Building Developing AVR a better way on Windows

Jon Wolfe
Anibit Technology
jonjwolfe@yahoo.com

WHY?

(Recapping from my Triembed presentation from 2 years ago)

- Ease of incorporating standard C/C++ code.
- Fantastic development support tools, and debugging!
- Much more customizable
- Free (as in beer)
- Plug-in support
  - AVR Specific
  - Visual Studio General
- Good integration with commercial programmer/debugger hardware
- No bootloader needed with external programmer. That’s more space for your code!
- Built-in simulator, so you can start prototyping before you have any chips or hardware!

Drawbacks

- Windows only
- Not open source
- While the IDE itself is not open source, the compiler and support command line tools are
- Often requires a deeper understanding of C/C++ and linking/building than arduino.
- Less “beginner friendly”
- With great responsibility comes great power.

What wrong with just using the Arduino IDE?

- The UI works ok for very small projects, but falls apart at scale.
- The build system is “mystical”, and hides standard C++ build practices.
- You don't *really* get away from C++ by using the Arduino “language”, they give you a life jacket, but the water still has sharks.
- Library management has made recent improvements, but is still awkward.

Chocolate and peanut butter

You can have the best of both worlds:

You can convert an Arduino project into an Atmel Studio project, and still use the Arduino libraries in your code!

There are 4 ways to do it:

1. Have an existing “raw C/C++” project, and pull in bits and pieces of an Arduino library.
2. Have an existing Arduino project that you manually convert to a “raw C/C++” project.
3. Use the new “Import Arduino project” option in Atmel Studio
4. Use the Visual Micro for Arduino plug in.

Pros/cons

Method 1: Have an existing “raw C/C++” project, and pull in bits and pieces of an Arduino library.

Pros:

- Least disruptive to your existing RAW c++ project.

Cons:

- You may end up pulling in more that you had planned or having to write “shim” code to satisfy some of the libraries dependencies.

Pros/cons

Method 2: Have an existing Arduino project that you manually convert to a “raw C/C++” project.

- Pros
  - You have a high degree of control over the conversion, and also know the state of the code when your done.
  - You can get very specific with your conversion, changing compiler and linker flags to suit special needs.
- Cons
  - The most amount of work to convert, especially a concern if you pull upstream changes from libraries.
  - Requires the most C++ language and compiler wisdom.

Pros/Cons

- Method 3: Use the new “Import Arduino project” option in Atmel Studio
- Pros:
  - Less work than options 1 & 2.
  - Changes your Arduino project into a C++ project.
- Cons
  - Not completely error free, sometimes manual changes may need to be made
  - Like method 2, future updates to published libraries may have to be manually incoporated.

Pros/Cons

- Method 4: Use the Visual Micro for Arduino plug in.
- Pros
  - Minimally invasive option, uses your existing project as-is (in other words, not an “import”)
  - Only option that has “debug over serial”*($$$)
- Cons
  - Subject to some of the same build system “voodoo” as the Arduino IDE, since it uses the the Arduino tools for building.
  - Free for development and building, but debuging features is a paid addon
  - Debug over serial is less capable than JTAG o DebugWire Atmel interfaces