Thought I would share this tutorial for importing Arduino projects into AvrStudio 4. AvrStudio 4 is Atmel's AVR IDE; it is full featured and has debug tools, just the things that the Arduino IDE is sorely lacking. Atmel has released a version 5, but I have had no luck in getting it to import the projects. Chatter on the boards is it is still too new and buggy. I have also had success in importing an Arduino project into the Eclipse IDE. I actually prefer the Eclipse IDE, but have yet to figure out the add on debugging tools.

As of this post I have not yet tried any debugging of the 328P, but no reason why it should not work. Anything is better than nothing. The main challenge was getting an Arduino project to compile and build.

Warning!!, I am no computer programmer or expert in these matters. It was only from the help of forum posts and other tutorials that I was able to cobble this together to work, so don't blame me if it is not done 100% correctly, or you toast your AVR.

First you will need to download AvrStudio 4. Install and then download the latest service pack which is SP3, this pack is at the bottom of the download screen.

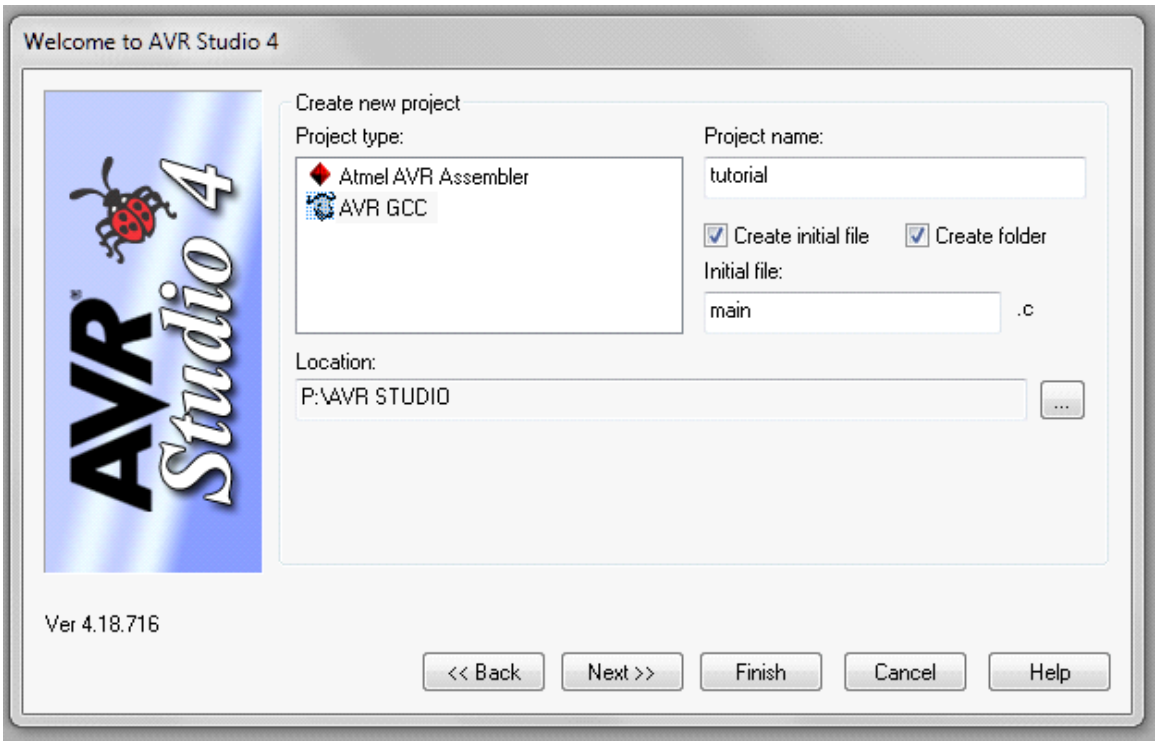
http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725
I am not sure if it is included with the AvrStudio 4 download, but it might be a good idea to download WinAVR which includes the GNU GCC compiler for C and C++.

The tutorial is based on these excellent forum posts from the Arduino forum and AvrFreaks forum. Read these not once, twice, but a dozen times so it all makes sense.

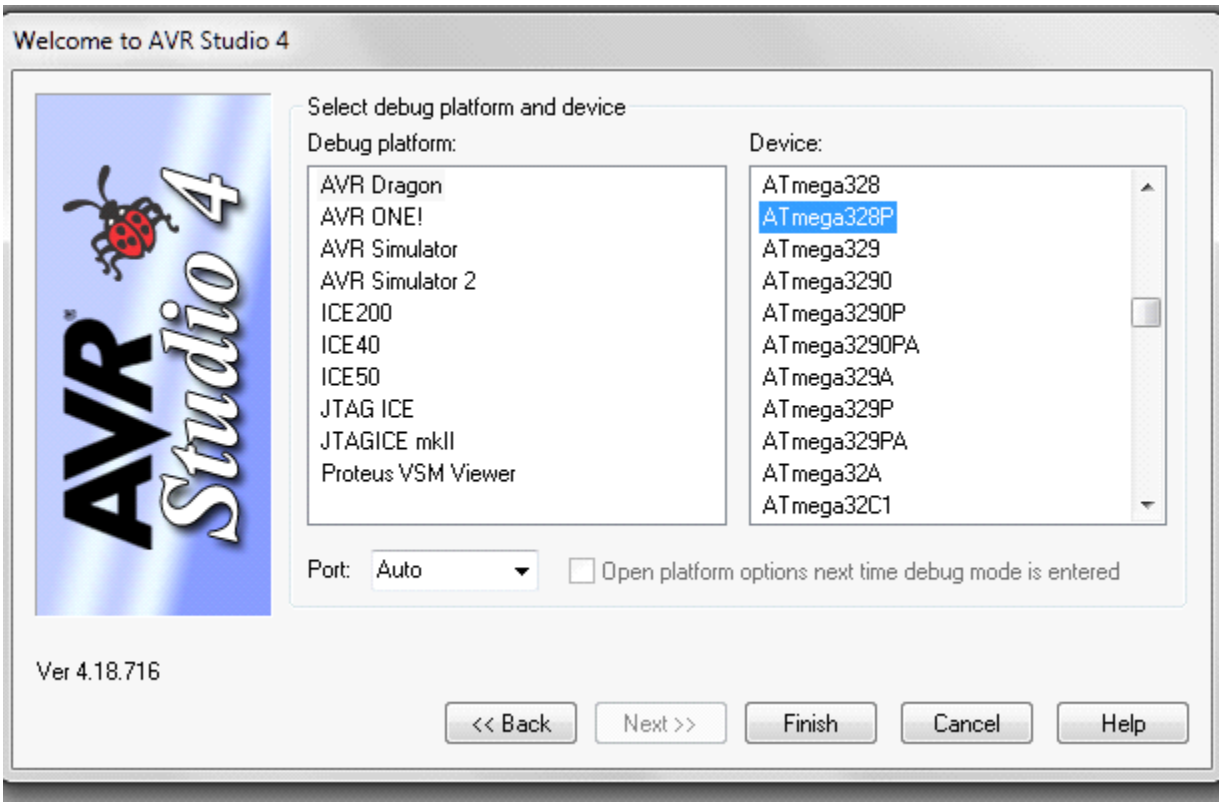
http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=59453

http://arduino.cc/forum/index.php/topic,62094.0.html (I will refer often refer to this one as the Arduino/Eclipse link)

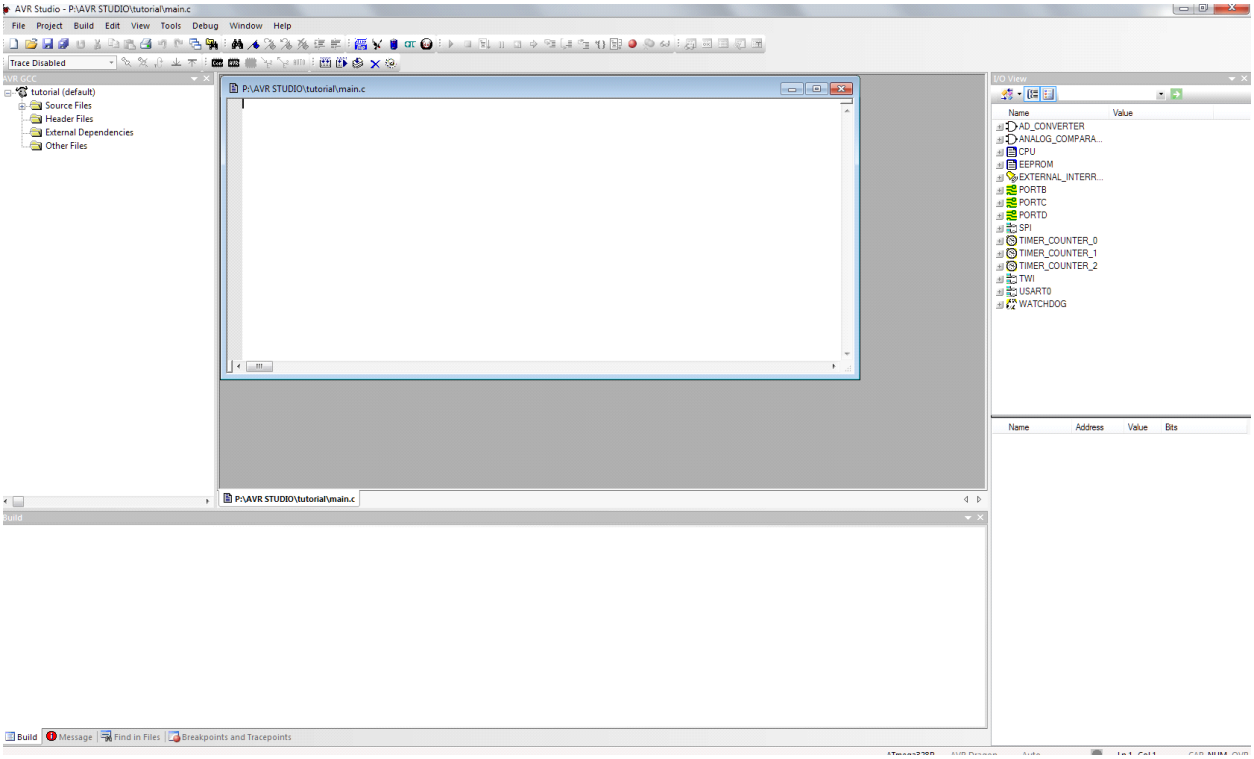
Okay so let's make a simple project based on the Reef Angel development libraries from Curt Binder and the info from the above mentioned links.
Once you have AvrStudio 4 installed and WinAVR. Start AvrStudio and create a new C project using AVR GCC. Enter a project name (no spaces) and name the initial .c file main. I called the project tutorial.




Select the debugger and the Device (328P)



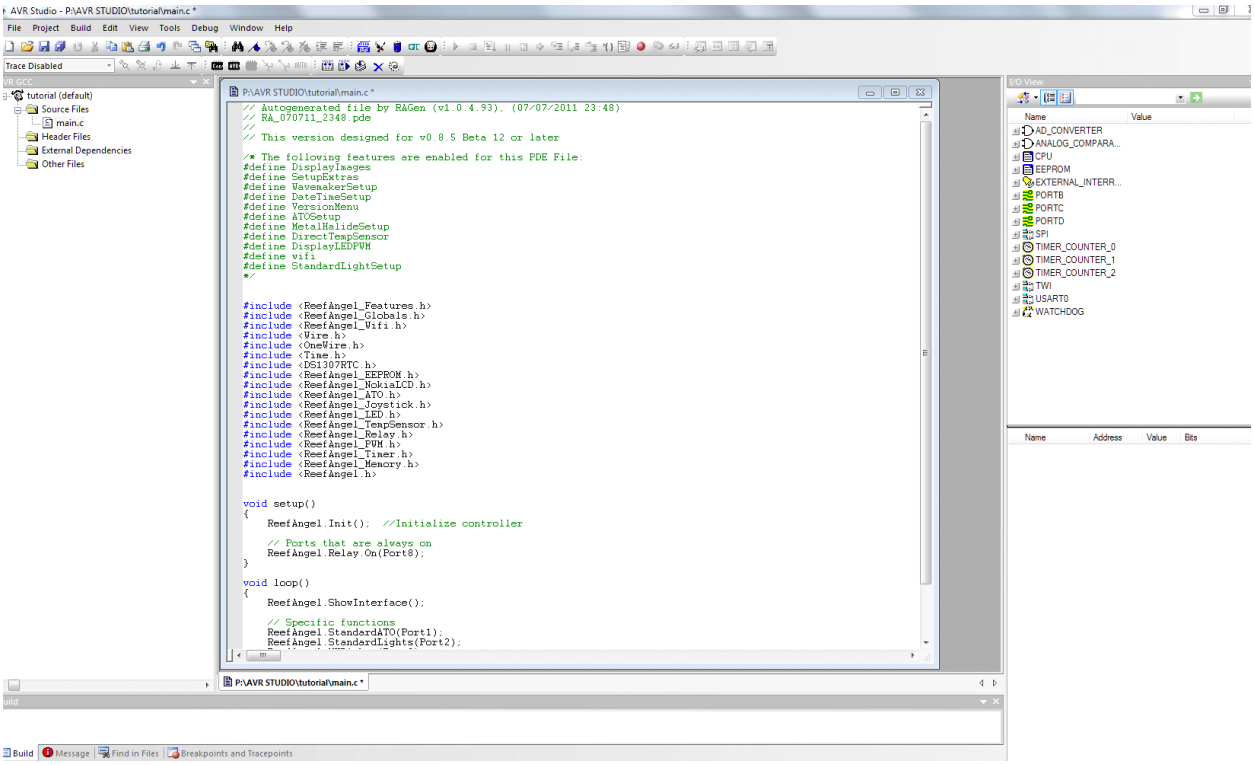
Okay, now we have our blank project




Let's create a simple example using some sample code created from the RA Generator. Paste the code below into the main.c file.

```
// Autogenerated file by RAGen (v1.0.4.93), (07/07/2011 23:48)
// RA_070711_2348.pde
//
// This version designed for v0.8.5 Beta 12 or later

/* The following features are enabled for this PDE File:
#define DisplayImages
#define SetupExtras
#define WavemakerSetup
#define DateTimeSetup
#define VersionMenu
#define ATOSetup
#define MetalHalideSetup
#define DirectTempSensor
#define DisplayLEDPWM
```

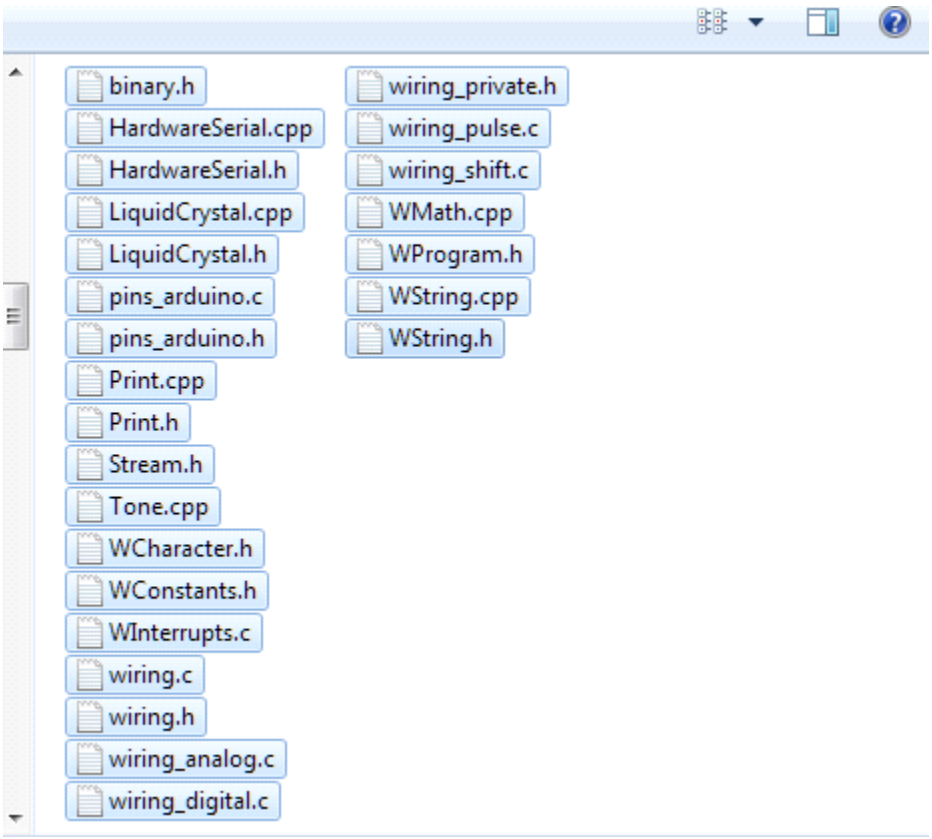
```
#define wifi
#define StandardLightSetup
*/


#include <ReefAngel_Features.h>
#include <ReefAngel_Globals.h>
#include <ReefAngel_Wifi.h>
#include <Wire.h>
#include <OneWire.h>
#include <Time.h>
#include <DS1307RTC.h>
#include <ReefAngel_EEPROM.h>
#include <ReefAngel_NokiaLCD.h>
#include <ReefAngel_ATO.h>
#include <ReefAngel_Joystick.h>
#include <ReefAngel_LED.h>
#include <ReefAngel_TempSensor.h>
#include <ReefAngel_Relay.h>
#include <ReefAngel_PWM.h>
#include <ReefAngel_Timer.h>
#include <ReefAngel_Memory.h>
#include <ReefAngel.h>


void setup()
{
  ReefAngel.Init();  //Initialize controller

  // Ports that are always on
  ReefAngel.Relay.On(Port8);
}

void loop()
{
  ReefAngel.ShowInterface();

  // Specific functions
  ReefAngel.StandardATO(Port1);
  ReefAngel.StandardLights(Port2);
  ReefAngel.MHLights(Port3);
  ReefAngel.Wavemaker1(Port4);
  ReefAngel.Wavemaker2(Port5);
  ReefAngel.StandardFan(Port6);
  ReefAngel.StandardHeater(Port7);
}
```



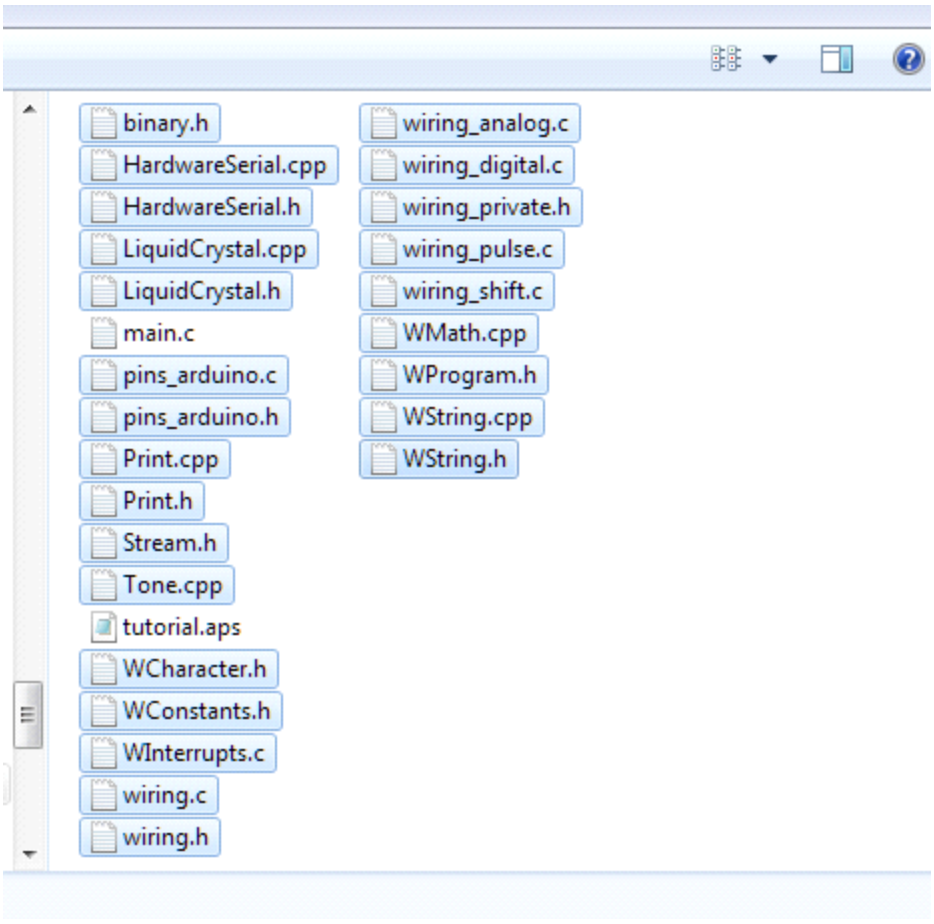

Don't bother compiling or building as it will fail, we still have lot's more to do

Now let’s add the necessary files needed into our project. Rather than modify the existing Arduino files and folders let's just copy the file we need into our project. So let's select the following files below from our Arduino directory. These are the core files and they are all necessary and form the basis for any Arduino project.

..\arduino-0022\arduino-0022\hardware\arduino\cores\arduino




Copy them into our Tutorial project folder. You will see that there are already 2 existing files already present, the main.c file we created earlier and the AvrStudio file config file.

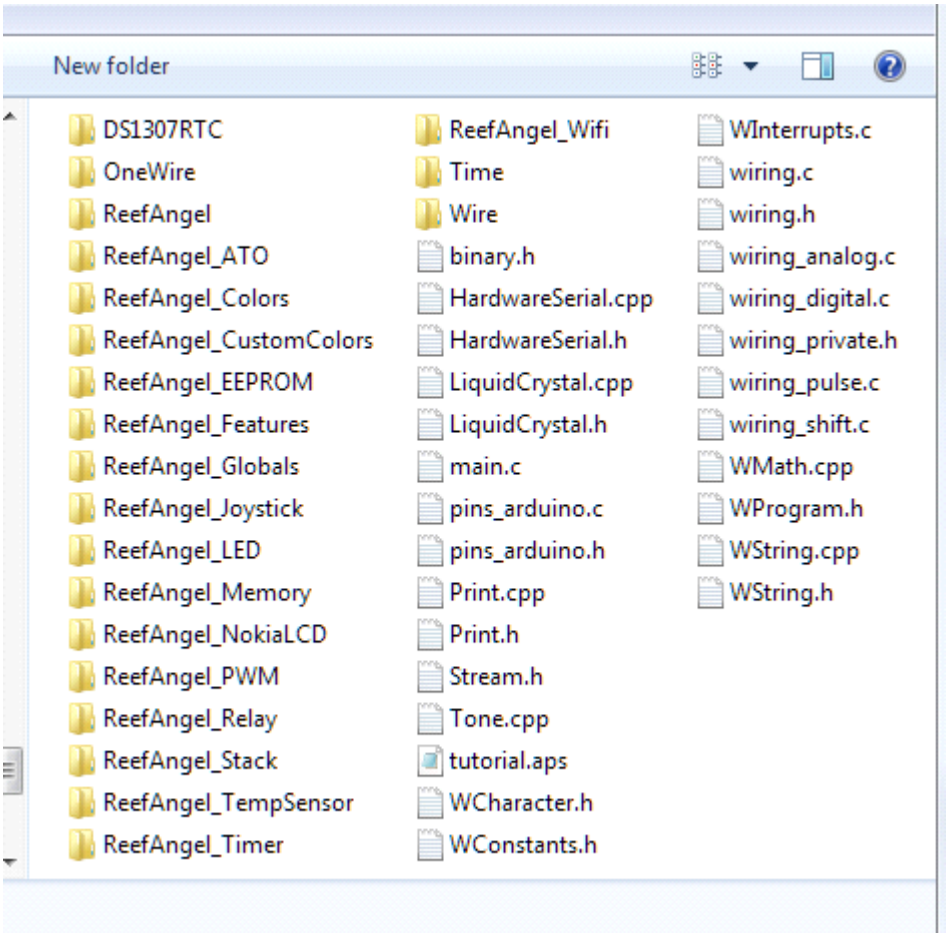



Now let's copy the required Reef Angel libraries for our project, not all are required for this specific project but we will copy them anyways. For simplicity sake we will just copy them into the same project folder. I suppose you could create a sub directory and call it My Libraries and copy them there. Make sure you have the latest development libraries, I believe at this time it is 0.8.15

These are the required libraries for our project

```
#include <ReefAngel_Features.h>
#include <ReefAngel_Globals.h>
#include <ReefAngel_Wifi.h>
#include <Wire.h>
#include <OneWire.h>
#include <Time.h>
#include <DS1307RTC.h>
#include <ReefAngel_EEPROM.h>
#include <ReefAngel_NokiaLCD.h>
#include <ReefAngel_ATO.h>
#include <ReefAngel_Joystick.h>
#include <ReefAngel_LED.h>
#include <ReefAngel_TempSensor.h>
#include <ReefAngel_Relay.h>
#include <ReefAngel_PWM.h>
#include <ReefAngel_Timer.h>
#include <ReefAngel_Memory.h>
#include <ReefAngel.h>
```

So let's copy them into our project folder

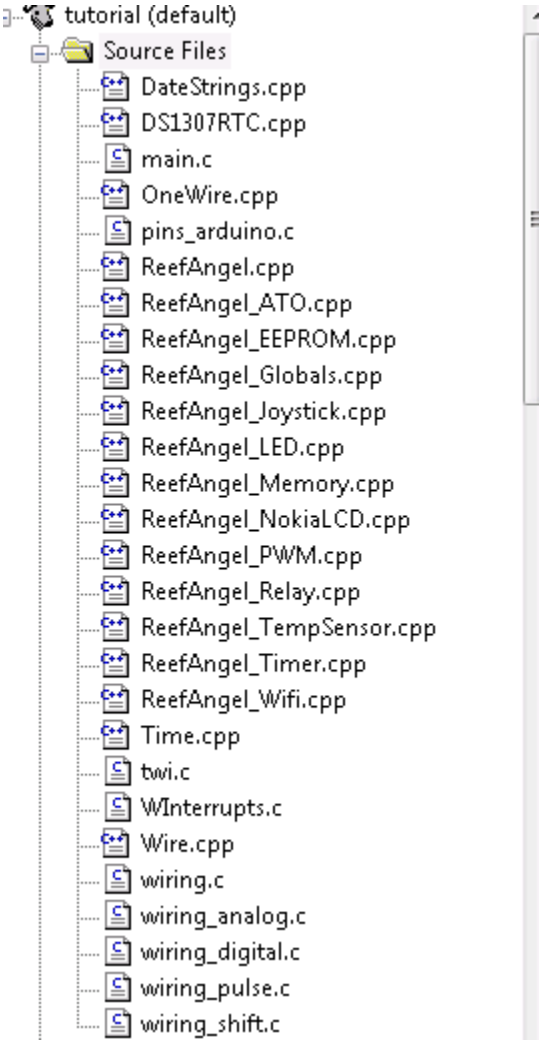


Let's now add all the required *.c source files for the project. Note that some of them have the C++ extension of .cpp. So select all files in the filter to display them, as the default filter searches for just *.c.

Right click the Source folder in the Studio project and select --> Add Existing Files(s). As well I select the Show Files path Option.

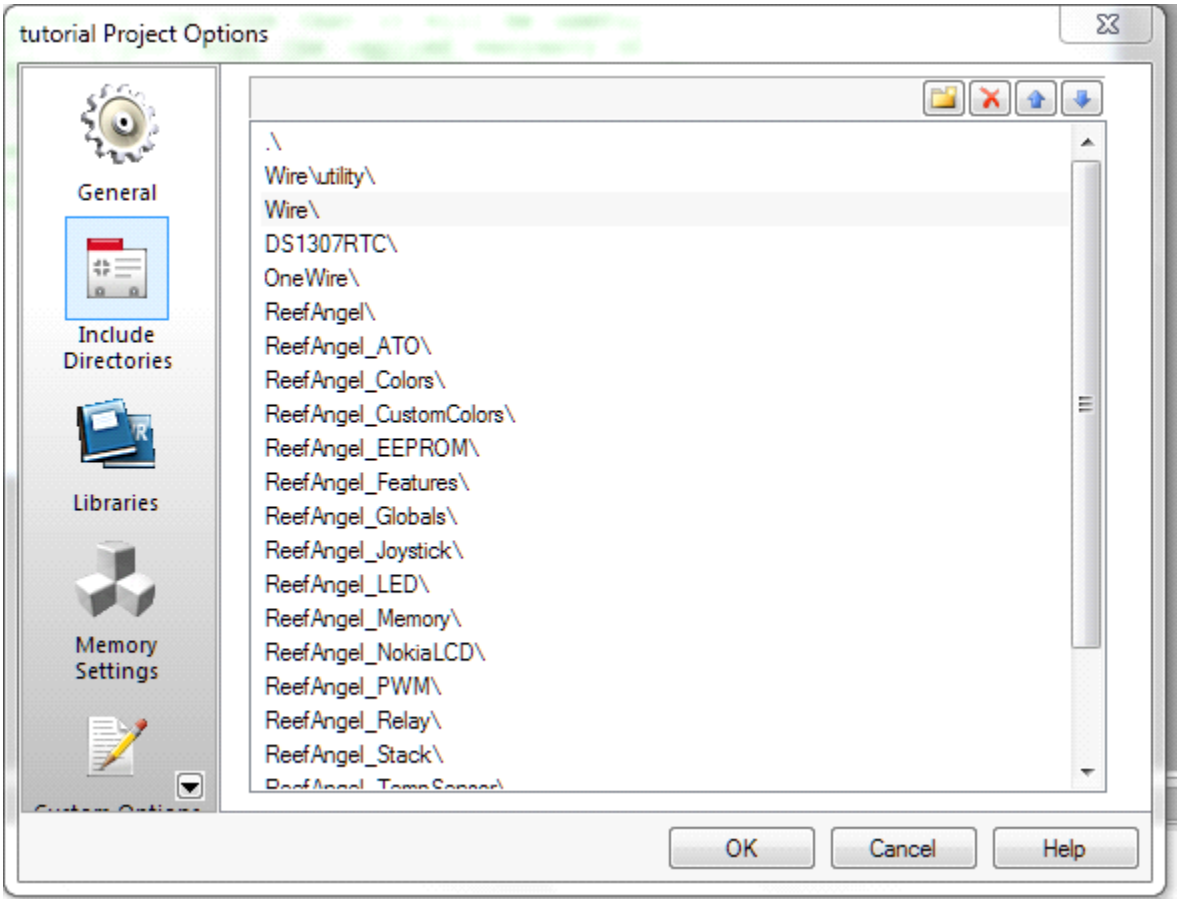
*Note you need to add the twi.c file that is in a subdirectory called utility of the folder Wire. It's easy to forget.

This is what you should have listed for the source file directory

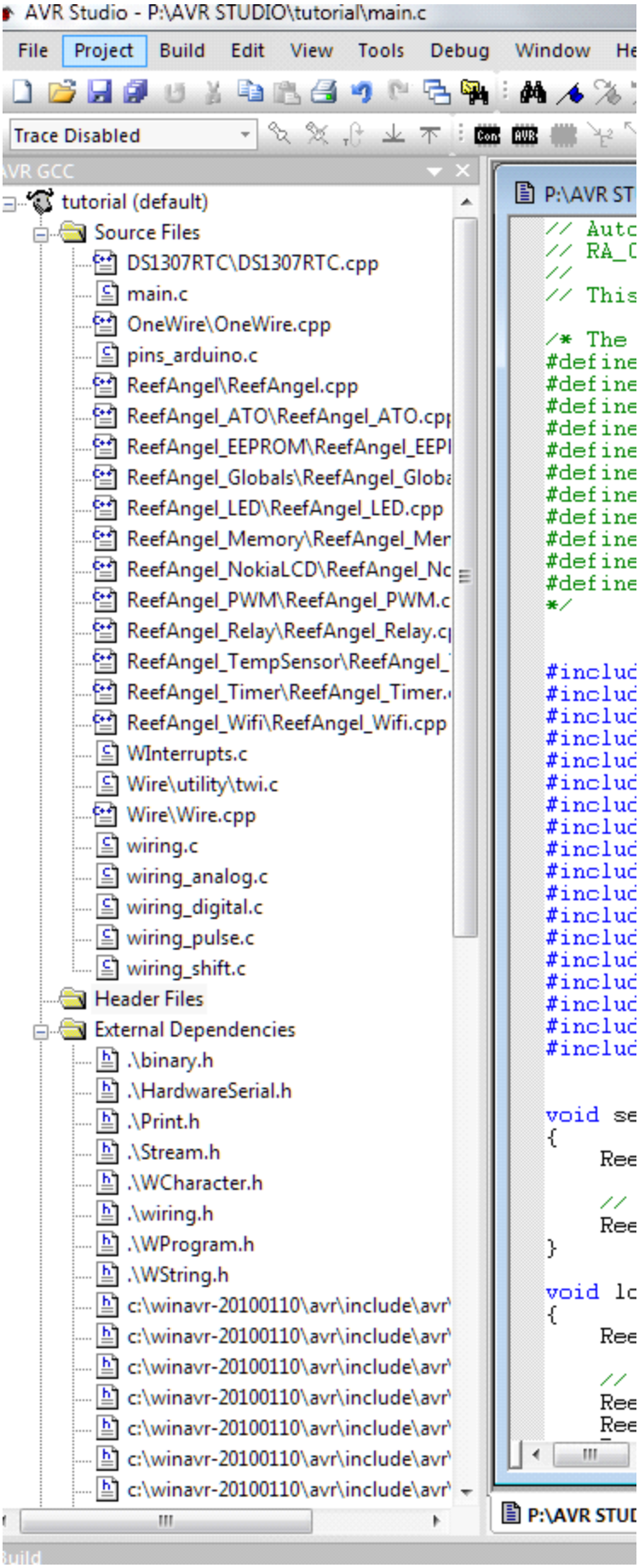



Now we could add the header files for the project in a similar way, but I prefer to just add the paths in the Include Directories Option. Maybe someone can tell me which way is better?

Project-->Configuration Options-->Include Directories. Add the folders where all the libraries are. **Don't forget to add the root folder and make sure you add the sub folder\ utility from the Wire folder.**




So this is what are project should look like at this point, you may have to select compile or double click the node External Dependencies for the files to display.


AVR Studio - P:\AVR STUDIO\tutorial\main.c
File
Project
Build
Edit
View
Tools
Debug
Window
Trace Disabled
tutorial (default)
Source Files
DS1307RTC\DS1307RTC.cpp
main.c
OneWire\OneWire.cpp
pins_arduino.c
ReefAngel\ReefAngel.cpp
ReefAngel_ATO\ReefAngel_ATO.cpp
ReefAngel_EEPROM\ReefAngel_EEPI
ReefAngel_Globals\ReefAngel_Globa
ReefAngel_LED\ReefAngel_LED.cpp
ReefAngel_Memory\ReefAngel_Mer
ReefAngel_NokiaLCD\ReefAngel_Nc
ReefAngel_PWM\ReefAngel_PWM.c
ReefAngel_Relay\ReefAngel_Relay.c
ReefAngel_TempSensor\ReefAngel_
ReefAngel_Timer\ReefAngel_Timer.
ReefAngel_Wifi\ReefAngel_Wifi.cpp
WInterrupts.c
Wire\utility\twi.c
Wire\Wire.cpp
wiring.c
wiring_analog.c
wiring_digital.c
wiring_pulse.c
wiring_shift.c
Header Files
External Dependencies
.\binary.h
.\HardwareSerial.h
.\Print.h
.\Stream.h
.\WCharacter.h
.\wiring.h
.\WProgram.h
.\WString.h
c:\winavr-20100110\avr\include\avr
P:\AVR ST
// Autc
// RA_C
//
// This
/* The
#define
#define
#define
#define
#define
#define
#define
#define
#define
#define
*/
#includ
#includ
#includ
#includ
#includ
#includ
#includ
#includ
#includ
#includ
#includ
#includ
#includ
#includ
#includ
#includ
#includ
void se
{
Ree
//
Ree
}
void lc
{
Ree
//
Ree
Ree
P:\AVR STUD

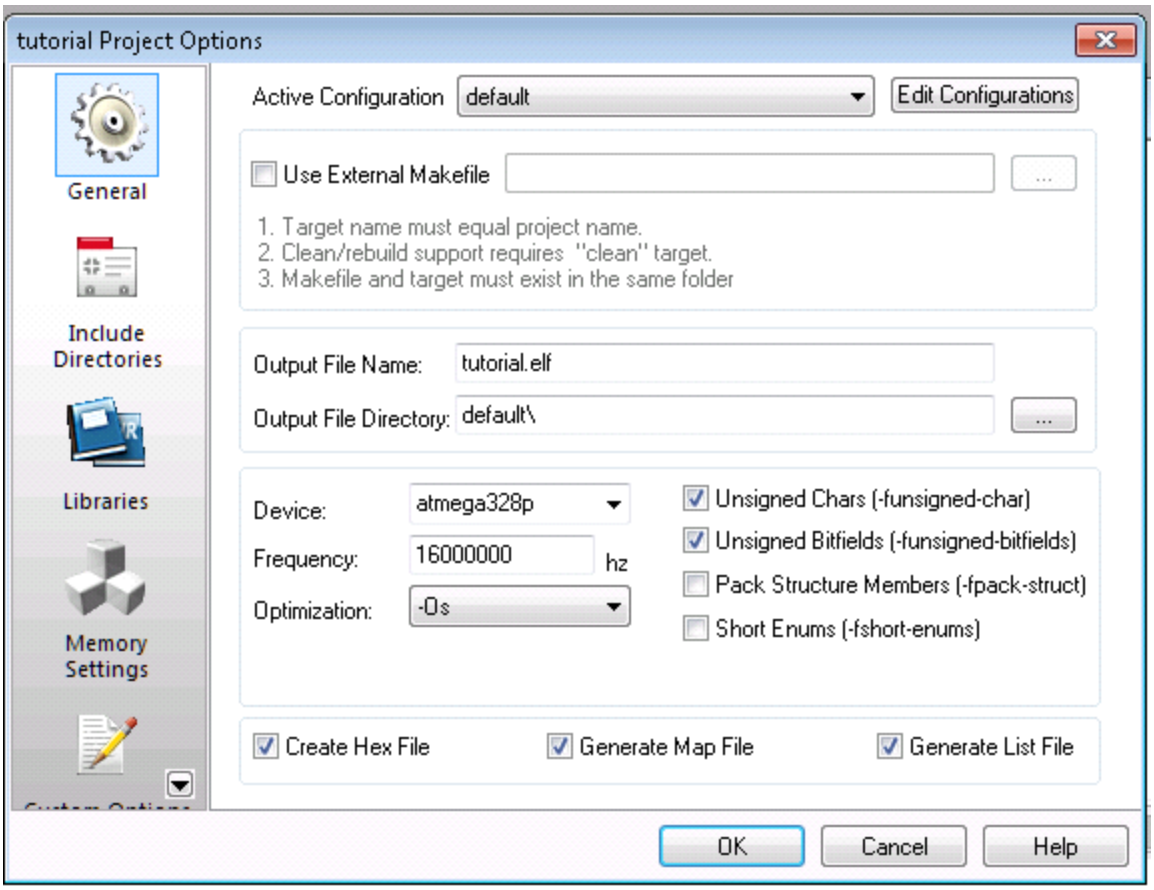
There is still one more file we need to add and we have to create it first. Quoting from the http://arduino.cc/forum/index.php/topic,62094.0.html link

Importing the Core Arduino Library

*4.All projects using the Arduino environment need the precompiled Arduino library, if you aren't compiling together with your own custom core. Open an example project in the Arduino IDE and "verify" it. Now in your Eclipse project, import the library from the filesystem, which is probably in /tmp/buildXXXXXXXXXXXXXXXXXXXXX.tmp/core.a. Put it in a new subfolder "arduino" within the project, and rename it* ***libcore.a.***

Copy this file to the project folder or in a sub folder.

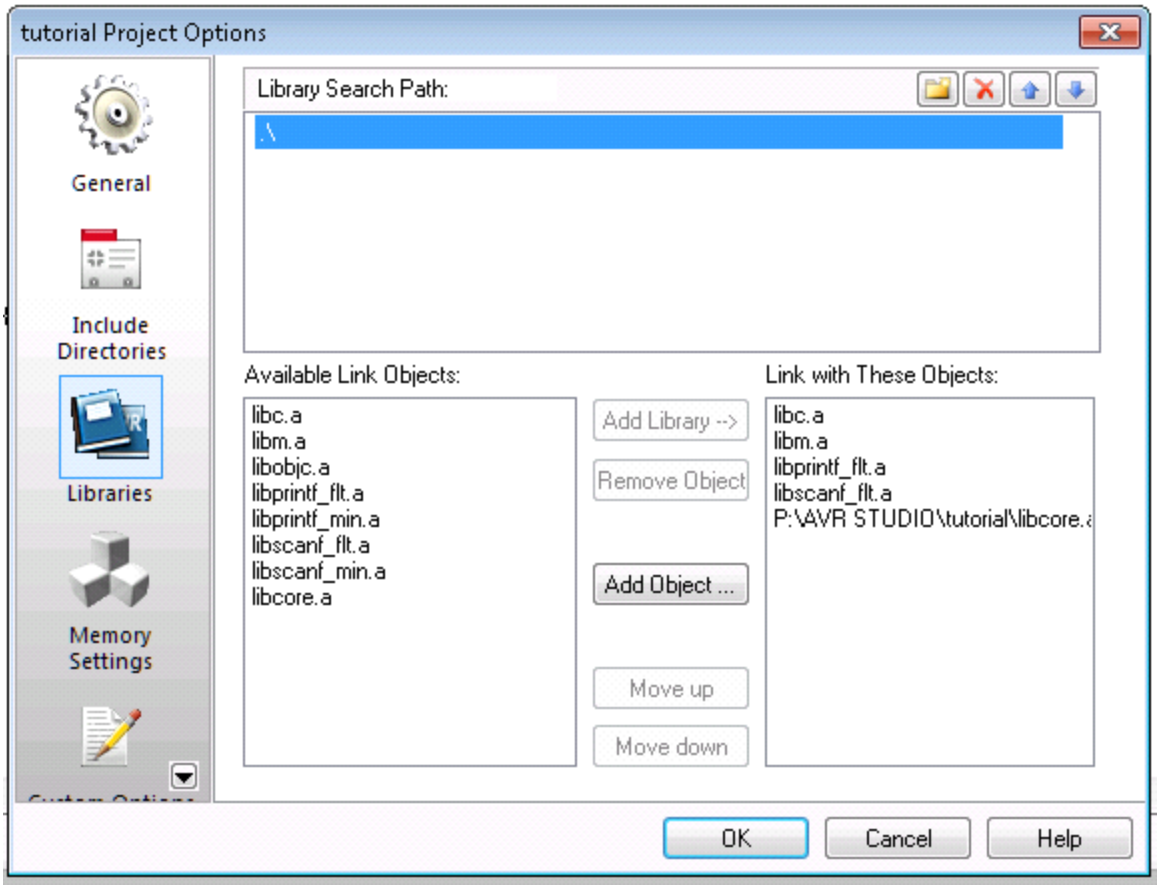
Okay, so know let's start configuring the project. From Project-->Configuration Options -General, add the Frequency and unselect Pack Structure and Short Enemas. A good primer for what these settings do is also outlined in the Arduino/Eclipse link from above. Make sure the Optimization is set to -Os




We have already modified our Include Directories, so we can skip to the 3rd box "Libraries". We need to add some core libraries as well as the custom Arduino library. Add c,m,printf_flt,scanf_flt using the **ADD LIBRARY** button.

Add the core library (libcore.a) file we copied earlier using the **ADD OBJECT** button, select the folder and object file libcore.a. Don't forget to add the Library Search path to the same path as the libcore.a file.

*Libraries*
*Libraries: add c, m, printf_flt, scanf_flt, and core.*
*These are the C core library, the math library, printf and scanf libraries with float support (the default versions don't have it), and the Arduino core library. Not all are necessary for all projects, but they are frequently needed and it's easy to add or remove them if you want.*



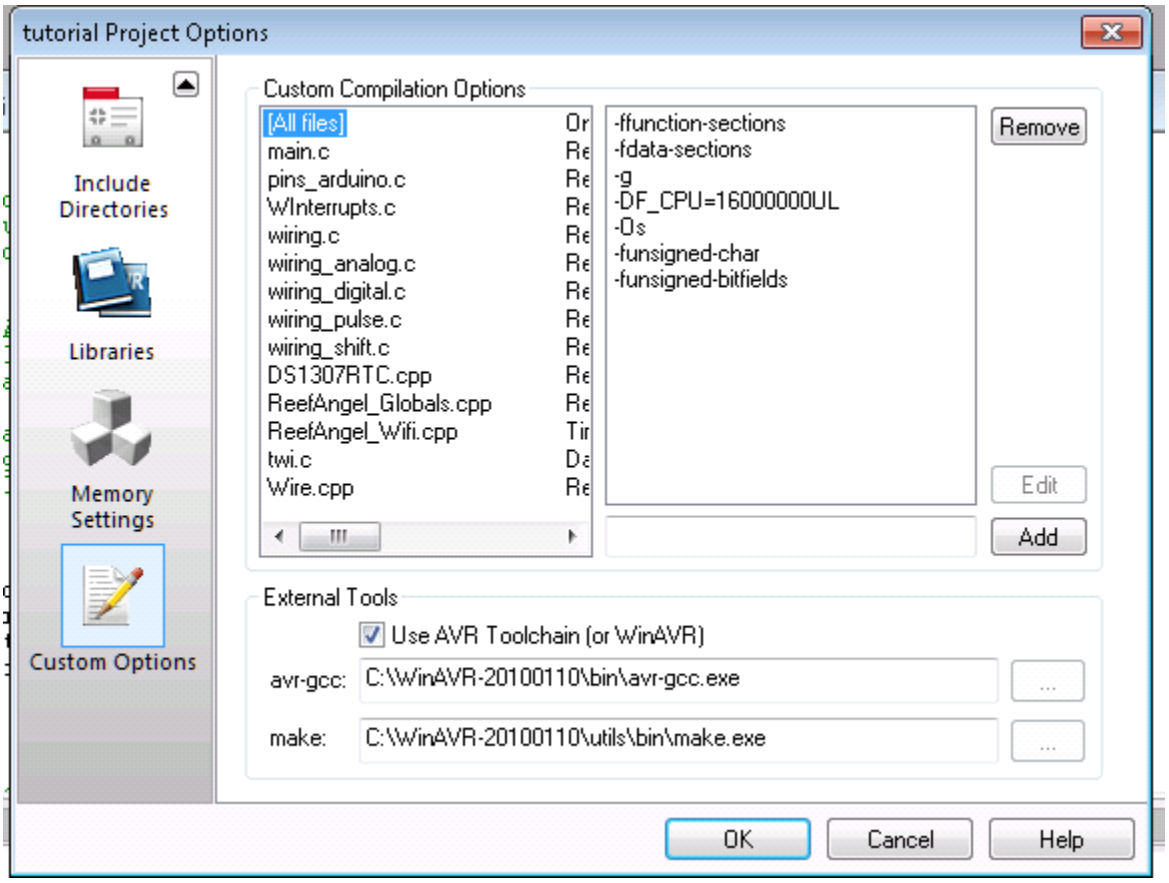

We do not need to do anything with the Memory Settings, or maybe we do but I don't know anything about these settings, so let's move on to the Custom Options. A lot of the compiler/build settings are explained in the Arduino/Eclipse post so refer to that for additional information.

For now we will leave the AVR tool chain as is, it can be pointed to other tool chains, but I found it works fine as is. I assume the compiler is smart enough to recognize it is ++ code.

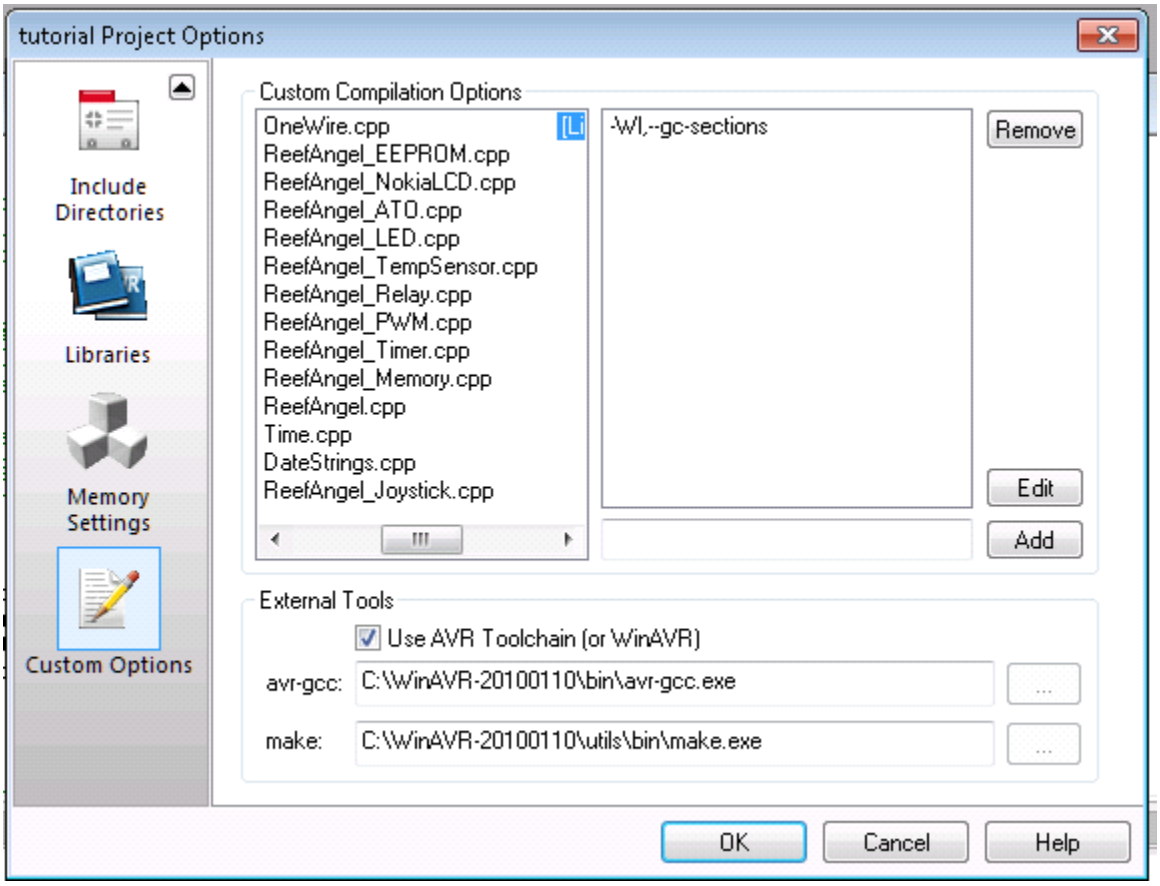
I tried using the Arduino tool chain..\arduino-0022\arduino-0022\hardware\tools\avr\bin which should be the same, but it would not work. Suspect it may because I have spaces in the path.

Make sure your mouse has highlighted [All files] and create the same settings as below for the Custom

Compilation Options




Alright now, highlight the [ Linker Options] and add/modify the settings as below. Once again these options are described in the Arduino/Eclipse link. The -g flag as above copies some of the settings over to the linker, so we don't need to duplicate




Now we need to add a custom header file. This is also explained in the Arduino/Eclipse link

*6. Template Code Setup*
*The Arduino IDE does a little bit of source mangling before compilation which we need to replicate. Add this code to a header file like cppsupport.h and #include it in your main.cpp source file, or have it inline. It provides some C++ style wrappers for malloc and the guards protect against accessing invalid memory locations. There is also a handler to catch calls to purely virtual functions.*

So, right click the "header files" folder and "create a new header file". Call it anything you want but for consistency sake let's call it cppsupport.h

Add the following code to the file and save. Not all is needed as mentioned in the Arduino/Eclipse link because some of the code is already included in reefangel_globals.h/cpp

```
#include <stdlib.h>
__extension__ typedef int __guard __attribute__((mode (__DI__)));
void * operator new(size_t size)
{
  return malloc(size);
```

```
}
void operator delete(void * ptr)
{
  free(ptr);
}
void * operator new[](size_t size)
{
    return malloc(size);
}
void operator delete[](void * ptr)
{
  if (ptr)
     free(ptr);
}
```

Almost there, so now let's move on to modifying the Main.c file. We need to add 2 additional headers to the project and they are wprogram.h and the new header we just created called cppsupport.h.

```
#include <ReefAngel_Features.h>
#include <ReefAngel_Globals.h>
#include <ReefAngel_Wifi.h>
#include <Wire.h>
#include <OneWire.h>
#include <Time.h>
#include <DS1307RTC.h>
#include <ReefAngel_EEPROM.h>
#include <ReefAngel_NokiaLCD.h>
#include <ReefAngel_ATO.h>
#include <ReefAngel_Joystick.h>
#include <ReefAngel_LED.h>
#include <ReefAngel_TempSensor.h>
#include <ReefAngel_Relay.h>
#include <ReefAngel_PWM.h>
#include <ReefAngel_Timer.h>
#include <ReefAngel_Memory.h>
#include <ReefAngel.h>

#include <wprogram.h>
#include <cppsupport.h>
```

Every C or C++ file needs a main function. Arduino does it behind the scenes so we need to add one and add a call to the init(); function.

```
void setup()
{
    ReefAngel.Init();  //Initialize controller

    // Ports that are always on
    ReefAngel.Relay.On(Port8);
}

void loop()
{
    ReefAngel.ShowInterface();

    // Specific functions
    ReefAngel.StandardATO(Port1);
    ReefAngel.StandardLights(Port2);
    ReefAngel.MHLights(Port3);
    ReefAngel.Wavemaker1(Port4);
    ReefAngel.Wavemaker2(Port5);
    ReefAngel.StandardFan(Port6);
    ReefAngel.StandardHeater(Port7);
}



int main(void)
{
   init();

   setup();

   for (;;)
   {
      loop();
   }
   return 0;
}
```

Alright so let's build our project, oops we choked with some errors.

```
Build
../pins_arduino.c:372: error: invalid conversion from 'volatile uint8_t*' to 'uint16_t'
../pins_arduino.c:372: error: invalid conversion from 'volatile uint8_t*' to 'uint16_t'
../pins_arduino.c:372: error: invalid conversion from 'volatile uint8_t*' to 'uint16_t'
../pins_arduino.c:374: warning: only initialized variables can be placed into program memory area
../pins_arduino.c:380: error: invalid conversion from 'volatile uint8_t*' to 'uint16_t'
../pins_arduino.c:380: error: invalid conversion from 'volatile uint8_t*' to 'uint16_t'
../pins_arduino.c:380: error: invalid conversion from 'volatile uint8_t*' to 'uint16_t'
../pins_arduino.c:382: warning: only initialized variables can be placed into program memory area
../pins_arduino.c:405: warning: only initialized variables can be placed into program memory area
../pins_arduino.c:428: warning: only initialized variables can be placed into program memory area
make: *** [pins_arduino.o] Error 1
Build failed with 9 errors and 21 warnings...
```

We need to fix this, and this is not the most elegant solution but it works. Open up the pins_arduino.c file and modify the following

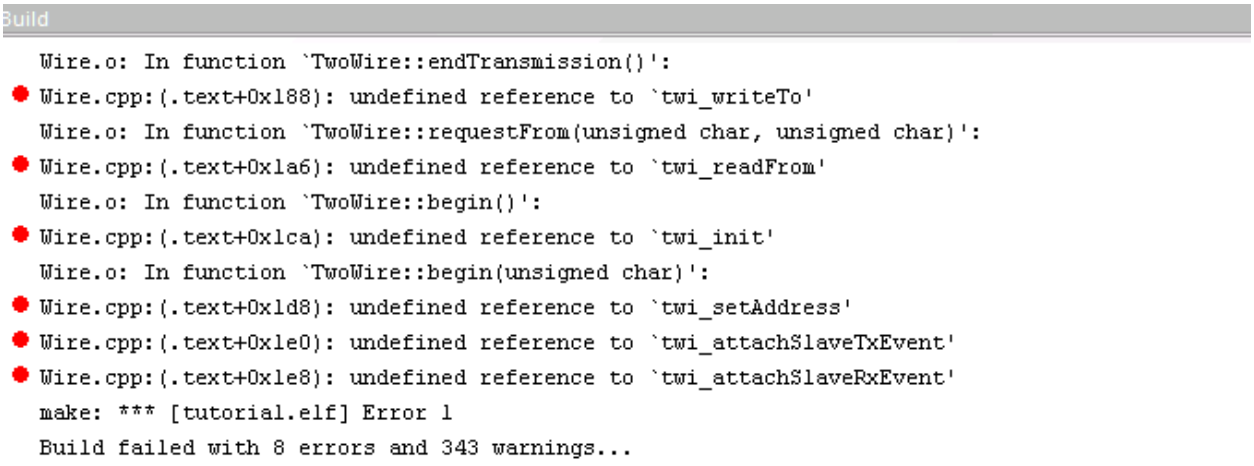
```
const uint16_t PROGMEM port_to_mode_PGM[] = {
	NOT_A_PORT,
	NOT_A_PORT,
	&DDRB,
	&DDRC,
	&DDRD,
};


const uint16_t PROGMEM port_to_output_PGM[] = {
	NOT_A_PORT,
	NOT_A_PORT,
	&PORTB,
	&PORTC,
	&PORTD,
};
const uint16_t PROGMEM port_to_input_PGM[] = {
	NOT_A_PORT,
	NOT_A_PORT,
	&PINB,
	&PINC,
	&PIND,
};
```

to

```
const uint16_t PROGMEM port_to_mode_PGM[] = {
	NOT_A_PORT,
	NOT_A_PORT,
	(uint16_t)(&DDRB),
	(uint16_t)(&DDRC),
	(uint16_t)(&DDRD),
};

const uint16_t PROGMEM port_to_output_PGM[] = {
	NOT_A_PORT,
	NOT_A_PORT,
	(uint16_t)(&PORTB),
	(uint16_t)(&PORTC),
	(uint16_t)(&PORTD),
};

const uint16_t PROGMEM port_to_input_PGM[] = {
```

```
        NOT_A_PORT,
        NOT_A_PORT,
        (uint16_t)(&PINB),
        (uint16_t)(&PINC),
        (uint16_t)(&PIND),
};
```

Darn, some more build errors.




I don't know why but these errors can be easily fixed by opening up the wire.cpp file and modifying the following

```
extern "C" {
  #include <stdlib.h>
  #include <string.h>
  #include <inttypes.h>
  #include "twi.h"
}
```

to

```
//extern "C" {
  #include <stdlib.h>
  #include <string.h>
  #include <inttypes.h>
  #include "twi.h"
//}
```

All right here we go. Success!!!.

```
AVR Memory Usage
----------------
Device: atmega328p

Program:   30512 bytes (93.1% Full)
(.text + .data + .bootloader)

Data:       1839 bytes (89.8% Full)
(.data + .bss + .noinit)


Build succeeded with 343 Warnings...
```

You can also burn your AVR from within AvrStudio by using Tools--Program AVR

Like i said earlier I am no expert, but have managed to cobble it together, I am sure there are some additional steps that may optimize things or make it easier. I would appreciate any input.

You may also want to save this file as a template so you do not need to recreate every time you want to import an Arduino project, whether it's something simple like a Blink Led or complex like the Reef Angel.

One couple notes, you will notice there are a lot of warnings when the project is compiled and built. You can eliminate the warnings by adding the -w flag, this flag inhibits all warnings, but I prefer to see the warnings. Arduino adds this flag in their compiler to hide warnings.

Have fun, and any improvements or suggestions let me know.

-=Bryan=-