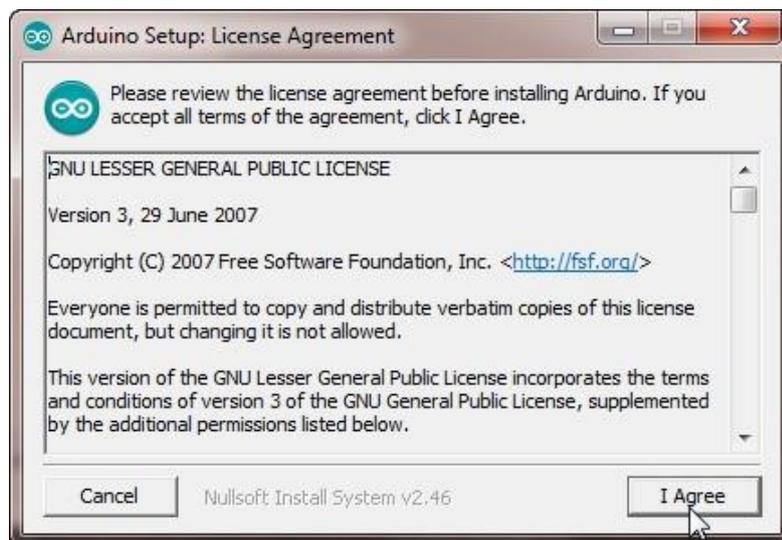


Figure 3 – Arduino IDE EULA

"If it works out of the box – what fun is that?"

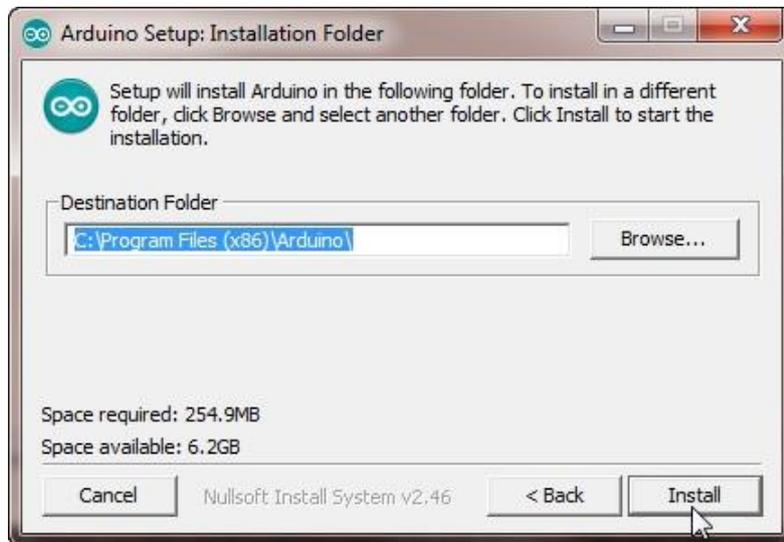


Figure 4 – Arduino IDE destination folder

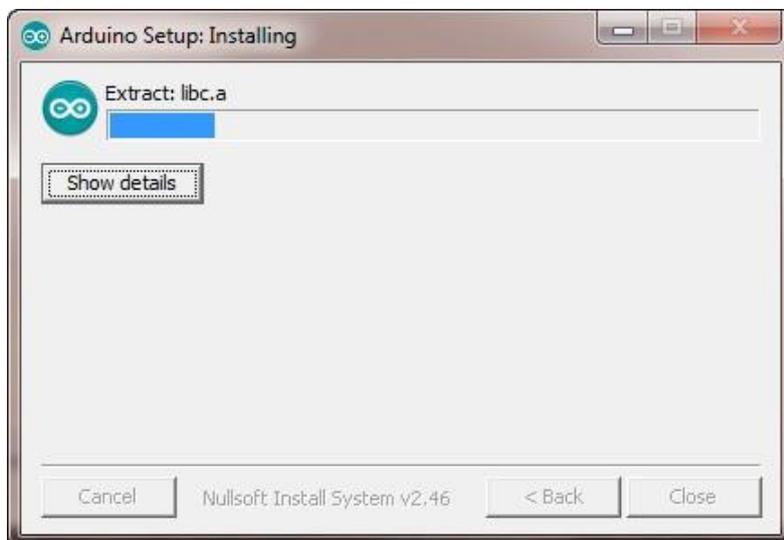


Figure 5 – Arduino IDE file copy



Figure 6 – Arduino IDE device driver install

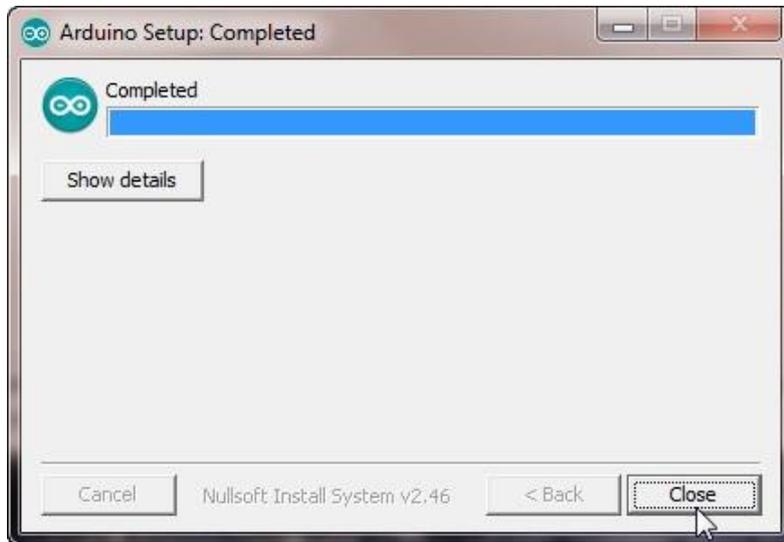


Figure 7 – Arduino IDE destination folder

At this point, you should have the Arduino IDE installed. Next, let's get the IDE to talk to the Arduino UNO.

3) Arduino Connection

The IDE installer is pretty capable. In my case, my UNO was identified and a new virtual serial port (Port-9) was created to connect to it. Your task is to ensure that you have the correct Arduino board selected and that the correct COM port has been assigned to it (Figure 8-9).

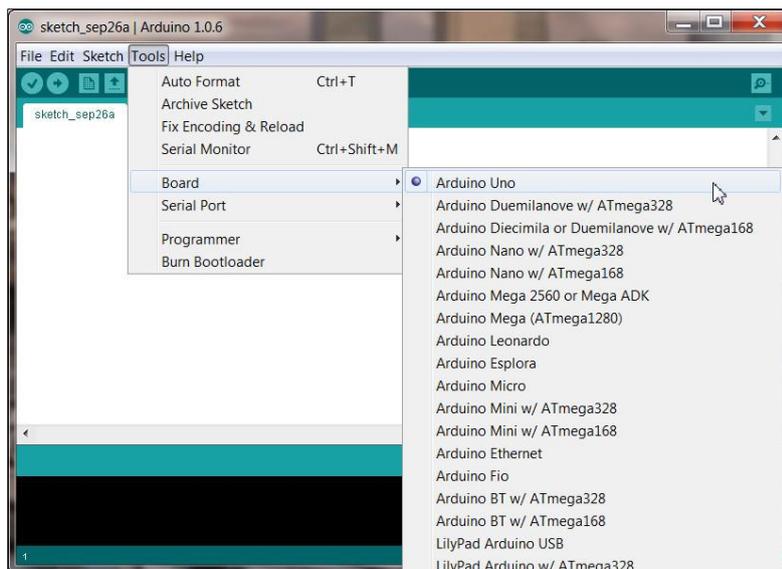


Figure 8 – Arduino IDE board selection

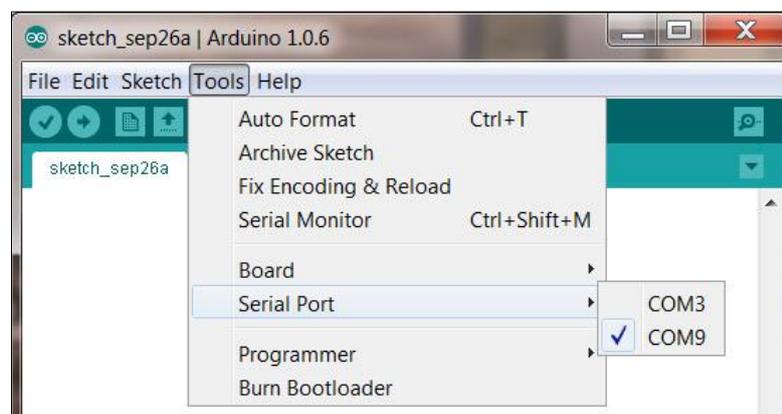


Figure 9 – Arduino IDE serial port selector

“If it works out of the box – what fun is that?”

To ensure your IDE is working correctly, let's load the 'Hello World' program for an Arduino – *Blink*. Click on *File | Examples | 01.Basics | Blink* to load *Blink* into the code editor (Figure 10).

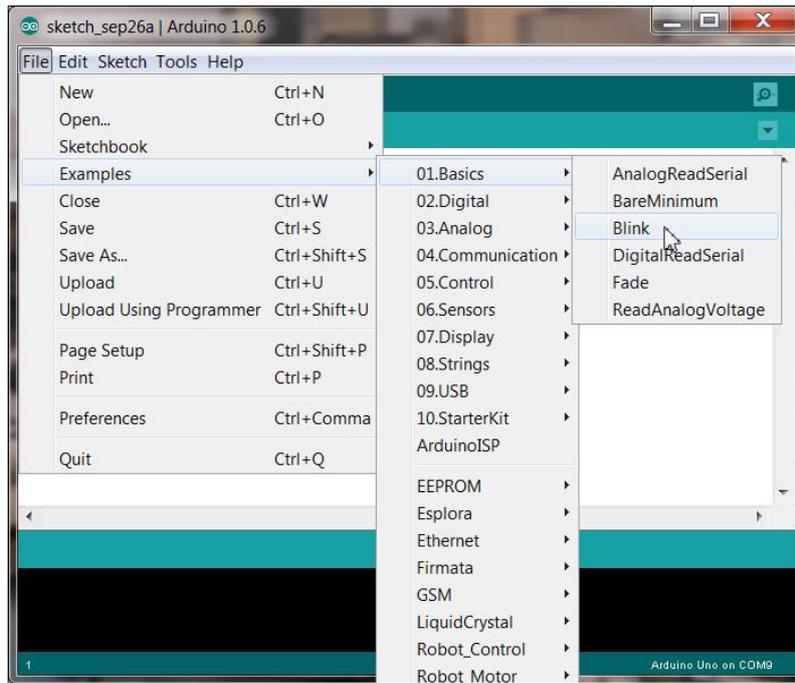


Figure 10 – Arduino IDE *Blink* example

Blink is now loaded in the IDE code editor. You can see in Figure 11 a basic Arduino program has two functions: `setup()` and `loop()`. The `setup()` function is used to initialize program variables and prepare your program to run. The `loop()` program is where your code will run. Whatever is placed in this function will run in a loop over and over until the Arduino is powered off.

As you can see from the *Blink* code, the `setup()` function simply selects Pin-13 (yellow LED) as a digital output. The `loop()` function toggles the LED pin ON and OFF with a delay of 1 second (1000 ms) between states.

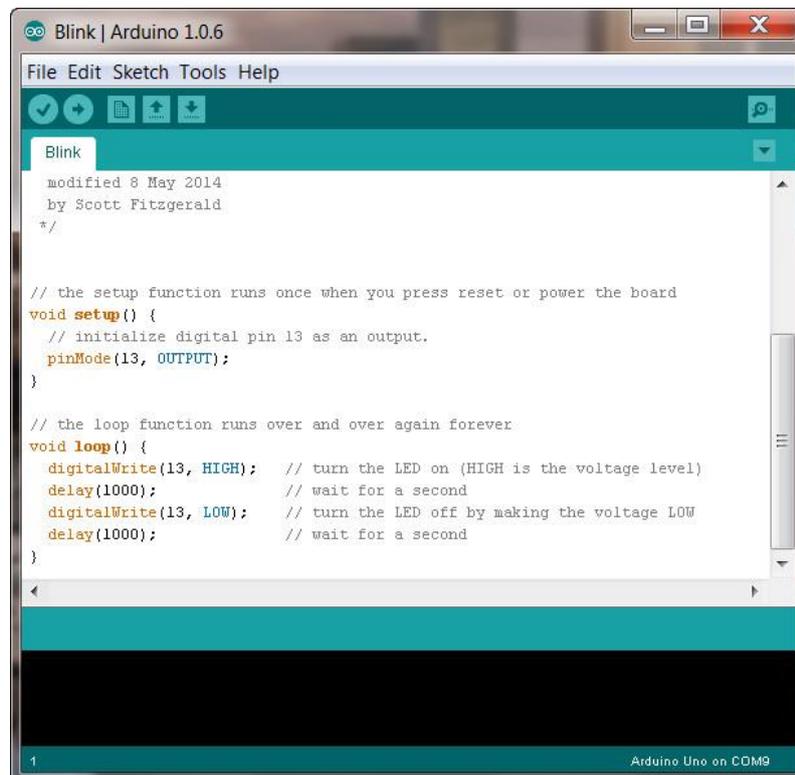


Figure 11 – *Blink* code

“If it works out of the box – what fun is that?”

To compile *Blink* and load it on your Arduino, press the Right arrow button at the top left corner of the IDE (Figure 12).

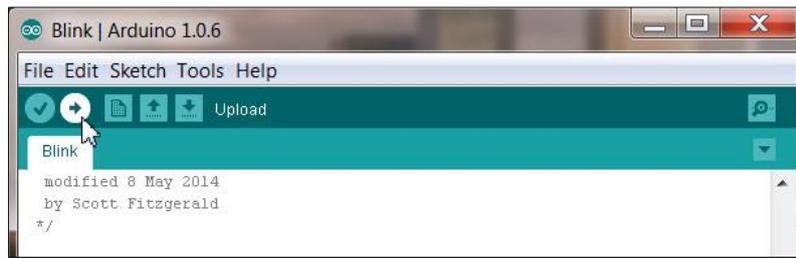


Figure 12 – *Blink* compile and upload

If your IDE is configured correctly you should see the IDE compile the *Blink* program and load it on your Arduino. As you can see in Figure 13, if the code was sent to the Arduino you will see 'Done uploading' above the compiler output window.

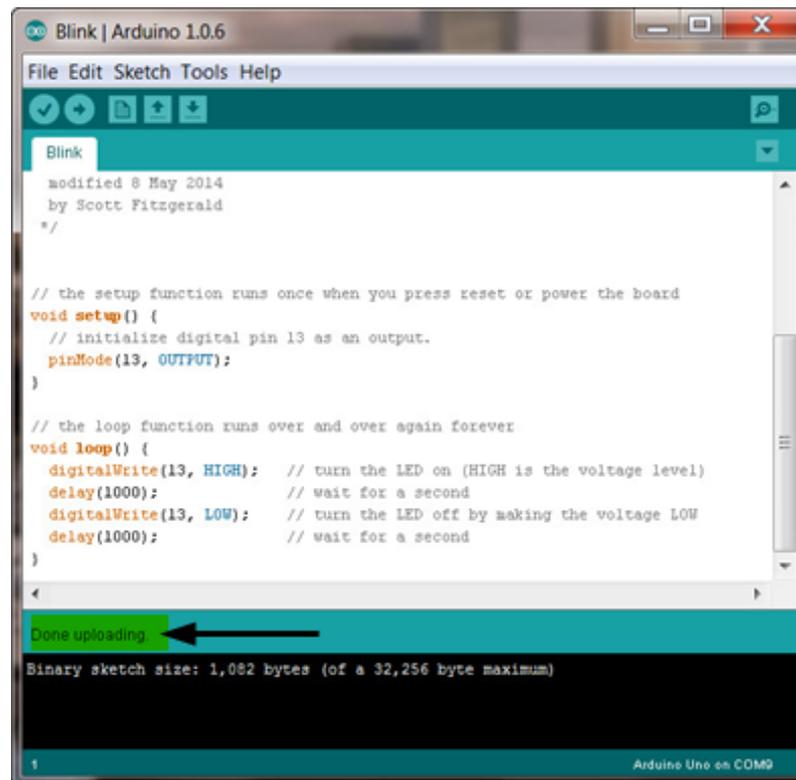


Figure 13 – *Blink* upload complete

Even more interesting, the bright yellow LED on your Arduino should be blinking on and off every second. Make a simple change to the program by changing the delay function parameter from 1000 to 500 (1 second to ½ a second). Upload the program to the device. You should now see the yellow LED blink twice as fast as before.

Now that your Arduino development environment is set up, let's spend a few moments and talk about Libraries.

4) Arduino Libraries

The Arduino IDE is truly a marvel. Hidden behind the scenes is a very complex C/C++ programming infrastructure. In order to introduce non-programmers to the embedded world of electronics the IDE has to be easy to use. It is important for new Arduino developers to understand the Arduino IDE concept of a library.

In the Arduino world a program is called a *Sketch*. All sketches are always saved in a folder called a *Sketchbook*. When you write a program for the Arduino the IDE will force you to save it in a sketchbook folder. In Windows 7 the IDE sets up a top-level folder named 'Arduino' in your user profile *My Documents* folder (Figure 14).

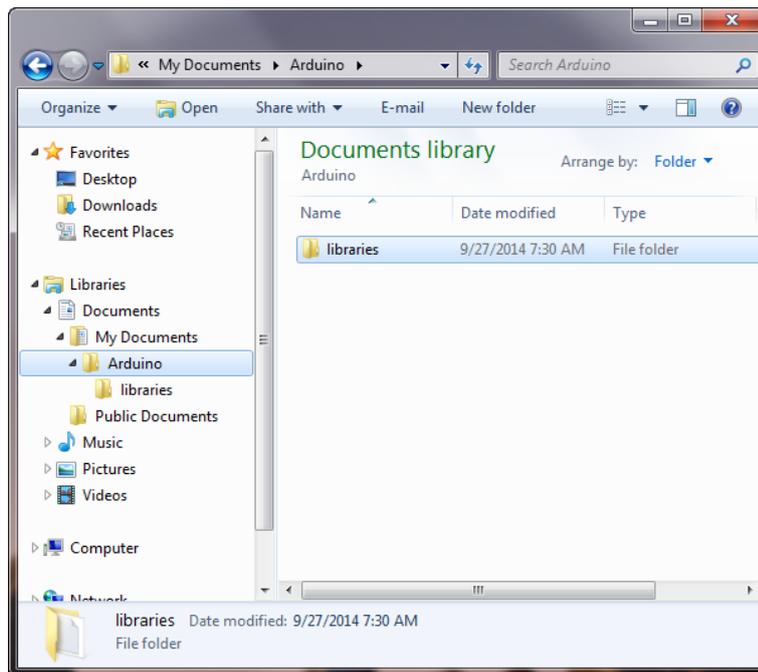


Figure 14 – Arduino folder location

As you can see, a new IDE install Arduino folder contains a single folder called libraries. This folder is used to store Arduino libraries written or installed by programmers. Libraries are programs that are written by programmers for other programmers. They are used to make programming tasks easier. This folder is empty until you install additional libraries. We will talk more about libraries in the next section.

I created a simple skeleton sketch and saved it with the name *sketch_empty* (Figure 15).

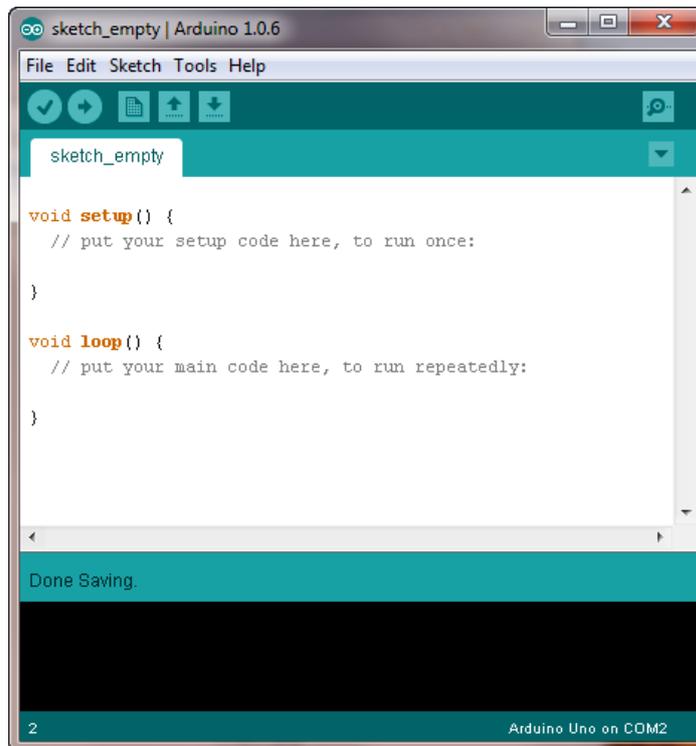


Figure 15 – Saved sketch 'sketch_empty'

“If it works out of the box – what fun is that?”

As you can see in Figure 16, the IDE saved my sketch in the Arduino folder within a folder with the same name as my sketch – *sketch_empty*. (Figure 16)

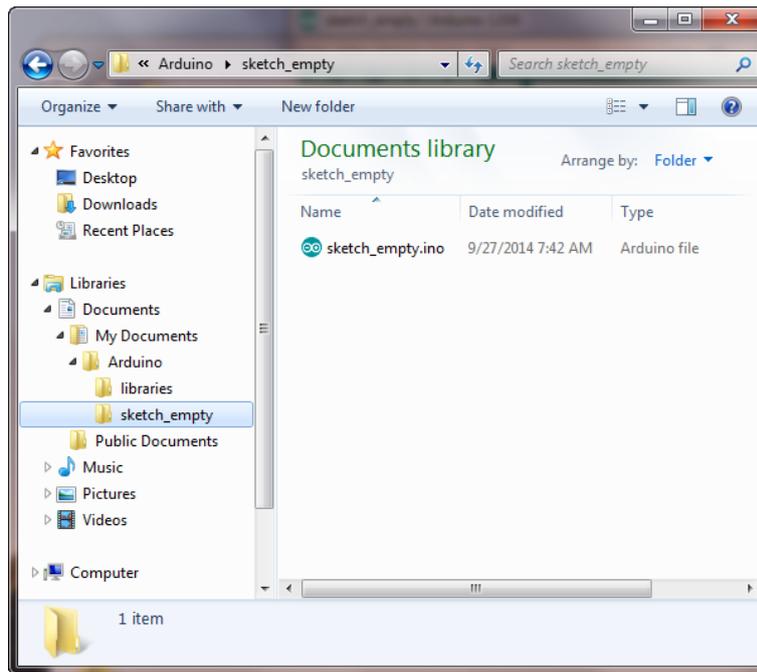


Figure 16 – Saved sketch folder ‘sketch_empty’

If you are a new programmer in the Arduino world it is important to understand how the IDE works. In summary, an Arduino program is called a *Sketch*. In Windows 7, Sketches are stored in the *Arduino* folder in your user profile *My Documents* folder. There is also a *libraries* folder in the *Arduino* folder that is used to store code libraries.

If you are having trouble compiling programs it is usually caused by a misunderstanding of the above concepts. If the compiler complains about missing header files or variables you can almost always bet it is due to missing libraries or libraries that are placed in locations the compiler does not look for them.

5) BV4618 LCD Hacking

Time to cut some code for the *BV4618* LCD! In this section the goal is to write a ‘Hello World!’ program and write a string on the LCD. Before we do we must install a code library for the *BV4618* controller.

The required library is [BV4618 Arduino library](#). Follow the link and download the library zip file. Once downloaded, unzip the zip file. Figure 17 shows the unzipped library in my *Downloads* folder.

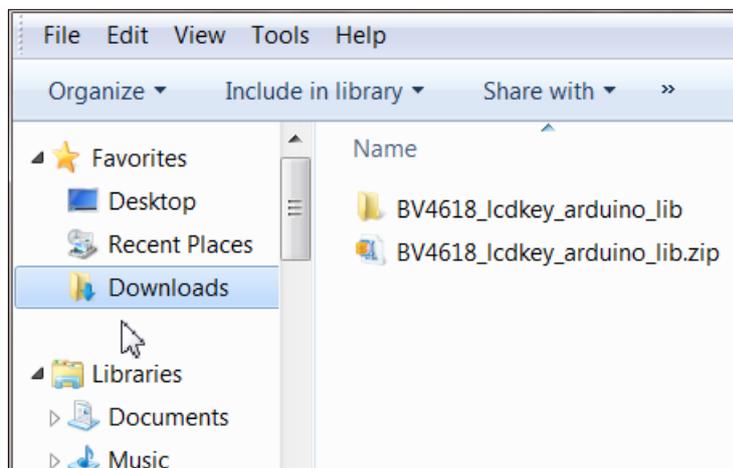


Figure 17 – BV4618 required library

“If it works out of the box – what fun is that?”

The *BV4618_lcd_arduino_lib* folder actually contains several library folders as shown in Figure 18.

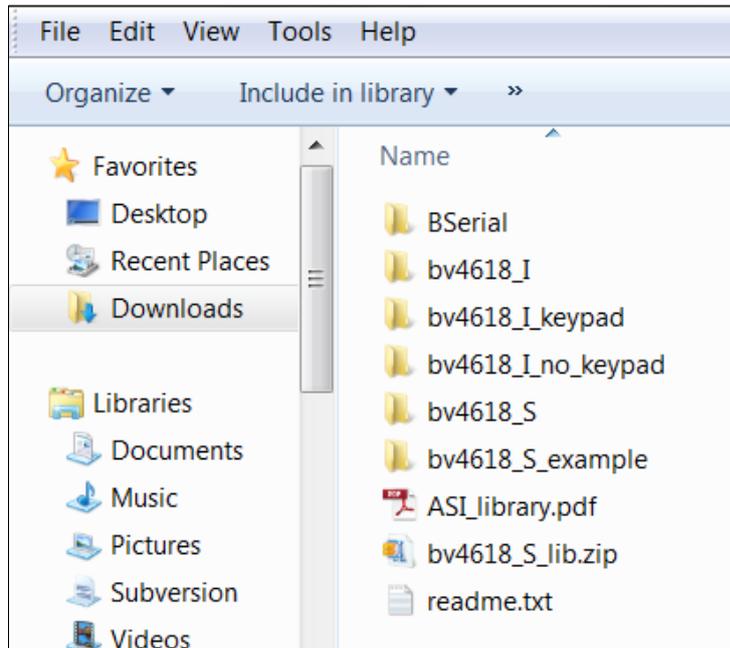


Figure 18 – BV4618 libraries

There are libraries here for the serial, I2C, and numeric keypad interfaces, plus the dependent *BSerial* library. We are only interested in the serial libraries *bv4618_S* and *BSerial*.

Installing these libraries is simple. Open a skeleton sketch from the IDE *File / Example / Basics / Bare Minimum* menu and save it with the name *Hello_BV4618* (Figure 19).

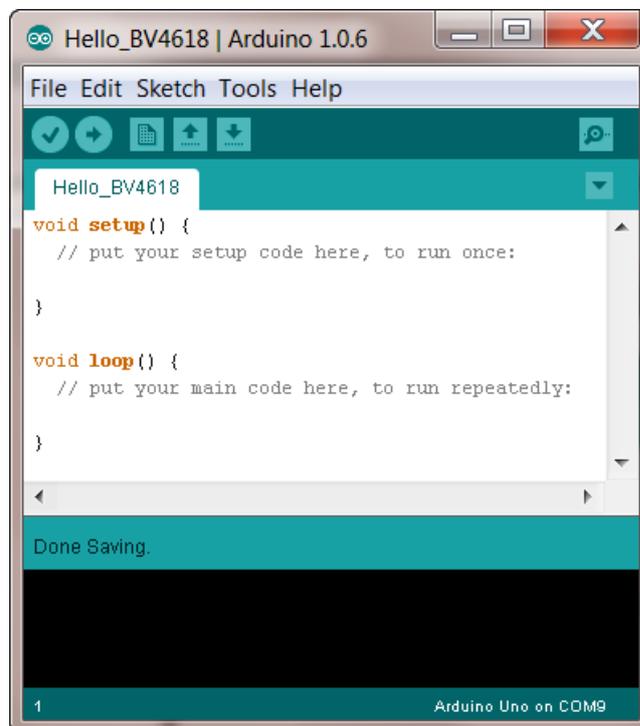


Figure 19 – Hello_BV4618 sketch

“If it works out of the box – what fun is that?”

Next, let's add the two required libraries. Click on *Sketch / Import Library... / Add Library...* (Figure 20).

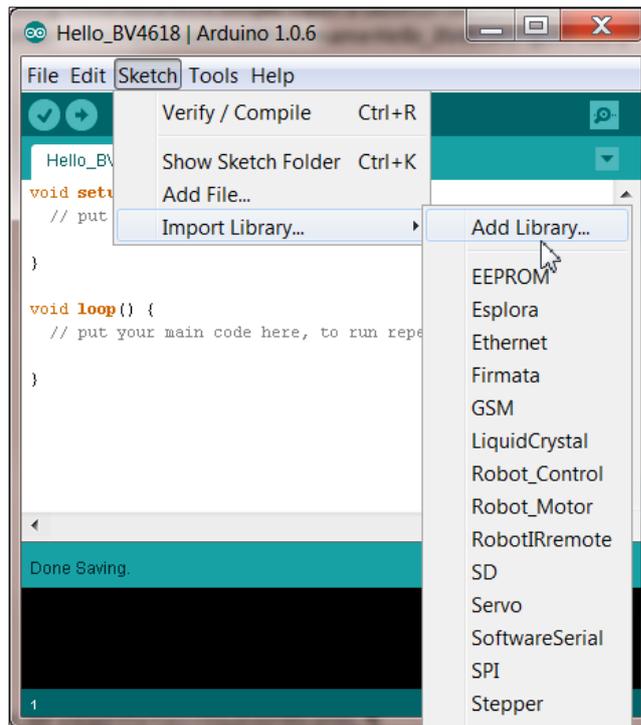


Figure 20 – Add required libraries

Traverse to your downloads folder and click on the *BV4618_lcdkey_arduino_lib* folder. This will open the folder. Select the *BV4618_S* folder and click on *Open*. This will load the library into the Arduino libraries folder making it visible to your program (Figure 21).

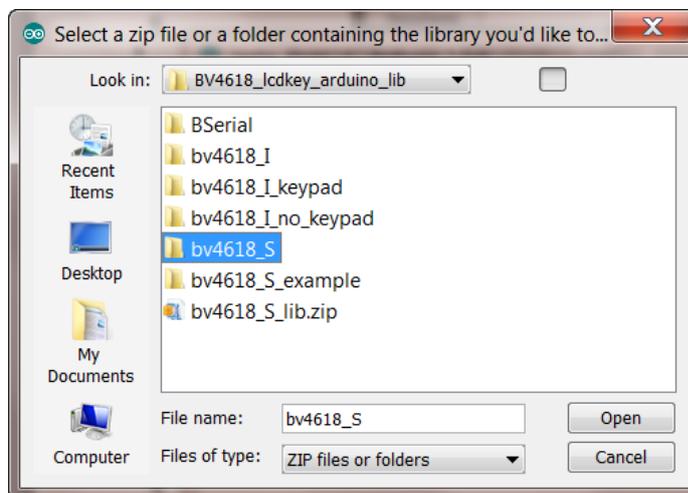


Figure 21 – Add BV4618_S library

Repeat the same steps and load the *BSerial* library by selecting the *BSerial* folder. If the libraries loaded correctly, you will see them in the IDE *Import Library* menu *Contributed* section (Figure 22).

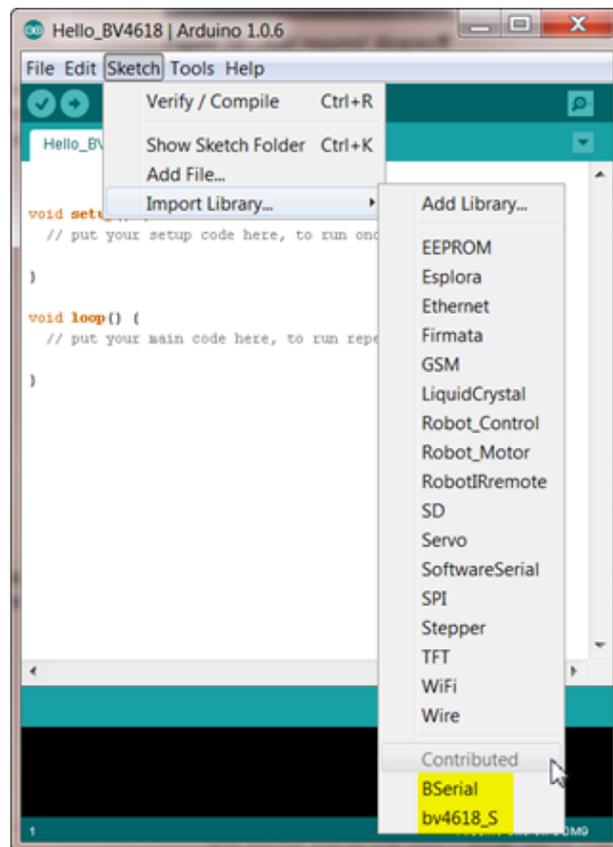


Figure 22 – Contributed libraries

The next step is to import the two libraries into our *Hello* sketch. To do this, use the *Import Library* menu and click on the *BSerial* and *BV4618_S* libraries at the bottom of the menu. Obviously, you will have to do this twice since you can only import one library at a time.

If the libraries were imported to your sketch properly you should see three new *#include* lines at the top of your sketch (Figure23). The IDE is smart and adds any required *#include* files.

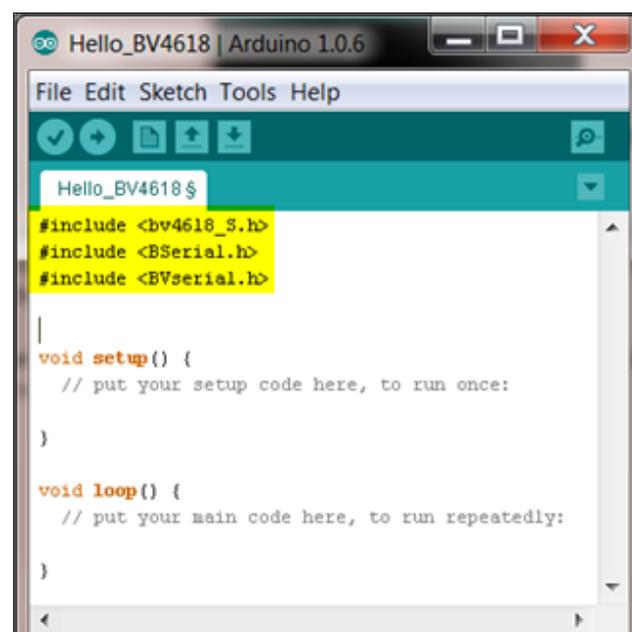
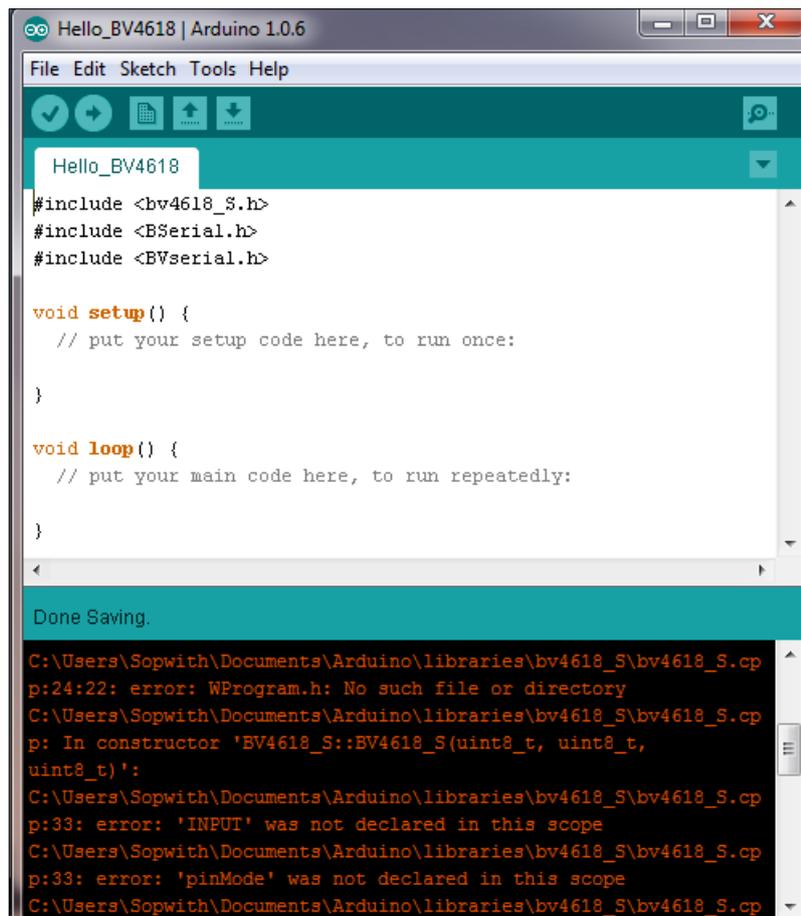


Figure 23 – Library include files

“If it works out of the box – what fun is that?”

If you try to compile this simple sketch you will discover it does not compile. There are lots of errors listed in the compiler output window. Oh the joy! Everybody knows that Sopwith gets grumpy when something works right out of the box. “You can’t have any fun when you don’t have anything to fix!”

The image shows a screenshot of the Arduino IDE interface. The title bar reads "Hello_BV4618 | Arduino 1.0.6". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for saving, running, and other functions. The main text area contains the following code:

```
Hello_BV4618
#include <bv4618_S.h>
#include <BSerial.h>
#include <BVserial.h>

void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Below the code area, a status bar indicates "Done Saving.". The bottom panel shows the compiler output with several error messages:

```
C:\Users\Sopwith\Documents\Arduino\libraries\bv4618_S\bv4618_S.cp
p:24:22: error: WProgram.h: No such file or directory
C:\Users\Sopwith\Documents\Arduino\libraries\bv4618_S\bv4618_S.cp
p: In constructor 'BV4618_S::BV4618_S(uint8_t, uint8_t,
uint8_t)':
C:\Users\Sopwith\Documents\Arduino\libraries\bv4618_S\bv4618_S.cp
p:33: error: 'INPUT' was not declared in this scope
C:\Users\Sopwith\Documents\Arduino\libraries\bv4618_S\bv4618_S.cp
p:33: error: 'pinMode' was not declared in this scope
C:\Users\Sopwith\Documents\Arduino\libraries\bv4618_S\bv4618_S.cp
```

Figure 24 – Hello_BV4618 compile errors

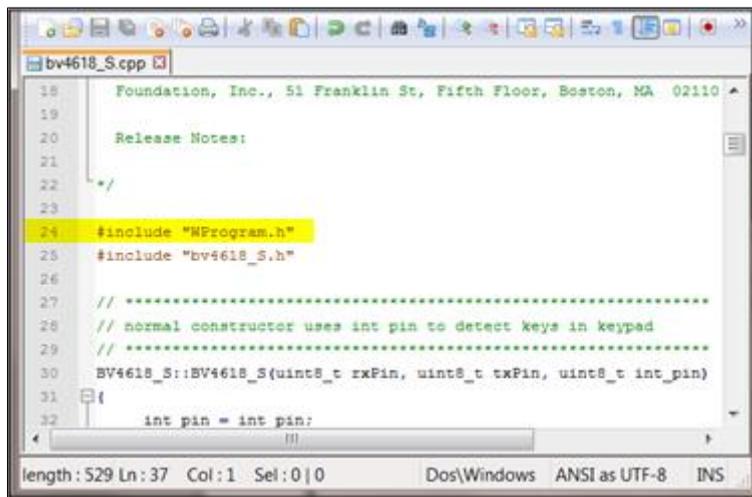
Notice the first error message in the output console: “WProgram.h: No such file or directory.” The source file that triggered this error is listed as `bv4618_s.cpp`. The error messages that follow describe missing variables.

To fix this error, let’s have a look at file:

`C:\Users\Sopwith\Documents\Arduino\libraries\bv4618_S\bv4618_S.cpp`. Obviously, this file is part of the BV4618_S library we installed earlier. To edit library code I do not use the Arduino IDE. This is because the IDE does not want you messing with libraries and flags all library code as ‘read-only.’ Instead, I edit library code in the free and popular [Notepad++](#) editor.

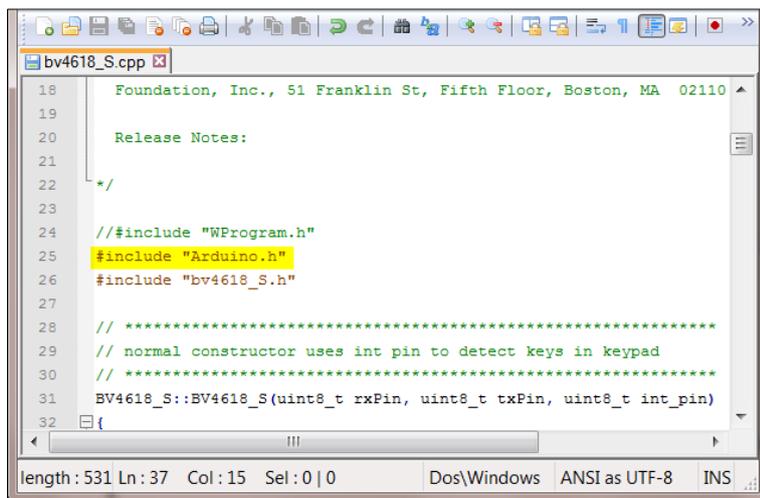
You can see *the bv4618_S.cpp* file in Figure 25. Sure enough, on line 24 we see the statement `#include WProgram.h`. The compiler complained this file does not exist, or at least it cannot find it. This error is quite common when you are using libraries that were written before Arduino IDE version 1.0 was released. There were many changes made to the Arduino code libraries in 1.0 including the removal of `WProgram.h`. The new header file is named `Arduino.h`. Many older programs were modified to include a conditional statement including `WProgram.h` if the IDE is older than 1.0 and conversely including `Arduino.h` if version 1.0 or newer.

This library code does not contain this conditional include. To fix this error, we just comment out line 24 and add the new header file. This is shown in Figure 26.



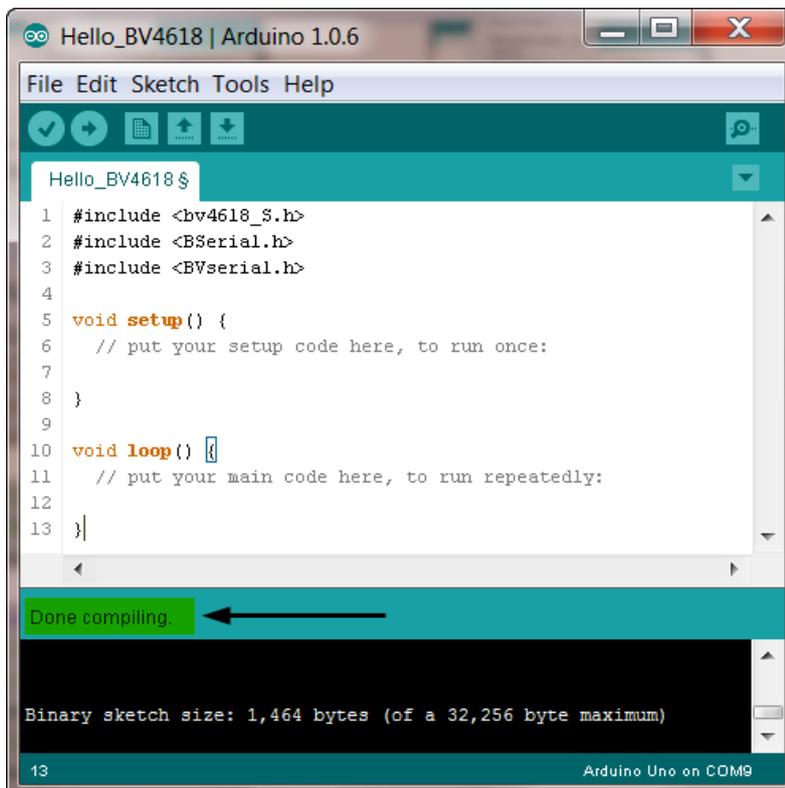
```
18 Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110
19
20 Release Notes:
21
22 */
23
24 #include "WProgram.h"
25 #include "bv4618_S.h"
26
27 // *****
28 // normal constructor uses int pin to detect keys in keypad
29 // *****
30 BV4618_S::BV4618_S(uint8_t rxPin, uint8_t txPin, uint8_t int_pin)
31 {
32     int pin = int_pin;
```

Figure 25 – WProgram.h include statement error



```
18 Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110
19
20 Release Notes:
21
22 */
23
24 // #include "WProgram.h"
25 #include "Arduino.h"
26 #include "bv4618_S.h"
27
28 // *****
29 // normal constructor uses int pin to detect keys in keypad
30 // *****
31 BV4618_S::BV4618_S(uint8_t rxPin, uint8_t txPin, uint8_t int_pin)
32 {
```

Figure 26 – Arduino.h include statement added



```
Hello_BV4618 | Arduino 1.0.6
File Edit Sketch Tools Help
Hello_BV4618 $
1 #include <bv4618_S.h>
2 #include <BSerial.h>
3 #include <BVserial.h>
4
5 void setup() {
6     // put your setup code here, to run once:
7 }
8
9
10 void loop() {
11     // put your main code here, to run repeatedly:
12 }
13 }

Done compiling
Binary sketch size: 1,464 bytes (of a 32,256 byte maximum)
13 Arduino Uno on COM9
```

Figure 27 – Hello_BV4618 clean compile

“If it works out of the box – what fun is that?”

Once the corrected *bv4618_S.cpp* file is saved to disk, clicking on the checkbox button at the top of the IDE window results in a clean compile. Our edit fixed the problem (Figure 27). At this point we are ready to add some code to write to the LCD. Figure 28 is a simple program that writes “Hello World!” to the LCD.

```

Hello_BV4618 $
1 #include <bv4618_S.h>
2 #include <BSerial.h>
3 #include <BVserial.h>
4
5 // Create an instance of the BV4618_S class
6 // BV4618_S bv(rxPin, txPin, int_pin)
7 BV4618_S bv(1, 0);
8
9 void setup() {
10 // put your setup code here, to run once:
11 // Set baud rate, delay, and ACK char
12 bv.begin(9600, 50, '*');
13 // LCD is 4x20
14 bv.puts("\e[4L\e[20c"); // 4 x 20 display
15 // Clear the screen
16 bv.puts("\e[2J");
17 delay(50);
18 // Hide the cursor
19 bv.puts("\e[?25I");
20 // In honor of Dennis Ritchie!
21 bv.puts("\n Hello World!");
22 }
23
24 void loop() {
25 // put your main code here, to run repeatedly:
26 // Nothing to do so this is empty
27 }

```

Figure 28 – “Hello World!”

Table 1 below describes what each line of code does.

Line	Action
7	Creates an instance of the BV4618_s C++ class
12	Sets the baud rate, standard delay, and ACK char
14	Sends the proper VT100 escape codes to set display at 4x20
16	Sends the proper VT100 escape codes to clear the display
17	Clearing the screen takes time – wait briefly
19	Sends the proper VT100 escape codes to hide the cursor
21	It all started with the late Dennis Ritchie

Table 1 – Code annotations

When you compile the *Hello World* program and upload it to your Arduino, you should see the string “Hello World!” centered on row 2 of the LCD.



Figure 29 – “Hello World!” success

“If it works out of the box – what fun is that?”

6) Summary

So there you have it. A *BV4618* LCD under your complete control. This 'How-To' explores all the steps required to get a *ByVac BV4618* LCD up and running using an Arduino UNO and IDE on the TTL serial interface. Be sure to reference the *BV4618* datasheet for the VT100 format of the command set

From this point on, you should have enough information to program the LCD for you specific project. If you have trouble getting started, have edits or enhancements to this document, or have other questions you can contact me via email at sopwith@ismellsmoke.net.

You can also visit my blog at sopwith.ismellsmoke.net.

Hack On!

Sopwith