



WinAVR 101

How to use WinAVR with AVR Studio 4, Visual C++ 2008 Express and Eclipse to program your AVR controller in C



OVERVIEW

This is a guide for setting-up the programming environment required to create, compile and upload your C code to your AVR controller boards.

Both Supermodified and Arduino controller boards are built around ATMEL's ATMEGA 48p/88p/168p/328p family which is an AVR - based "C. Once these simple instructions are followed you will be able to compile and upload your code to any AVR - based controller, so this document can come really handy.

Before we make a start it is necessary for you to download the following open - source programs:

- WinAVR
- AVR Studio
- Microsoft Visual C++ Express 2008
- Eclipse

Let's have a closer look at these tools so their functionality becomes clear to those who are not awfully familiar with them.

WinAVR

WinAVR is ATMEL's native compiler comes with a suite of executable, open source software development tools for the Atmel AVR series of RISC microprocessors hosted on the Windows platform. It includes the GNU GCC compiler for C and C++.


Download WinAVR from: <http://winavr.sourceforge.net/> and run WinAVR_install.exe on your computer.

Following a successful installation, go to:

Start → Programs → WinAVR → MFile

This is a **Makefile** generation utility. It allows you to configure your compiler according to your project needs and produce the hex files that will be later uploaded to your "C. Here is where you instruct your compiler what device you are using so it can load the appropriate register map, what is the oscillator speed the device will run at, the target file name, compiler optimisation options etc. You have to make sure the following settings are chosen from the **Makefile** drop-down menu:

Main File Name:	your_main_filename.c	
MCU Type:	ATMEGA328P	(your controller here)
Output Format:	ihex	
Optimisation level:	2	
Debug level:	ELF/DWARF-2	
C Standard Level:	gnu99	



Once you are happy with the configuration settings, go to:

File -> Save As.

and save the **Makefile** in your computer, later to be placed inside your project wannabe directory.

You can also edit the Makefile by hand using WinAVR's native text editor or any other standard text editor like notepad (or notepad++, SciTe etc). Experienced users tend to prefer this option, but novice users are advised to stick to this guide in order to avoid silly downfalls.

If you wish to compile your code manually the following steps can be followed:

1. Create a project directory.
2. Use a text editor to write your C code and save it using a **.c** extension. If header files are required they should be placed inside the project directory as well.
3. Use **Mfile** as instructed above to create the required **Makefile**.
4. Open a **Command Prompt** window and navigate to the project directory.
5. Type in → **make your_filename.c** and press **Enter**

The output of the compilation is going to be your_filename.**hex** file which you can then directly flash to your controller using an appropriate programmer.

In conjunction with a text editor, WinAVR offers all the necessary functionality to fully implement your AVR projects. Even though this true, there are much more advanced tools available that allow maximisation of your work output and minimisation of development times. Of course programming is all about logic behind processes, and this you have to learn and get accustomed to yourself. Nonetheless there are tools available that can make your life a bit easier.

Please read carefully...

AVR controllers come with truly amazing support libraries which have been built from guru users over the years for the community. For references one needs to look no further than the AVR LibC documentation:

<http://www.nongnu.org/avr-libc/>

This place is full of useful info and really helps to understand generic principles behind the functionality of the libraries that support the AVR.

Last but not least, the Procyon AVR Library written by **Pascal Stang** offers an enormous source of working sample projects that you can use, built upon and learn from:

<http://www.mil.ufl.edu/~chrisarnold/components/microcontrollerBoard/AVR/avrilib/docs/html/index.html>



AVR Studio 4

Starting-off with ATMEL's Integrated Development Environment (IDE), the AVR Studio comes with:

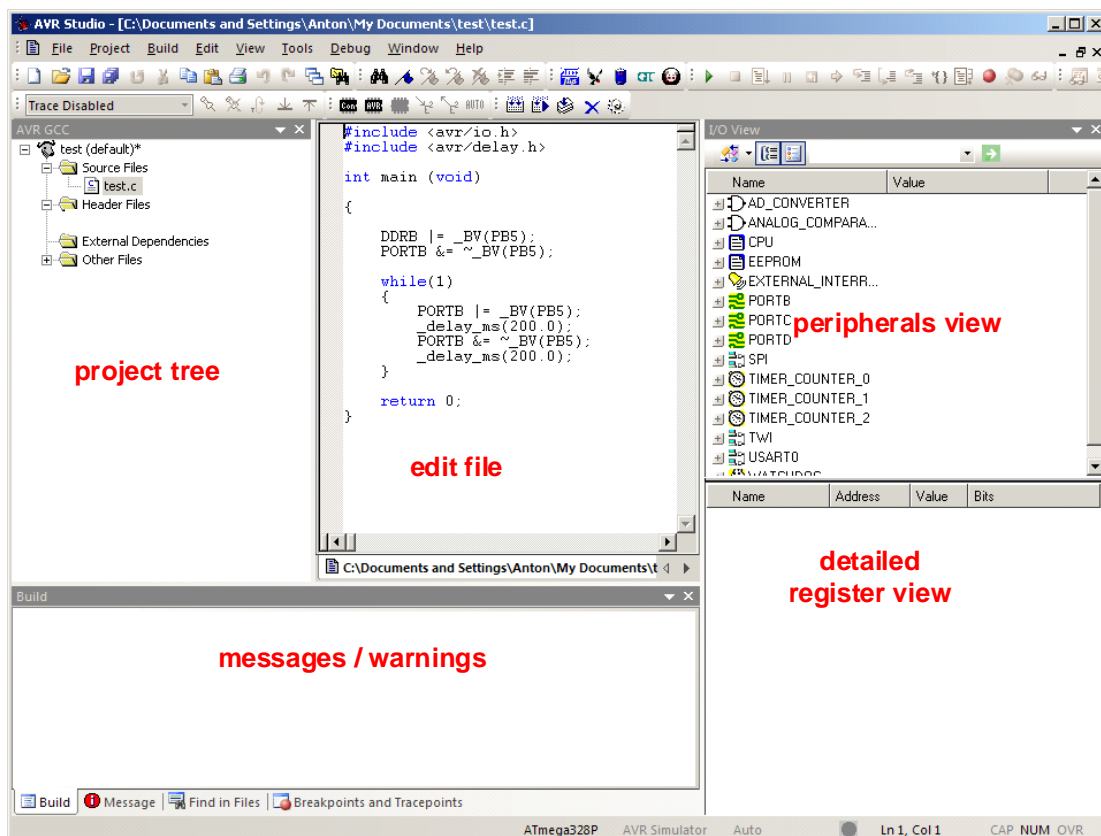
- a built-in text editor,
- a debugger supporting program run control including source and instruction-level stepping and breakpoints,
- registers, memory and I/O views,
- target configuration and management,
- full programming support for standalone programmers.

The installation procedure is as follows:

Download and run AVRStudioXX.exe from:

http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725

together with any available Service Packs. At the time of writing the AVR Studio 4 comes with the 4.18 SP1. Once the installation is complete you are ready to start your first project.



Above you can see a snapshot from the AVR Studio 4 with a sample project.

To start your project go to: **Project → New Project** and choose the **AVR GCC** compiler in the **Project Type** field.

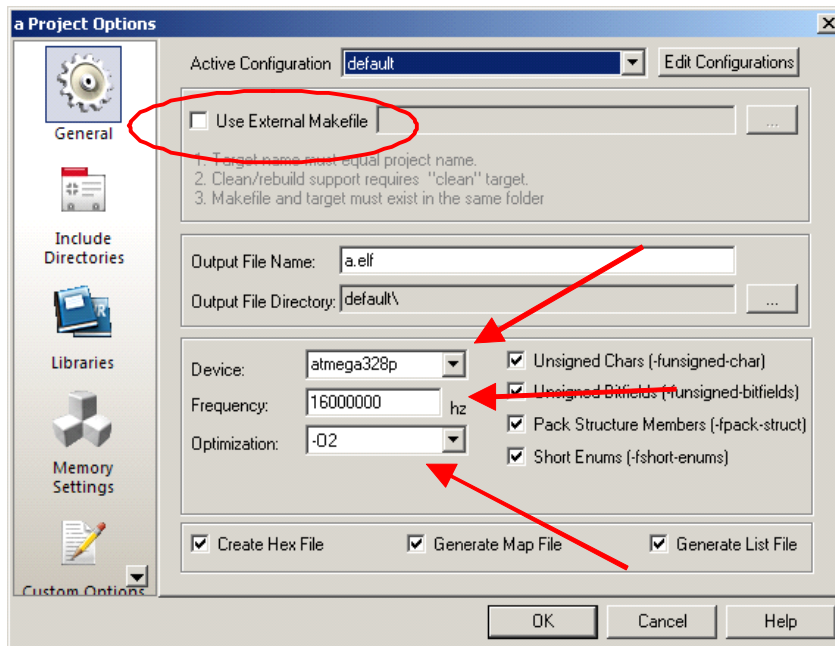
In the **Project Name** field you are required to enter the name of your project. AVR Studio will automatically update the field **Initial file** to match the project name. Note this will be a **.c** file, since you will be using the AVR GCC compiler. The **Location** field indicates the directory you are going to save your project.

Click **Next**.

The next dialogue allows you to select a **Debug Platform** as well as, the **Device** you are going to be programming. At this instance we will not be using a debugger so select the **AVR Simulator** from the **Debug Platform** field. Following that you must specify the **Device** you will be programming.

Click **Finish**.

By default you should be able to see your `project_file_name.c` open and ready to edit. In the top-left window you can find a description of your project in terms of: **Source Files**, **Header Files**, **External Dependencies**, as well as **Other Files** associated with your project. At this point there are no source or header files associated with your project. These you must add manually by right-clicking on the relevant field on the top-left window. Under Other Files you can include the Makefile produced from WinAVR as described earlier above.



The Makefile must be manually copied in the project directory. Once included in the project the Makefile can also be manually edited using the AVR Studio's text editor. Alternatively AVR Studio offers an internal Makefile generation utility shown above. To access the **Project Options** pane go to:

Project → Configuration Options

The **Options** dialogue appears where you can specify all the project settings like we did earlier with WinAVR. If you wish to use the Makefile produced by the WinAVR you must choose the **General** tag, tick the **Use External Makefile** option and manually locate the file.

After including your Source and Header files and specifying the Configuration Options (or use external Makefile) you must build your project. To do so, go to:

Built → Build

If there are no errors your **.hex** file has been generated and you are ready to download the code onto your controller. There is a multitude of different programmers to do that. One can only find out for themselves by having a look on the web. For a really cheap solution check:

<http://www.olimex.com/dev/avr-pg1.html>

The description of different programmers and their capabilities are beyond the scope of this document. I will be using the commonly available AVRISP mkII programmer since it is an official ATMEL programmer which implies that is guaranteed to perform and after all it costs about 20\$.



Ensure the programmer is connected to your USB port after the installation of the drivers that came with it. Connect your Arduino or Supermodified board to the power supply and the programmer. Go to:

Tools → Program AVR → Connect

You will be prompted to select your programmer and the connection port you will be using. Choose **AVRISP mkII** from the menu. **USB** is highlighted by default. Click **Connect**.

An '**AVRISP mkII in ISP mode with ATmegaXXXX**' menu now pops-up with multiple tags. Under the **Program** tag, in the **Flash** field browse to your project file and locate the **.hex** file produced earlier. Click **OK** and on the **Flash** field click **Program**.

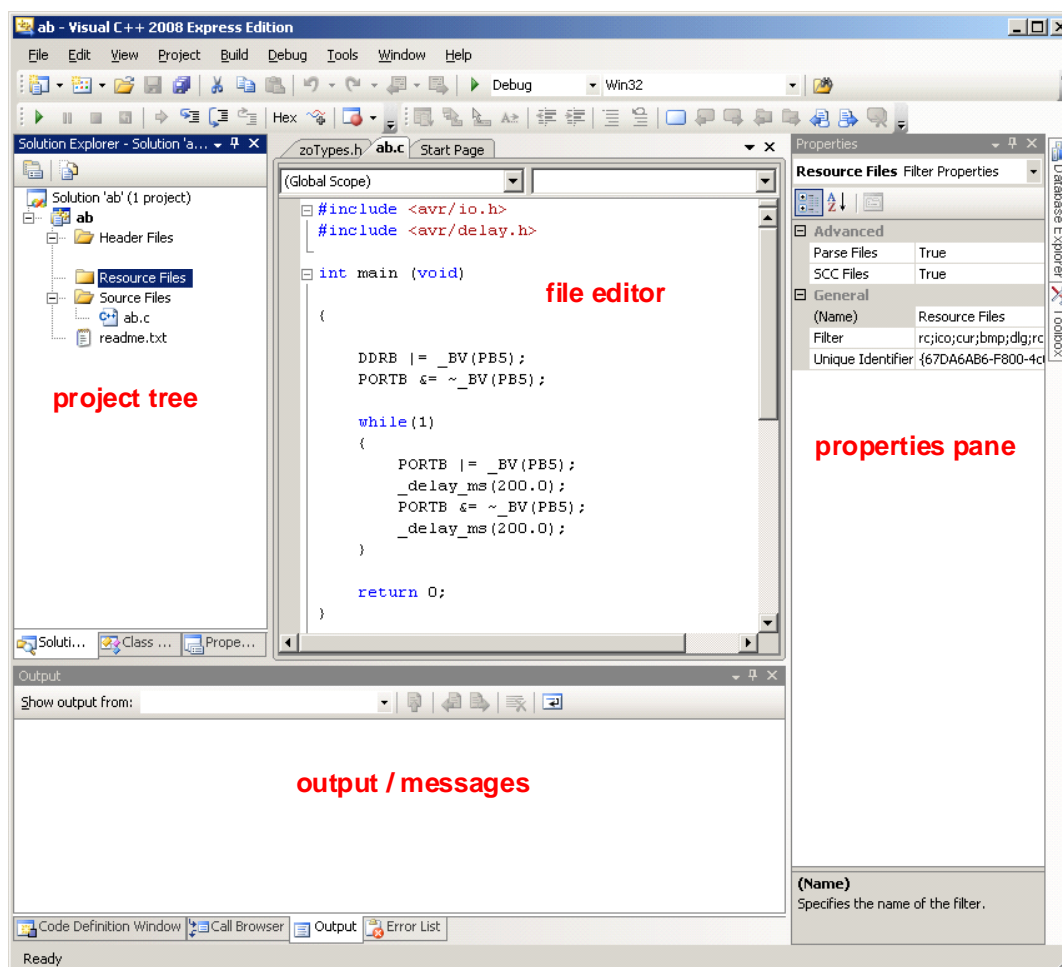
Your code now resides inside your Supermodified / Arduino controller!

Visual C++ 2008 Express

The AVR Studio is free to download and install and comes as a complete solution for ATMEL's low-range "Cs. Even though AVR Studio is very powerful there are other available free tools that integrate even more capabilities into the Integrated Development Environment. Next we will be looking at Microsoft's Visual C++ Express Edition. This also comes free with powerful highlighting and auto complete / reference text editor utilities. For more info check:

<http://msdn.microsoft.com/en-us/library/ms379615%28VS.80%29.aspx>

We will be using Visual C++ to generate the hex file that we will need, but we will still be using AVR Studio to flash the .hex into our controllers.



Shown above, a snapshot from the Visual C++ 2008 with a sample project.

The installation procedure for Visual C++ is as follows:

Download and run Visual C++ installation file from:

<http://www.microsoft.com/express/Windows/>

To start your project in Visual C++ 2008:

Go to **File** → **New** → **Project** and choose **General** in the **Project Type** field. On the **Template** field choose **Makefile Project**, and specify the project's **Name** and **Location**.

The next window, **Makefile Project Wizard**, allows you to set the **Debug & Release Configuration Settings**.

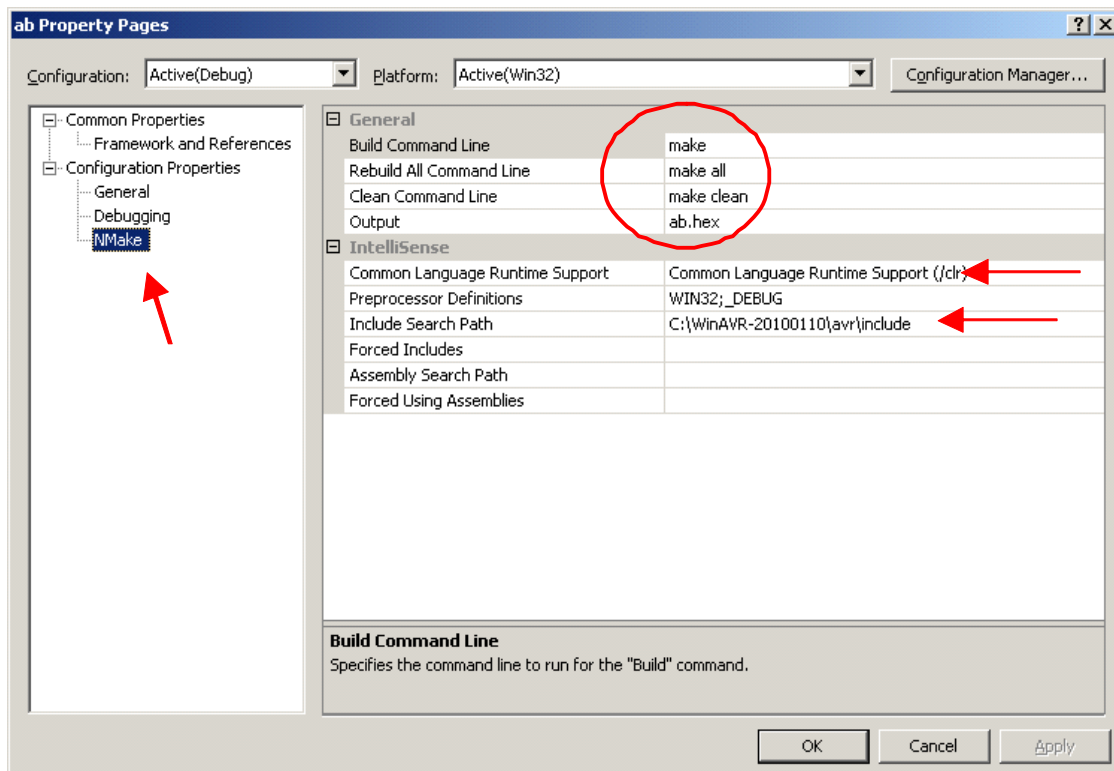
You are required to enter Built, Re-built and Clean commands for debugging and releasing the project files.

Built Command Line:	make
Re-built Command Line:	make all
Clean Command Line:	make clean
Output (for debugging):	your_project_name.hex

Once you set the above **Debug Configuration Settings**, continue to the **Release** dialogue and make sure that '**same as debug configuration**' is ticked. Click **Finish**.

This dialogue can also be accessed by:

Project → **Properties** → **Configuration Properties** → **NMake**



Also under the **Configuration Properties** tag, select **Common Language Runtime Support** → **Common Language Runtime Support (/clr)**. This enables the IntelliSense features of your Visual C++ text editor. These dynamic user code information features can really work miracles in large projects. Quick Information, Auto Completion and Parameter Help will undoubtedly assist you keep track of your project and produce cleaner code. Additionally under the **Include Search Path** field you are required to input the WinAVR includes directory path. This will allow you to see the contents of include files just by right-clicking and selecting '**Go to definition**'. Click **Apply** & **OK**.

You are now ready to include your Source and Header files to the project, much like you did in the AVR Studio. Similarly, you must manually copy the Makefile produced by WinAVR's NMake utility, and add it in your project directory. Following that you must add the file under the **Resource Files** field.

To compile your project go to: **Built → Built Project_Name**.

This will produce a hex project file that can then be downloaded onto your controller using ATMEL's AVR Studio like described above.

Eclipse with AVR plugin

The last Integrated Development Environment we will be looking through during this small tutorial is Eclipse.

The installation files can be downloaded for free from:

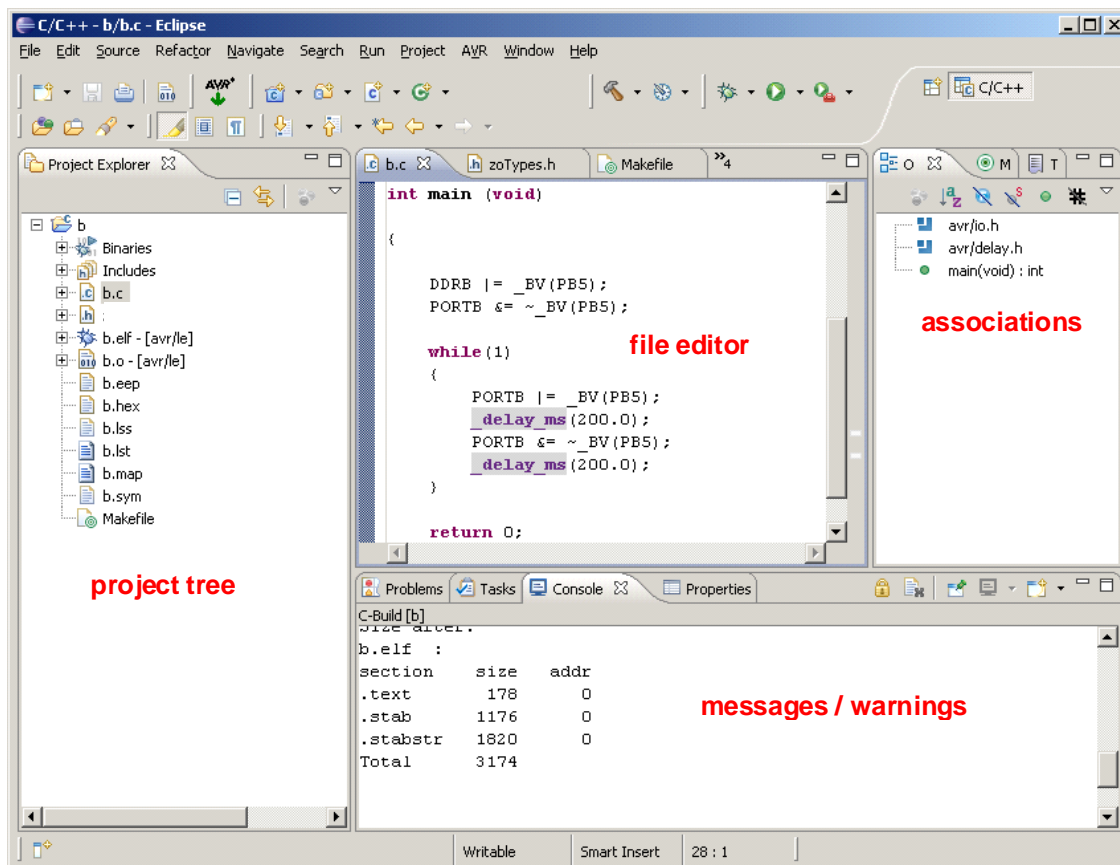
<http://www.eclipse.org/>

After the **.rar** file is downloaded you must extract it to a directory. This will be the 'installation' directory and can be placed anywhere since Eclipse is a stand-alone platform.

You will also need the AVR-Eclipse plugin. This can be downloaded from:

<http://sourceforge.net/projects/avr-eclipse/files/>

To install the plugin you simply extract the files from the downloaded **.rar** file and manually copy the extracted files to their corresponding directories (**features & plugin**) of the Eclipse installation directory.



Shown above, a snapshot from the Eclipse IDE with a sample project.



Once all files are placed in the desired directory simply run **eclipse.exe** to launch the Eclipse IDE. The AVR plugin is automatically loaded.

Next you are prompted to select your working directory. You are advised to place that in the root of your hard drive. If you need to nest your Eclipse working directory inside another directory make sure the name of the parent directory does not contain any spaces. Do so and click **OK** ;)

To start a new project in Eclipse go to:

File → New → C Project

The **C Project** window appears and in the **Project Name** field you are required to enter your desired project name. Here you can also choose a different project location to the default one. In the **Project Type** field select **Makefile Project** and in the **Toolchains** field, the **AVR-GCC Toolchain**.

The new project appears in the top-left window in a tree-like structure, which seems to be the norm for all IDE applications seen so far.

Source and Header files need to be created and/or added next. For once again the **Makefile** needs to be manually copied inside the working project directory.

Save your project, **File → Save**, and built it, **Project → Build All**.

Provided you get no errors you should now have available the **.hex** file in your project directory. This can be downloaded directly onto your Arduino and Supermodified boards using AVR Studio. It should be noted that Eclipse offers native AVR support and it could be used in conjunction with AVR Dude to program a controller directly through the Eclipse environment. This is going to be discussed in a later tutorial.

Happy building!



www.01mech.com

