Getting Started

Let's start with a simple example. We have an ATMega8 and want to blink an LED connected to Port D bit 4. To create a template project, run the following commands in a Terminal window:

Shell Session:

```
bash$ cd ~/Documents
bash$ mkdir AVR
bash$ cd AVR
bash$ avr-project Demo
bash$ open Demo
```

We have added a new subdirectory to your Documents folder and created a project named "Demo" in it. Then we opened the newly created folder in Finder. The contents of the folder are:

Shell Session:

```
bash$ cd Demo
bash$ ls -l
total 0
drwxr-xr-x   5 cs  cs  170 Nov 19 13:58 Demo.xcodeproj
drwxr-xr-x   4 cs  cs  136 Nov 19 13:58 firmware
```

You can later create new subdirectories for the circuit diagrams, construction drawings and maybe Mac software here. If you don't have Xcode installed, you can delete Demo.xcodeproj. The subdirectory `firmware` is intended for the AVR firmware, it contains:

Shell Session:

```
bash$ cd firmware
bash$ ls -l
total 24
-rw-r--r--   1 cs  cs  4139 Nov 19 13:58 Makefile
-rw-r--r--   1 cs  cs   348 Nov 19 13:58 main.c
```

`Makefile` controls the build process and `main.c` contains the C source code for the project. We first edit Makefile. We want to configure the fuse values so that the AVR runs on an internal RC oscillator at 8 MHz and choose our programming tool. For this example we assume that you use Thomas Fischl's USBasp. We edit the options at the beginning of `Makefile` to look like this:

Excerpt of Makefile:

```
DEVICE     = atmega8
CLOCK      = 8000000    # 8 MHz in Hz
PROGRAMMER = -c USBasp
FUSES      = -U hfuse:w:0xd9:m -U lfuse:w:0x24:m
OBJECTS    = main.o
```

The first two lines are obvious. The third line is *avrdude*'s command line option for the USBasp programmer. The fourth line contains *avrdude*'s command line options to program the fuses appropriately (see the ATMega8 data sheet for more information on fuse bits or use one of the tools in the Links section to calculate the values). And the last line lists all modules which should be compiled. That's the same as the list of source files, but with the file extension changed to `.o`.

Then we edit `main.c`, make it look like this:

main.c:

```
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRD = 1 << 4;          /* make the LED pin an output */
    for(;;){
        char i;
        for(i = 0; i < 10; i++){
```

```
        for(i = 0; i < 10; i++){
            _delay_ms(30);  /* max is 262.14 ms / F_CPU in MHz */
        }
        PORTD ^= 1 << 4;    /* toggle the LED */
    }
    return 0;               /* never reached */
}
```

Now we can run the build process in the `firmware` directory:

Shell Session:

```
bash$ make
avr-gcc -Wall -Os -DF_CPU=8000000 -mmcu=atmega8 -c main.c -o main.o
avr-gcc -Wall -Os -DF_CPU=8000000 -mmcu=atmega8 -o main.elf main.o
rm -f main.hex
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
```

If you use Xcode, you can also click the Build button to build the firmware. `Make` has now created the files `main.o`, the compiled version of `main.c`, `main.elf`, that is `main.o` linked with all required external libraries and finally `main.hex`, an Intel-Hex file ready for passing to the programmer:

Shell Session:

```
bash$ ls -l
total 48
-rw-r--r--   1 cs  cs  4142 Nov 19 14:05 Makefile
-rw-r--r--   1 cs  cs   367 Nov 19 14:06 main.c
-rwxr-xr-x   1 cs  cs  3161 Nov 19 14:07 main.elf
-rw-r--r--   1 cs  cs   390 Nov 19 14:07 main.hex
-rw-r--r--   1 cs  cs   808 Nov 19 14:07 main.o
```

The final step consists of uploading the code and fuses through the programmer:

Shell Session:

```
bash$ make flash
avrdude -c USBasp -p atmega8 -U flash:w:main.hex:i

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.19s

avrdude: Device signature = 0x1e9307
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "main.hex"
avrdude: writing flash (130 bytes):

Writing | ################################################## | 100% 0.21s

avrdude: 130 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: load data flash data from input file main.hex:
avrdude: input file main.hex contains 130 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.15s

avrdude: verifying ...
avrdude: 130 bytes of flash verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.
```

Shell Session:

```
bash$ make fuse
avrdude -c USBasp -p atmega8 -U hfuse:w:0xd9:m -U lfuse:w:0x24:m

avrdude: AVR device initialized and ready to accept instructions
```

```
Reading | ################################################## | 100% 0.19s

avrdude: Device signature = 0x1e9307
avrdude: reading input file "0xd9"
avrdude: writing hfuse (1 bytes):

Writing | ################################################## | 100% 0.19s

avrdude: 1 bytes of hfuse written
avrdude: verifying hfuse memory against 0xd9:
avrdude: load data hfuse data from input file 0xd9:
avrdude: input file 0xd9 contains 1 bytes
avrdude: reading on-chip hfuse data:

Reading | ################################################## | 100% 0.07s

avrdude: verifying ...
avrdude: 1 bytes of hfuse verified
avrdude: reading input file "0x24"
avrdude: writing lfuse (1 bytes):

Writing | ################################################## | 100% 0.20s

avrdude: 1 bytes of lfuse written
avrdude: verifying lfuse memory against 0x24:
avrdude: load data lfuse data from input file 0x24:
avrdude: input file 0x24 contains 1 bytes
avrdude: reading on-chip lfuse data:

Reading | ################################################## | 100% 0.07s

avrdude: verifying ...
avrdude: 1 bytes of lfuse verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.
```

The LED connected to Port D bit 4 should blink now.