Tips to Avoid Trouble

Following is a list of suggestions that will help you keep track of your files.

1. Do not use any special characters (blank, dot, dash, semicolon, etc.) in directory and file names even if they are allowed by the operating system. English letters, digits, and underscore (_) are the valid characters. Other characters may cause problems when you try to access a design tool or a project file using Windows command interface. For example, the complete file path cannot be identified if a directory or file name contains a blank, unless the entire file path is enclosed between quotation marks. Turkish letters (ç, Ç, ğ, Ğ, ı, İ, ö, Ö, ş, Ş, ü, Ü) may cause trouble when you copy or send files to another computer although they look fine on your computer.

2. Do not install the design and programming tools in the "Program Files" directory because its name contains a blank. Make a new directory such as "EEtools" and install all professional software in that directory avoiding any special characters.

3. Do not place your project on your desktop (masaüstü), in "My Documents", or any similar directory that is actually located somewhere deep in the C: drive. The actual path information will be too long and it will contain blank characters. A directory path, such as ""

"C:\Users\<user name>\My Documents\EExxx\Project1\" or

"C:\Documents and Settings\<user name>\Desktop\EExxx\Project1\"

is not completely visible in window titles or project file listings. If you cannot identify the complete file path, then you may end up editing a different set of files while the compiler or the design tool uses the files in another directory. You should go to your user partition and make an easily accessible directory, such as

"D:\EExxx\Project1\"

4. Make at least two partitions when you initialize your hard disk next time. One partition is for installation of operating system and other executable programs and the second partition is for user files (pictures, music, homework, projects, etc.). Most of the malicious programs, such as viruses and spyware, invade the system partition. Next time your computer gets an infection; you can format the system partition only, and leave your user partition intact. It is advisable to have a third partition to keep the backup copies of setup/installation files of the design and programming tools and other application programs. You can use these backup copies to install the applications easily, without downloading them again.

==

Necessary Software

Arduino main boards have the AVR series processors manufactured by Atmel. You will need the following software utilities to develop and compile source code for Arduino and to load the resultant executable code into the FLASH memory of the processor.

1. Arduino Software (~88 MBytes)

`http://arduino.cc/en/main/software`

Arduino software provides an easy-to-use programming platform for Arduino processor boards and other add-on hardware. You cannot use Arduino software for programming in C. Installation of this software is necessary because of two reasons:

- The AVRdude utility is required to program the FLASH memory of the Arduino processors. AVRdude version installed with the Arduino software is for sure compatible with the Arduino boards.
- You may need the Arduino libraries to access the add-on hardware, such as the wireless communication boards.

Useful Arduino reference pages:

Listing of all Arduino Tutorials:
`http://arduino.cc/playground/Main/TutorialList`

Arduino Development Tools:
`http://playground.arduino.cc/Main/DevelopmentTools`

Arduino C programming tutorials
`http://arduino.cc/forum/index.php?topic=1539.0`

2. WinAVR Compiler (~28 MBytes)

`http://winavr.sourceforge.net/`

WinAVR is a collection of open source software development tools for the Windows platform. WinAVR includes the GNU GCC compiler for C and C++ supporting Atmel AVR series processors. All development tools, including the Arduino software, rely on the WinAVR compiler.

WinAVR References on the Internet:

AVR Libc Home Page:
`http://www.nongnu.org/avr-libc/`

AVR User's Manual in PDF:
`http://savannah.nongnu.org/download/avr-libc/avr-libc-user-manual-1.8.0.pdf.bz2`

AVR User's Manual in HTML:
`http://savannah.nongnu.org/download/avr-libc/avr-libc-user-manual-1.8.0.tar.bz2`

Online AVR Libc Modules Documentation:
`http://www.nongnu.org/avr-libc/user-manual/modules.html`

Online AVR Libc Interrupt Module Documentation:
`http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html`

3. Code::Blocks IDE (~100 MBytes)

`http://www.codeblocks.org/downloads`

Code::Blocks is an open-source cross-platform IDE (Integrated Development Environment) that supports multiple compilers. Its installation package contains the GCC compiler for the Windows platform. You can immediately start writing programs that will run on your PC. Code::Blocks can also be used for creating programs and applications for AVR, ARM, and many other platforms. Code::Blocks provides:

- A text editor with syntax highlighting for C and C++ programs.
- A project management tool that keeps track of the source files and other resources in a software development project.
- An easy interface to the WinAVR compiler. This interface activates the GCC AVR compiler specifying all source components of a project and the target locations and file names for the object code and executable files that will be generated by the compiler.

You can easily introduce the WinAVR compiler to Code::Blocks (if it is not done automatically) that supports program development for Atmel AVR processors. If you need an ARM compiler in the future, then you can download the GCC ARM compiler and use Code::Blocks as a convenient interface.

Other Software

Dev-C++ IDE (~9 MBytes)
`http://www.bloodshed.net/devcpp.html`

Dev-C++ was developed for older operating systems, such as Windows 2000 or XP. You may end up with using Dev-C++ if you have an old system that has some compatibility problems with Code::Blocks. Dev-C++ is not a cross-platform IDE so it cannot be directly linked to WinAVR compiler. You can use Dev-C++ as a project manager and program editor. Dev-C++ can be configured to run WinAVR compiler through a custom Makefile.

A new version of Dev-C++ was recently released in 2013 under the project name, **Orwell Dev-C++**. There is no information about the compatibility of this new version with the old Windows systems. The home page of Orwell Dev-C++ is

`http://orwelldevcpp.blogspot.com/`

and the installation file (~43 MBytes) can be downloaded at

`http://sourceforge.net/projects/orwelldevcpp/`

Atmel Studio (~760 MBytes)
`http://www.atmel.com/design-support/software-tools/`

Atmel Studio is another IDE provided by Atmel. It uses the WinAVR compiler like any other development tool for the AVR processors. Atmel Studio has an additional feature for simulation of Atmel microcontrollers.

Installation of Atmel Studio is not suggested unless its simulation capability is necessary. Atmel Studio is a bulky program based on Microsoft Visual Studio and it also installs a version of Microsoft .NET even if you already have it on your computer. In case you decide to use Atmel Studio for some reason, you must select the correct CPU type and define the CPU clock frequency for the compiler as follows:

- Select **Project Properties** under **Project** menu
- Select **Toolchain** tab
- Select **AVR/GNU C++ Compiler/Symbols**, and add "**F_CPU=16000000L**" to the list of defined symbols.

AVRdude - AVR FLASH Loader
`http://www.nongnu.org/avrdude/`
`http://savannah.nongnu.org/projects/avrdude`
`http://sourceforge.net/projects/avrdude-gui/`

AVRdude is the program that loads the executable code into the FLASH memory of the processor. A Windows executable of AVRdude is included in WinAVR and Arduino packages. **AVRdude-gui** is a simple GUI (Graphical User Interface) for AVRdude which is a command line tool to program the Atmel AVR Microcontrollers. The most practical solution is to define AVRdude as a tool in Code::Blocks for quick access (see below).

===

Arduino Software

Installation:

1. Create an Arduino installation directory (remember: no special characters).

2. Extract **arduino-1.0.1-windows.zip** contents.

3. Copy extracted contents into the Arduino installation directory.

4. Connect USB cable to Arduino board.

5. Open **Control Panel>System>Device manager**.

6. **Update driver** for the "**Unknown Device**" in the "**Ports (COM & LPT)**" category:

- Do not let Windows do anything for you.
- Browse to the "**Drivers**" directory in the Arduino installation directory.
- Click on Install.
- Arduino board should be listed as a port (i.e. Arduino Mega 2560 R3) in the Windows Device Manager. Take a note of the **COM port number** specified for the device.

7. Find the AVRdude configuration file, **avrdude.conf**, file in the following directory:
`<Arduino installation directory>\hardware\tools\avr\etc\`

Place a second copy of **avrdude.conf** in the following directory:
`<Arduino installation directory>\hardware\tools\avr\bin\`

Running Arduino Software:

1. Connect USB cable to Arduino board.

2. Run Arduino.exe.

3. Select the proper "**Sketchbook location**" under **File>Preferences**.

4. Select "**Arduino Mega 2560 or Mega ADK**" under **Tools>Board**.

5. Select the COM port specified in the device manager under **Tools>Serial Port** (you should mark it even if it is the only selection).

6. Select "**Arduino as ISP**" under **Tools>Programmer**.

7. Open the "**Blink**" example and save it as a new sketch.

8. Yellow LED on the board is controlled by the MCU port output that drives pin-13 of the board connectors. If you want to verify that the MCU FLASH is really loaded then program a different blink pattern:

```
void loop()
{ digitalWrite(led, HIGH);   // turn the LED ON
  delay(250);                // wait for 250ms
  digitalWrite(led, LOW);    // turn the LED OFF
  delay(250);                // wait for 250ms
  digitalWrite(led, HIGH);   // turn the LED ON
  delay(250);                // wait for 250ms
  digitalWrite(led, LOW);    // turn the LED OFF
```

```
    delay(1000);               // wait for a second
  }
```

9. Compile the program: **Sketch>Verify/Compile**.

10. Send program to the Arduino board: **File>Upload**. The blink pattern changes after a successful upload.

===

Code::Blocks

Run the Code::Blocks setup program and install it in a directory created without any special characters preferably. The default compiler of Code::Blocks is the GNU GCC Compiler that can be used for the programs written for Windows environment. You will need to follow a few configuration steps to add WinAVR as a compiler option for your projects.

Adding WinAVR compiler:

1. Select the **Compiler...** option under the **Settings** menu.

2. Use the **Global compiler settings** option on the left.

3. Select the **GNU AVR GCC Compiler** in the pull-down list.

4. Click on the **Toolchain executables** tab

5. Under the Compiler's installation directory select the directory where **WinAVR** is installed.

6. In the **Program Files** tab, make sure that the following executables are listed:

C compiler: `<WinAVR installation dir>\bin\avr-gcc.exe`
C++ compiler: `<WinAVR installation dir>\bin\avr-g++.exe`
Linker for dynamic libs: `<WinAVR installation dir>\bin\avr-g++.exe`
Linker for static libs: `<WinAVR installation dir>\bin\avr-ar.exe`
Debugger: `GDB/CDB debugger : default`
Resource compiler: *Leave empty, no need for a resource compiler*
Make program: `<WinAVR installation dir>\utils\bin\make.exe`

7. In the **Additional Paths** tab, add the following directory where make.exe is located.

```
<WinAVR installation dir>\utils\bin
```

8. Click on the **OK** button.

Adding AVRdude as a Tool in Code::Blocks:

AVRdude program can be executed by opening the Windows Command Prompt. You should type the access path to avrdude.exe followed by the required command options that specify the MCU type, communication parameters, and the file to be loaded into the MCU FLASH. You can create a menu item under the Tools menu of Code::Blocks for quick access to AVRdude program.

1. Select **Configure tools...** under the **Tools** menu.

2. Click on the **Add** button to open the **Edit tool** window and enter the required information:

Name: Any meaningful name (i.e. AVRdude>>Uno) that will appear under the **Tools** menu.

Executable: Locate **avrdude.exe** in the Arduino installation directory:

```
<Arduino installation dir>\hardware\tools\avr\bin\avrdude.exe
```

Parameters: Enter the required command options given below:

For Arduino Uno:

```
-patmega328p -carduino -P\\.\COMn -b115200 -D -v -v -v
-Uflash:w:${PROJECT_DIR}${TARGET_OUTPUT_DIR}${PROJECT_NAME}.hex:i
```

For Arduino Mega:

```
-patmega2560 -cstk500v2 -P\\.\COMn -b115200 -D -v -v -v
-Uflash:w:${PROJECT_DIR}${TARGET_OUTPUT_DIR}${PROJECT_NAME}.hex:i
```

The `COMn` parameter should be replaced by the serial port (i.e. `COM3, COM4`) assigned to **Arduino Uno R3** or **Arduino Mega 2560 R3** in the Windows Device Manager. The macro entries specified as `${...}` are automatically replaced with the corresponding parameters according to the active project.

Note that the `COM` port number may change for some reason after the installation. If you receive the error message,

"*can't open device "\\.\COMn": The system cannot find the file specified.*"

then open the Windows Device Manager and verify the `COM` port number given for the Arduino board.

Working directory: `${PROJECT_DIR}`

```
<Arduino installation dir>\hardware\tools\avr\bin
```

3. Select the first Launching option, "***Launch tool in a new console window and wait for a key press when done***", to see the errors and other critical information in the log output generated by AVRdude.

You can start the FLASH loading operation simply by selecting the AVRdude menu item created under the Tools menu. You should start AVRdude immediately after pressing the reset button on the Arduino processor board. The boot loader on the main processor waits only for one second to check if there is a new program to be loaded. See the following section on loading FLASH memory on Arduino Boards for detailed information.

Create an AVR project in Code::Blocks IDE:

1. Click on the **Create a new project** option.

2. Select the **AVR Project** option.

3. Enter the **project title** and select the folder where the project folder will be created.

4. Uncheck the **Create "Debug" configuration** box.

5. Choose the processor, **atmega328p** (or **atmega2560** for Arduino Mega), and define **F__CPU** as **8000000UL** for Proteus simulations (or **16000000UL** for Arduino Uno and Mega boards). The CPU clock frequency is critical for proper execution of the time delay macros (i.e. `_delay_us(...)` defined in **<util/delay.h>**). Make sure that "**Use external memory**" box is unchecked and "**Create hex files**" box is checked.

The executable code generated by WinAVR should be written into an output file with the "**hex**" extension. This output file is a plain text file that contains the byte-by-byte hexadecimal programming data in Intel hex format.

6. Click on the **Finish** button.

7. Add/remove source files using the **Project** menu. You should keep the source file, **fuse.c**, just to avoid build errors. The fuse settings generated by the compiler will be ignored in Proteus and programming of fuse bits on the Arduino board is not possible through USB interface. In Proteus, CPU clock and the other necessary parameters for simulation can be entered as device properties on the schematic.

==

Using WinAVR Without an IDE

Compiler packages contain a "**make**" program that can execute a series of instructions specified in a **Makefile**. Running a compiler requires entering a series of long commands with several compiler options. These commands can be specified in the **Makefile** instead of typing them over and over again for every compilation. The "**make**" program supports some features that provide flexibility and convenience when preparing a Makefile.

- You can use **macros** to avoid repeated entry of file names and other settings. For example, a `TARGET` symbol can be defined as "`TestProject1`":

  `TARGET = TestProject1`

  After this definition, the macro, `$(TARGET)`, will be replaced by the text string, "`TestProject1`", anywhere in the Makefile
- The Makefile instructions can be organized under separate sections. One of the sections can be selected by typing the section title with the make command. The two sections common in all Makefiles are

  `all` : compiles source files and rebuilds the executable code when "`make all`" command is entered.

  `clean` : removes all files generated by the compiler when "`make clean`" command is entered. A "`make clean`" is necessary before entering "`make all`" to compile and rebuild all source files.

You can install WinAVR and prepare a **Makefile** to compile your program entering "**make**" commands at the Windows Command Prompt. For those with a little experience in using compilers, typing a few filenames and compiler options in the makefile can be easier than entering the same information through a GUI.

1. Create a project directory (i.e. `D:\Projects\P1`).

2. Write your program in C, C++, or assembly language using any text editor, and save the files as **text-only** in the project directory.

3. Copy the sample **Makefile** provided with WinAVR from

`<WinAVR installation dir>\sample`

into your project directory.

4. Start with the sample Makefile and enter the necessary parameters as described below.

Specify MCU type: `MCU = atmega2560`

Specify CPU clock frequency: `F_CPU = 16000000`

Name of the **.hex** file: `TARGET = <project_name>`

If you like to keep the object files out of sight, then specify an "**obj**" directory in the project directory: `OBJDIR = obj`

If you gave a name other than the `<project_name>` to your source file, or if there are other source files, then list them here: `SRC = $(TARGET).c`

5. Start the **Windows Command Prompt**.

6. Change the working directory to your project directory. For example:

```
C:\>d:
D:\>cd Projects\P1
D:\Projects\P1>
```

The **Makefile** prepared for the project should be located in the working directory when you enter the "**make**" commands.

7. Compile your program entering the "`make all`" command:

```
D:\Projects\P1><WinAVR installation dir>\utils\bin\make.exe all
```

If you want to rebuild all source files in the project then you should first enter the "`make clean`" command:

```
D:\Projects\P1><WinAVR installation dir>\utils\bin\make.exe clean
```

The "`make clean`" command removes the previously generated compiler output files so that all source files will be compiled again next time you enter the "`make all`" command.

You can use the "`make program`" command to run AVRdude, but you must first customize the AVRdude settings in the sample Makefile. You can also run AVRdude at the Windows Command Prompt to load the generated **.hex** file into the Arduino processor memory. You need to enter the following command after replacing the required options according to the sytem and project settings:

```
<Arduino installation dir>\hardware\tools\avr\bin\avrdude.exe -
patmega2560 -cstk500v2 -P\\.\COMn -b115200 -D -v -v -v -
Uflash:w:<project_directory>\<project_name>.hex:i
```

`<Arduino installation dir>`: main directory of Arduino installation

`COMn`: should be replaced by the serial port (i.e. COM3, COM4) assigned to the **Arduino Mega 2560 R3** in the Windows Device Manager.

<project_directory>: the directory that contains the **.hex** file

<project_name>: the target name assigned to the **.hex** file in the Makefile

==

Loading FLASH Memory on Arduino Boards

Arduino main processor boards have two microcontroller units (MCUs):

1. **ATmega2560** (or **ATmega328** on Arduino Uno) is the **main MCU** that runs your programs. The executable code generated by the compiler is loaded into the FLASH memory in this MCU.

2. **ATmega16U2** handles the USB interface. It takes the programming information received through the USB cable and sends the necessary data to ATmega2560 or ATmega328 through a two-line serial connection.

The USB interface to Arduino board appears as a serial communication port (i.e. COM3, COM4, etc.) in the Device Manager list of your computer. This USB interface can also be used for bidirectional data transmission between your computer and the main MCU on the Arduino board.

The main MCU's FLASH memory is divided into two sections. The main section of the FLASH memory stores the application program. The second section is reserved for a **boot loader** program that takes a small part of the FLASH memory. After a reset condition, the processor can be directed either to the starting address of main application (i.e. 0x0000) or to the boot loader program. The target program address is determined according to the "**fuse bits**" that are a small programmable memory to control the basic MCU options. Arduino boards are shipped with the boot loader program stored in the reserved section. The main section of the FLASH memory is available for the user programs. The following steps take place every time the main MCU is reset:

1. Processor starts execution of the boot loader program.

2. Boot loader waits about one second while it continuously monitors the serial connection to ATmega16U2.

3. If no instructions are received from ATmega16U2 during the wait period, then boot loader simply executes a "JUMP" instruction to 0x0000 address and gives control of the MCU to the previously stored program in the main FLASH memory. If boot loader receives a load instruction from ATmega16U2 during the wait period, then it carries on the following steps.

3. Boot loader receives the new executable code through the USB interface and loads it into the main part of FLASH memory.

4. Boot loader gives control of the MCU to the new program stored in the FLASH after the load operation is completed.

Memory organization of user programs written for Arduino boards are not any different with or without a boot loader. The user program starts running at the program address 0x0000 directly after a reset condition when there is no boot loader. In the other case, boot loader first takes control of the MCU after a reset and then starts the user program at 0x0000. Once the user program starts running, boot loader cannot be activated until the next reset condition.

In short:

- Write your program

- Compile your program and obtain the executable code in a **.hex** file.
- Push the reset button on the Arduino board.
- Start AVRdude in less than a second, while boot loader is still running.

==