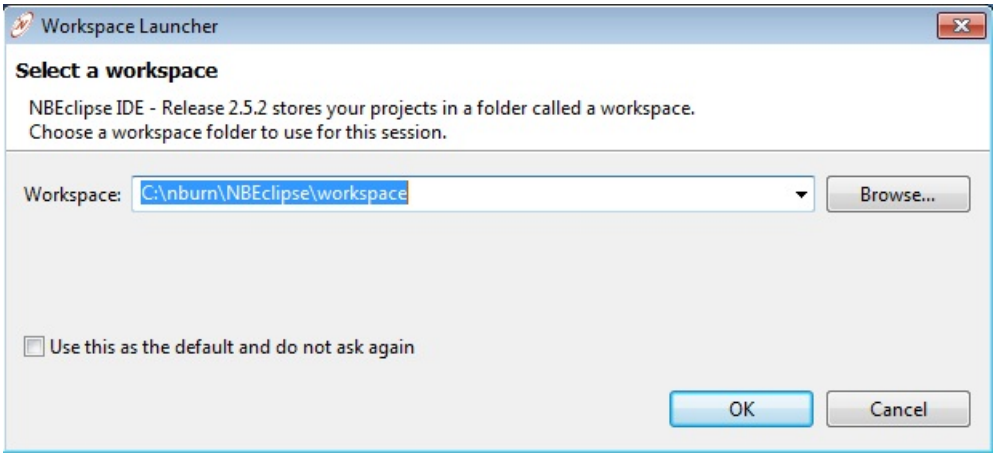
Lab 0: Tutorial - Introduction to Netburner

Starting Netburner Eclipse IDE

Double click the NBEclipse shortcut icon on the desktop or
Start → **All Programs** → **Netburner NNDK** → **NBEclipse** → **NBEclipse**

Default NBEclipse Workspace

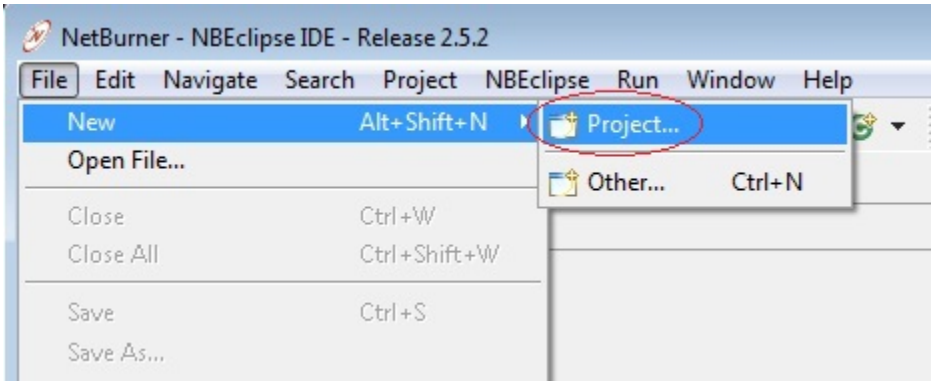
When starting NBEclipse, you are asked to choose a workspace directory where your projects are to be stored. By default all project files will be created and stored in the default "**Workspace**" directory *\Nburn \NBEclipse \workspace*. Do not change this setting.



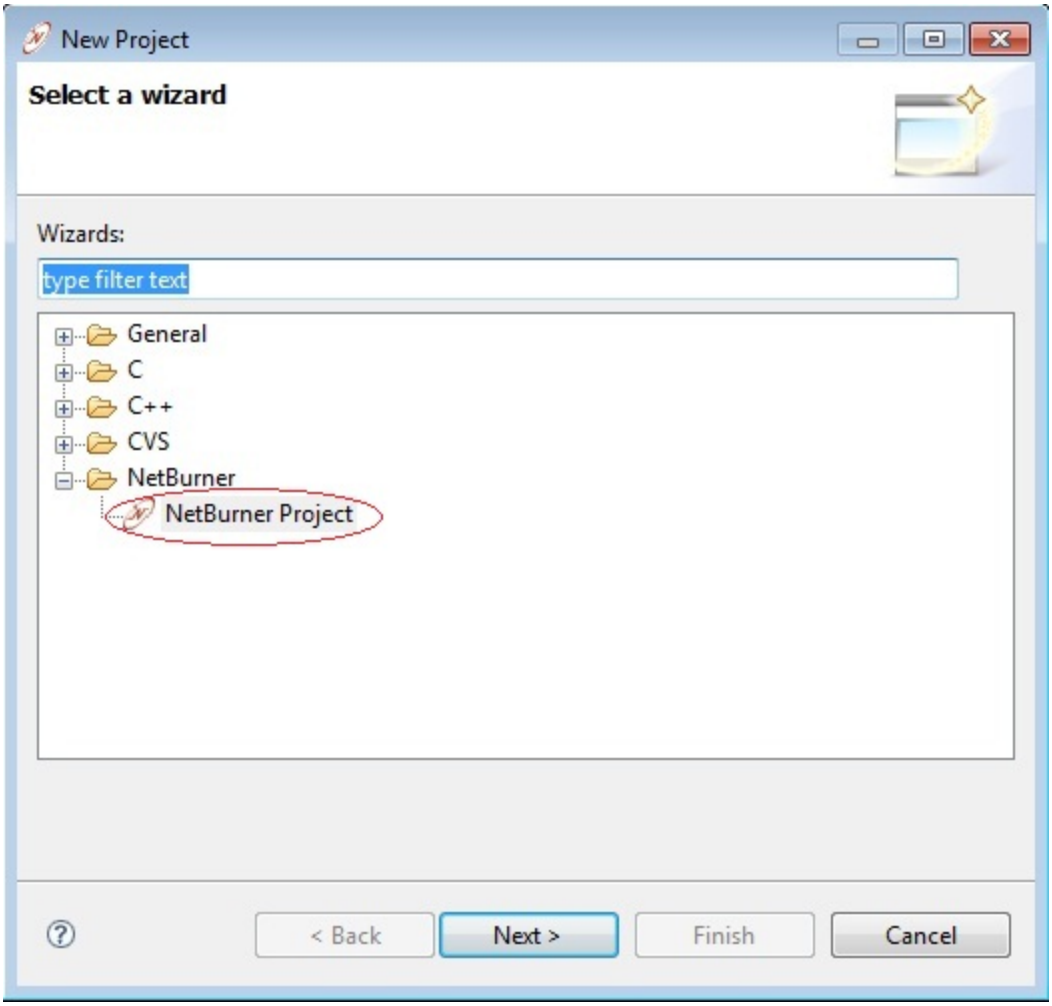
Click "**OK**"

Creating a New Project

From the NBEclipse main menu, select
File → New → Project

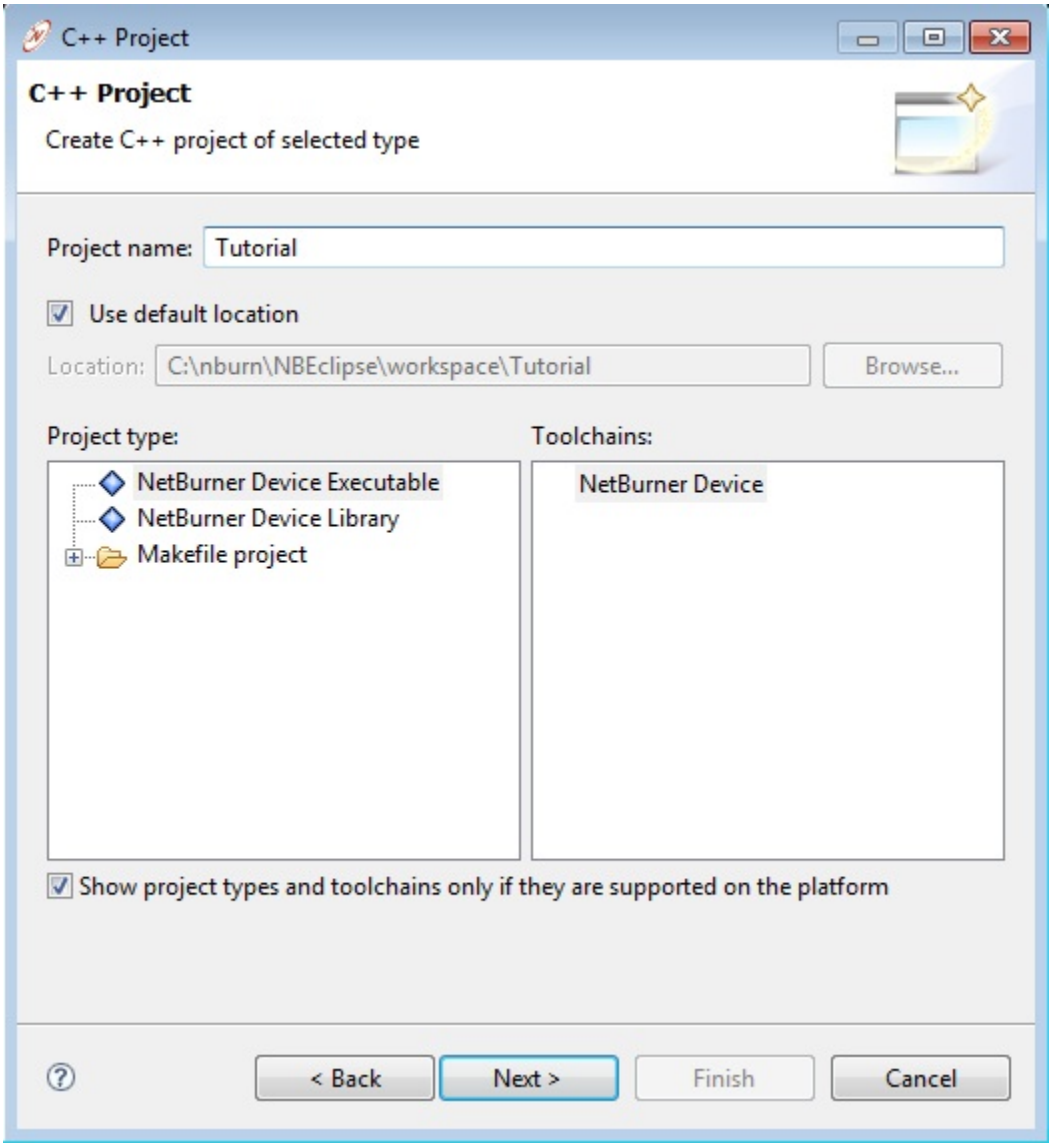


The "**New Project**" dialog box below will appear. Under the NetBurner folder, choose the "**Netburner Project**" option. This option enables NBEclipse to handle all the project build settings.

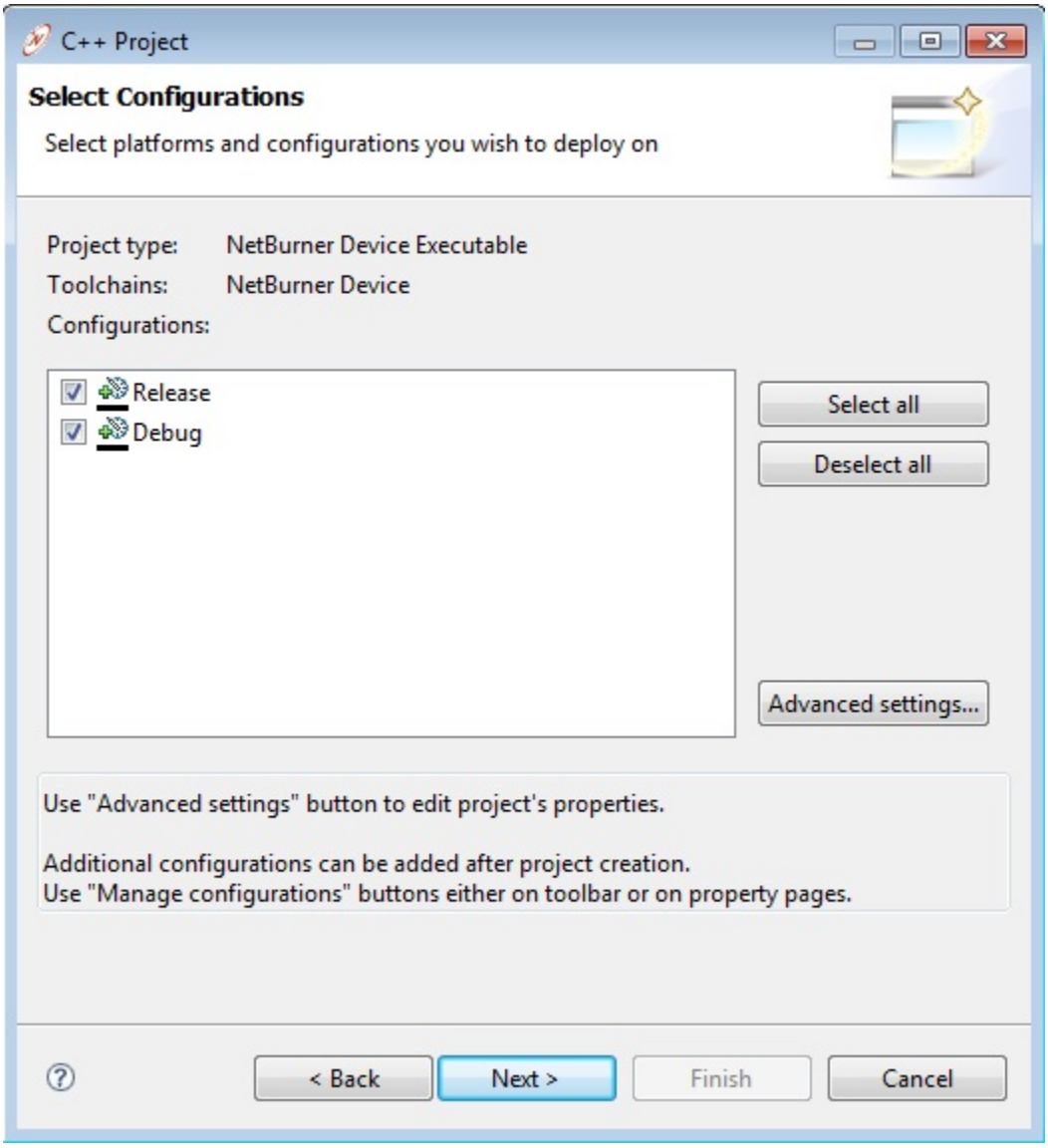


Selecting "**Next**" will bring up the "**C++ Project**" dialog box. Enter the project name as "**Tutorial**". Leave the "**use default location**" checkbox highlighted so that the project will be located in the default workspace. Under project types and toolchains, make sure "**Netburner Device Executable**" and "**Netburner Device**" are selected

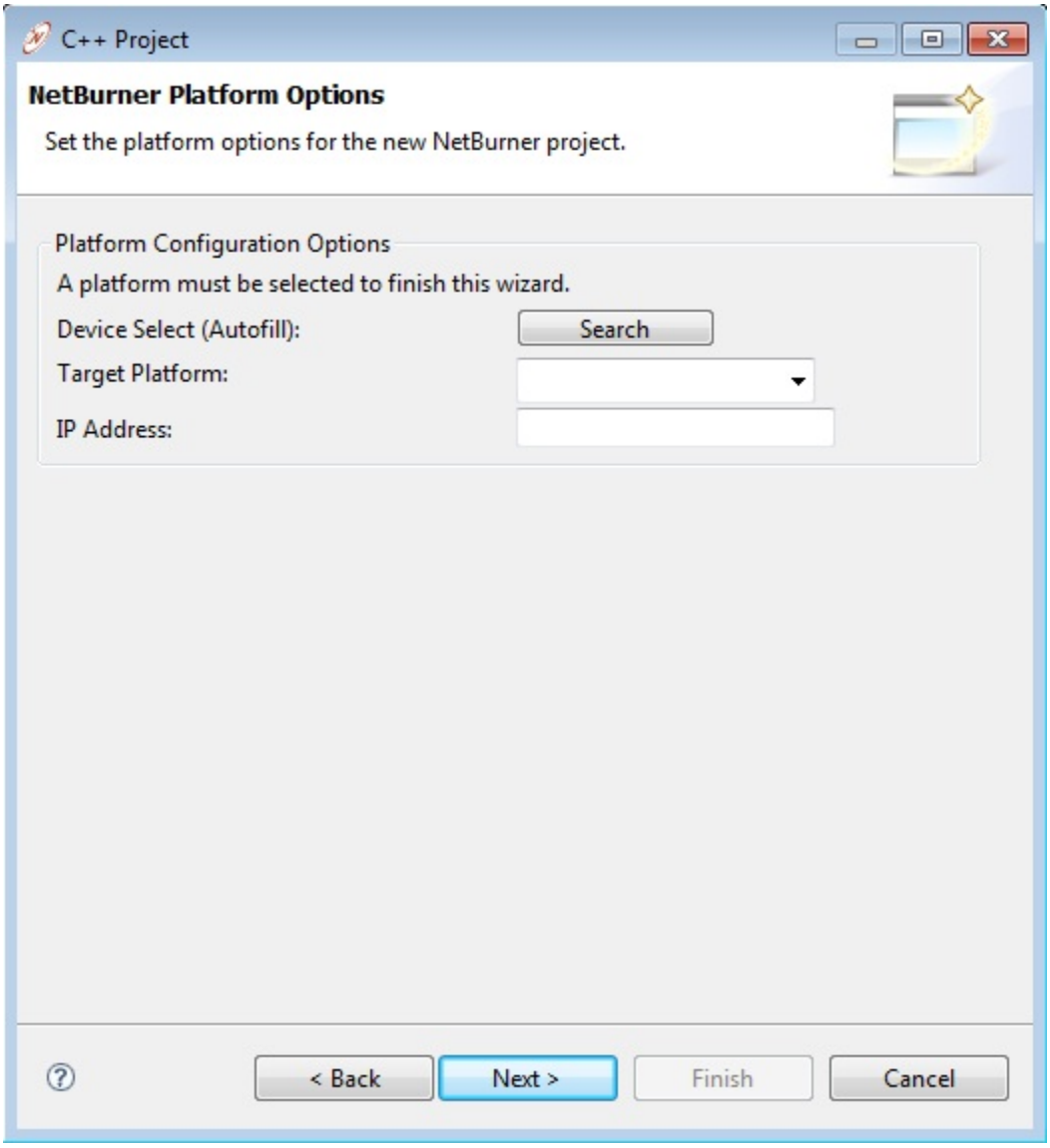
.



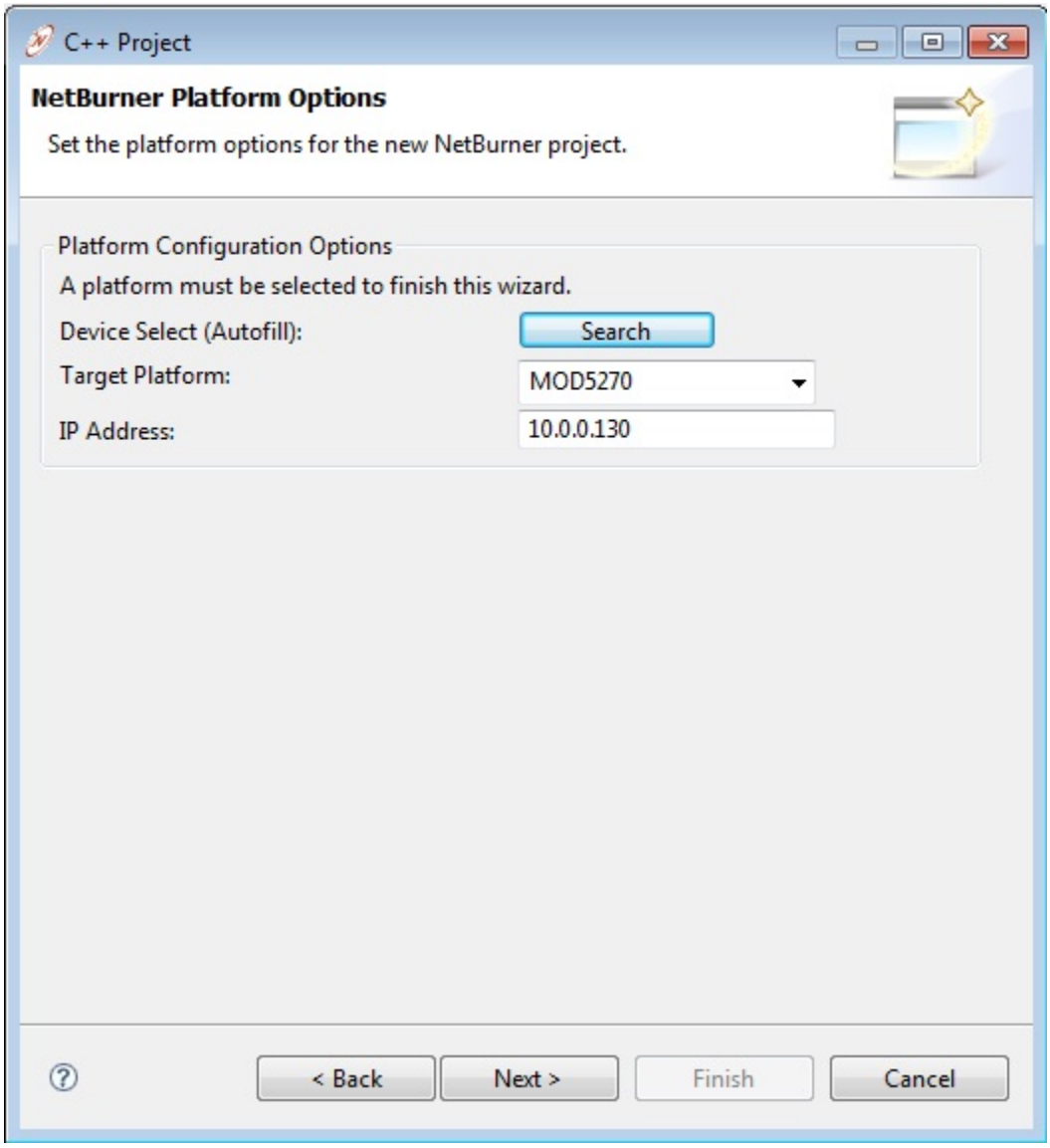
Selecting "**Next**" will display the dialog box to select the platform configurations types. Leave the "**Release**" and "**Debug**" checkbox highlighted so that we will be able to build both the release and debug.



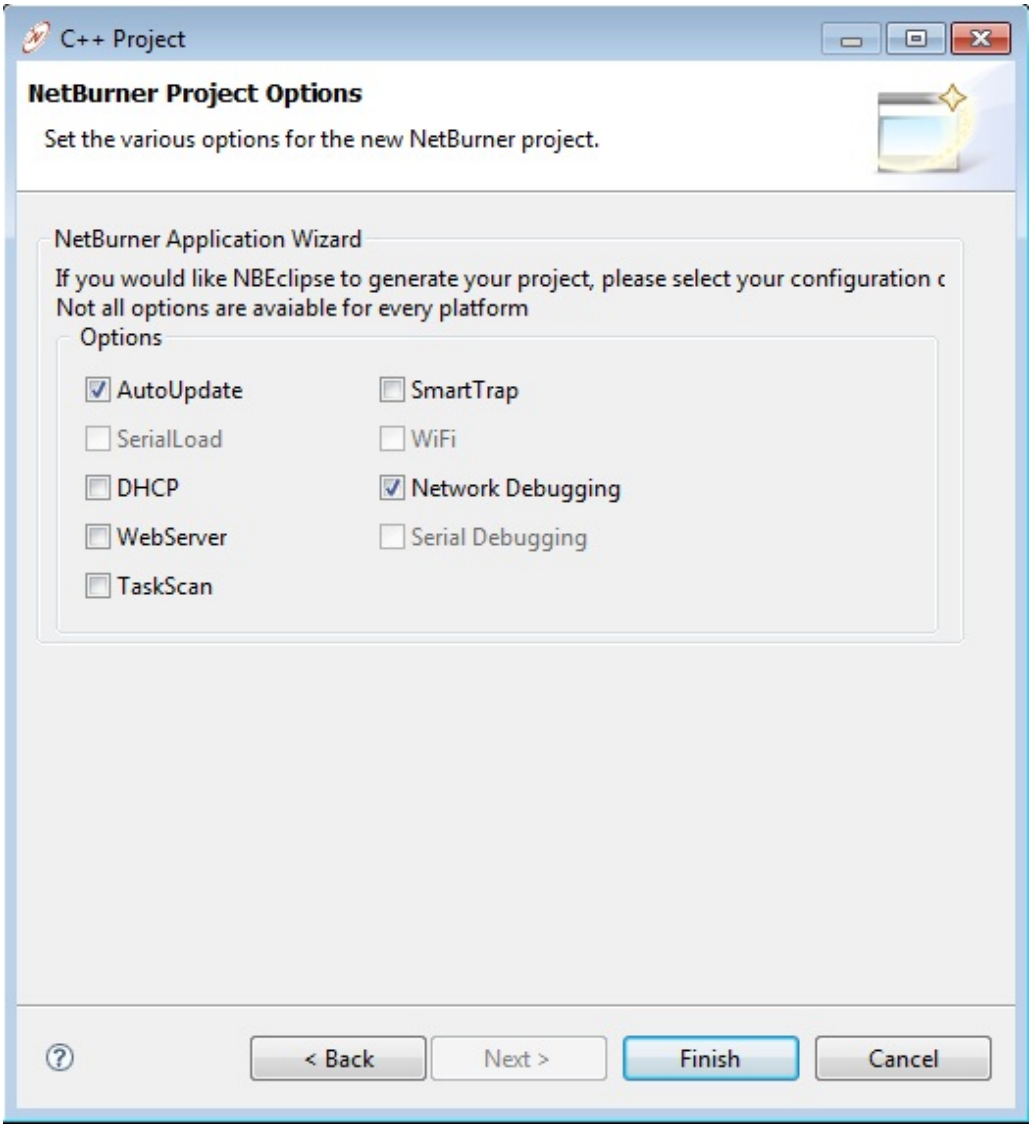
Selecting "**Next**" will display the "**Netburner Platform options**" dialog box. Click the search button to find the Netburner board connected to the computer.



The target platform should be "**Mod5270**" and the IP Address should be filled with a number. Each board in the lab has an unique IP address and may not match the one in the screen shot.

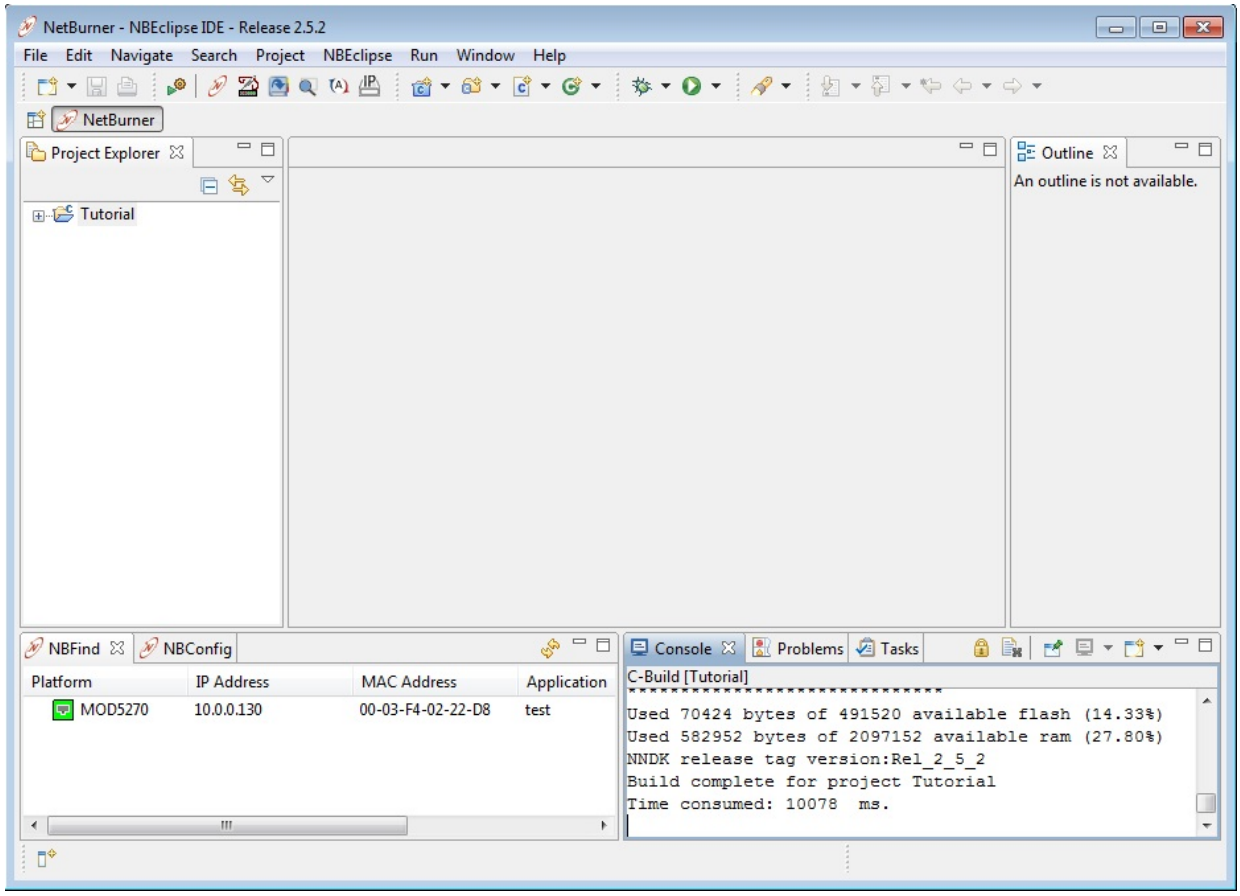


Selecting "**Next**" will display the "**Netburner Project Options**" dialog box. In the "**Netburner Application Wizard**", check off the box for "**AutoUpdate**" and "**Network Debugging**". By selecting these options, we can now perform Network debugging and updates over the network.

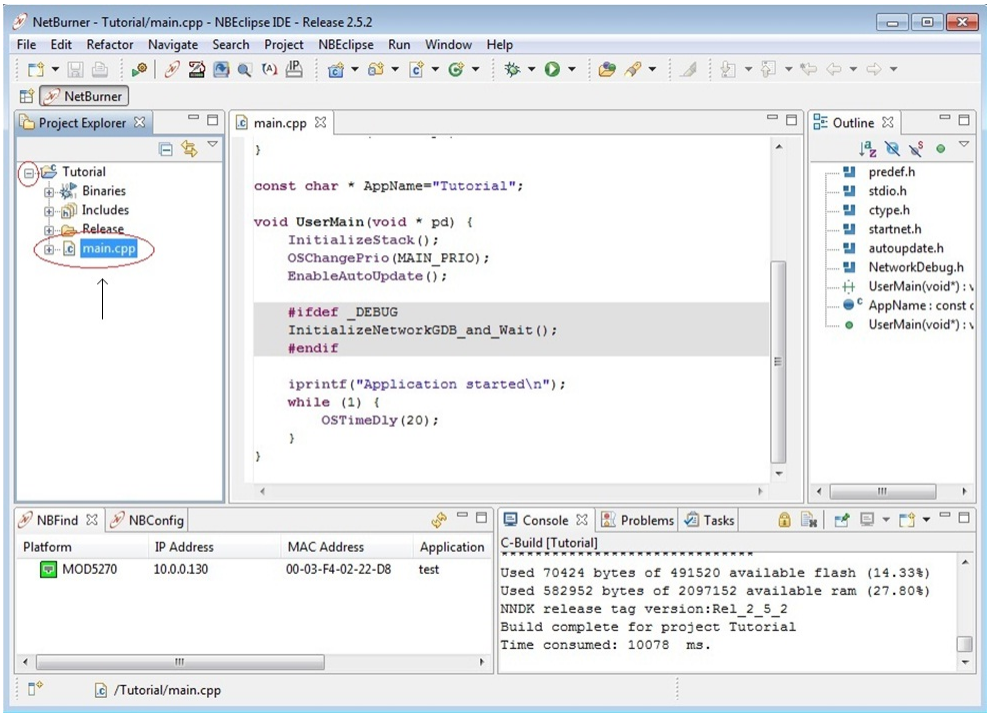


Selecting "**Finish**" will complete the new project creation.

At this point, a blank project should be created similar to the screenshot below.



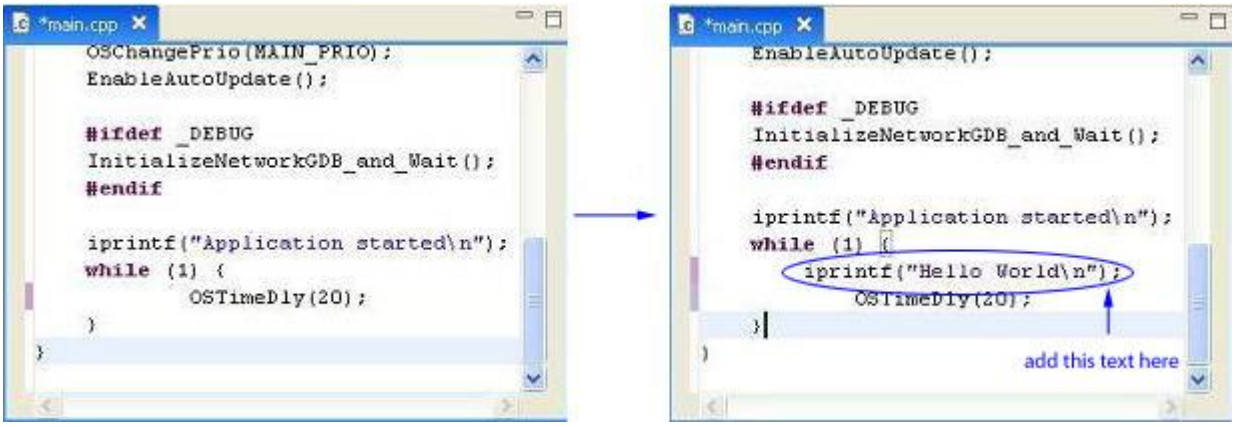
Click on the '+' sign next to the "**Tutorial**" folder to expand it. Double click on the "**main.cpp**" file to view it. The main.cpp is the main source code for your project. Ideally, this is where your main program code will be written.



The majority of the code is what we will refer to as the "**template program**". Do not alter it unless you are specifically told to do so. For more line by line explanation of the "**template program**", please refer to the Appendix.

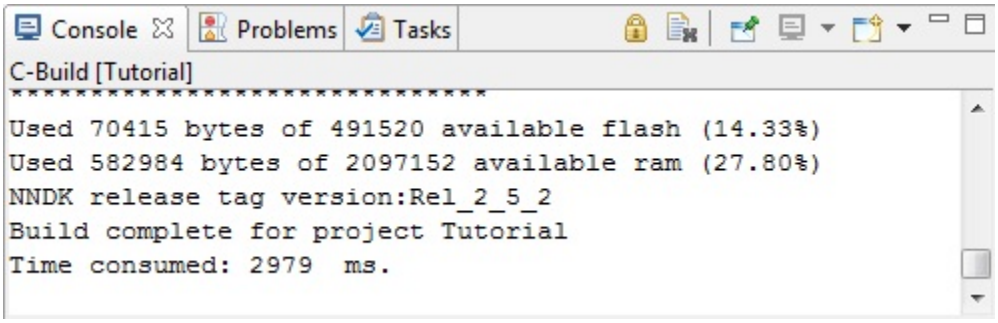
Starting Netburner Eclipse IDE

For our tutorial, we will simply create an application that will print the standard "**Hello World**" message to the monitor. Make the addition "iprintf("Hello World\n");" below as indicated by the arrow.



Save the modifications by either clicking 

Or **File** → **Save**



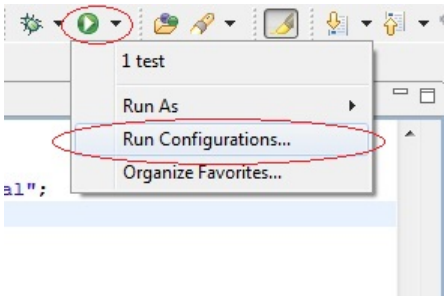
Each time an application is built as a "**Release**", two files are created. One is mapped for execution in RAM with an extension "**.s19**". The other is compressed and mapped for storage in flash memory with an extension of "**_APP.s19**". The two files are stored in the folder name "**Release**". In our case, the two files, tutorial.s19 and tutorial_APP.s19, are generated and stored in "**C:\Nburn\NBEclipse\workspace\Tutorial\Release**".

Running our application

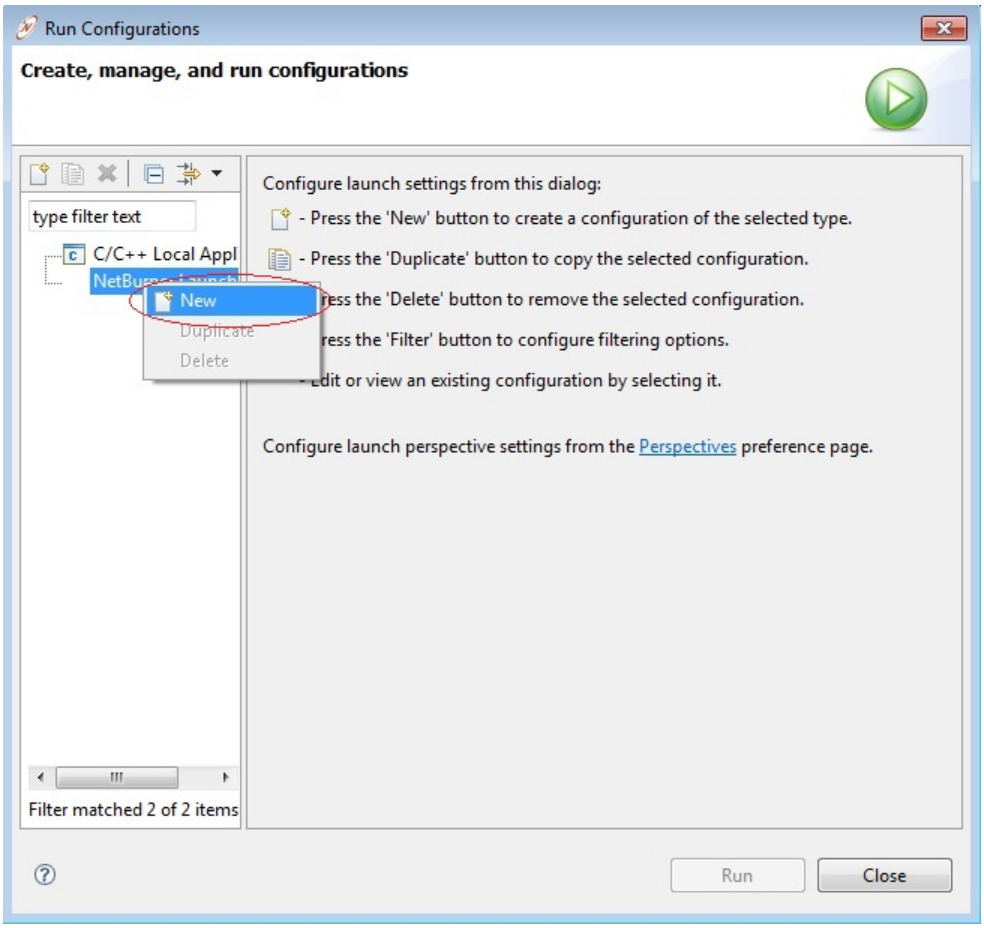
There are two ways to download the application to the board. The first and preferred way is to download via the network. The second way is to download it via the serial cable.

Downloading through network

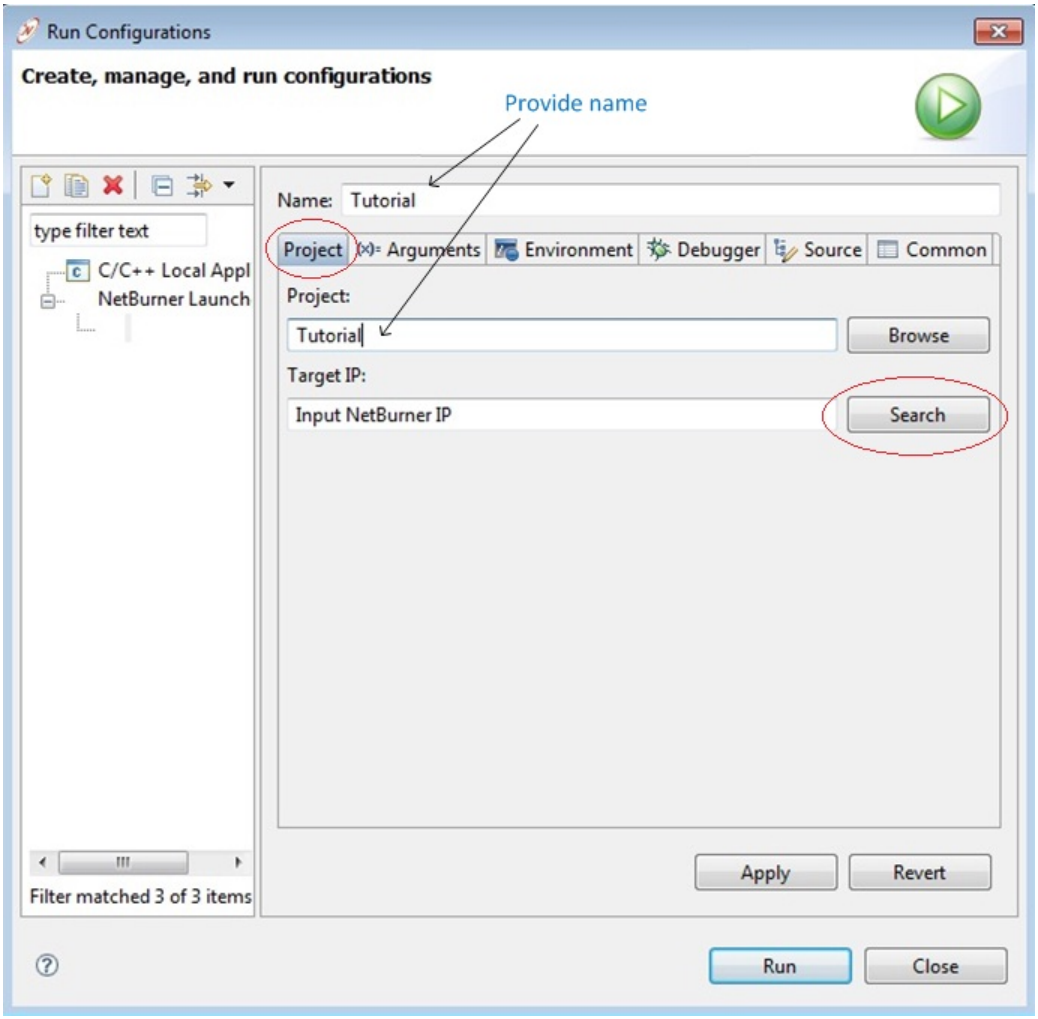
To download through the network, we must first create "**run**" configuration. To create a "**Run**" configuration we will use the "**Run Configurations**" combo box on the NBEclipse tool bar.



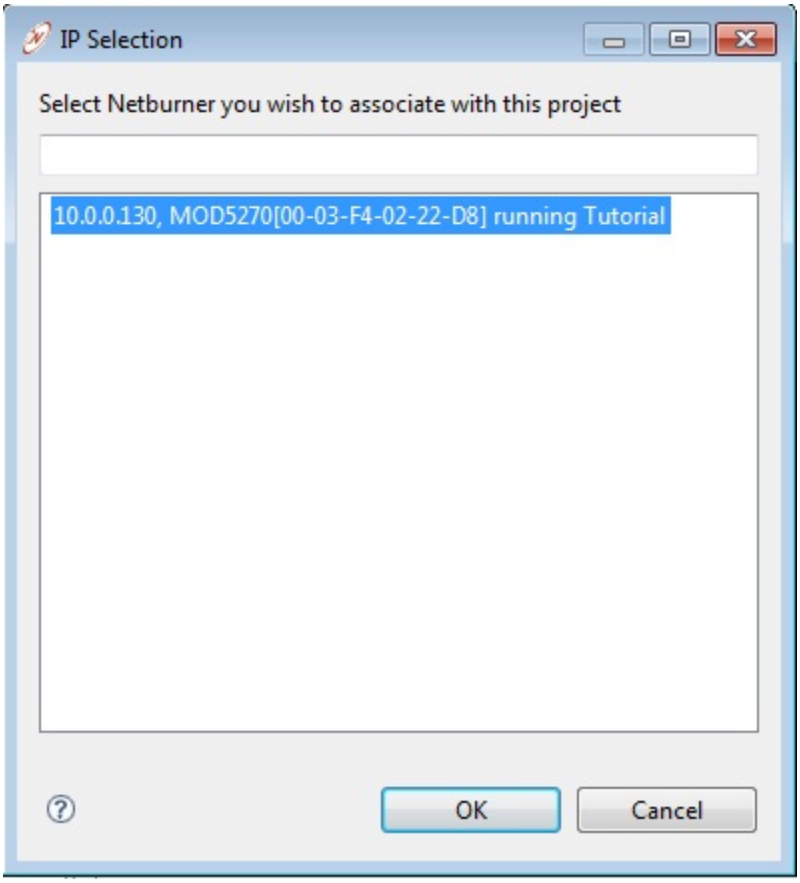
Selecting "**Run Configurations**" will display the "**Create, manage, and run configurations**" dialog box. Right click with the mouse on "**Netburner Launcher**" and select "**new**"



The configuration options are shown below with the "**project**" tab already selected. Fill in the textbox fields "**Name**" and "**Project**" with the name Tutorial.

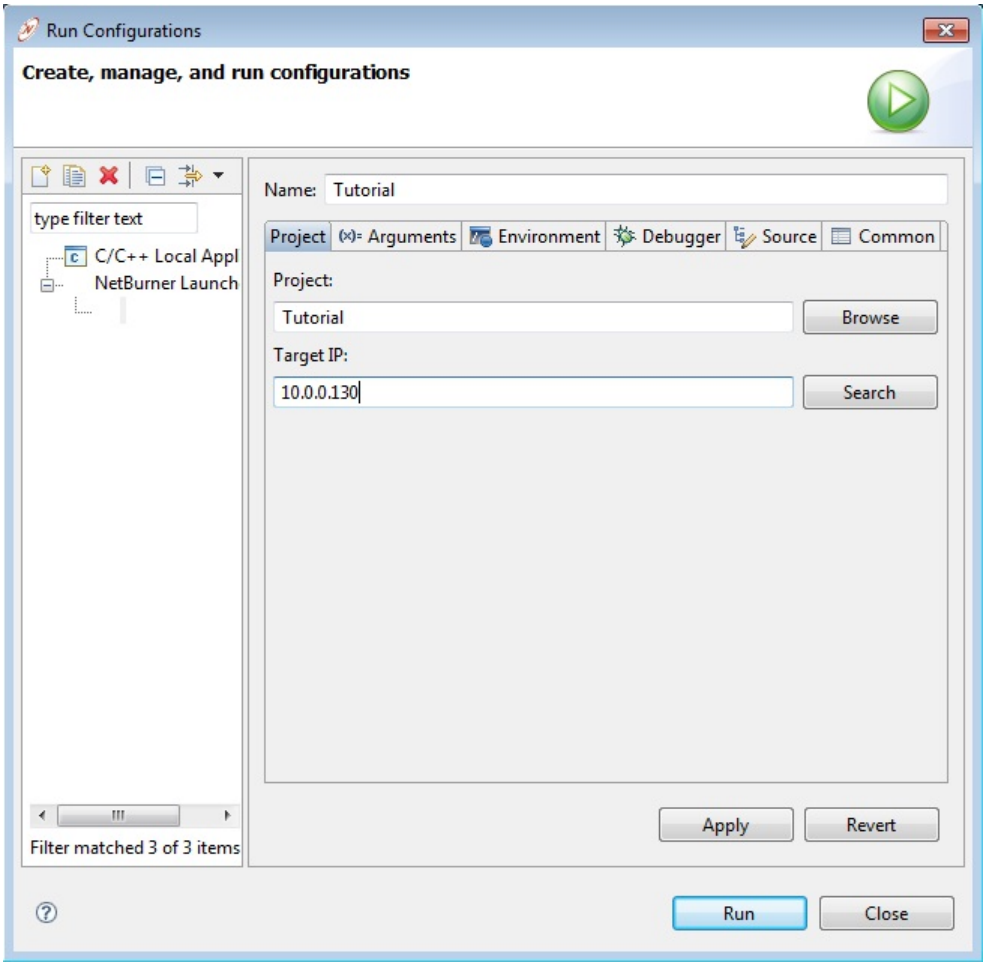


In the "**Target IP:**" textbox, either manually enter the IP address (If you know it) or click on search. **Note: Each station will have a different IP address**. Clicking on Search will bring up a dialog box for all the Netburner boards associated with the project. Since we only have one netburner hooked up, select the one that is show. For the figure below, the IP address is 10.0.0.100. Yours should be different.

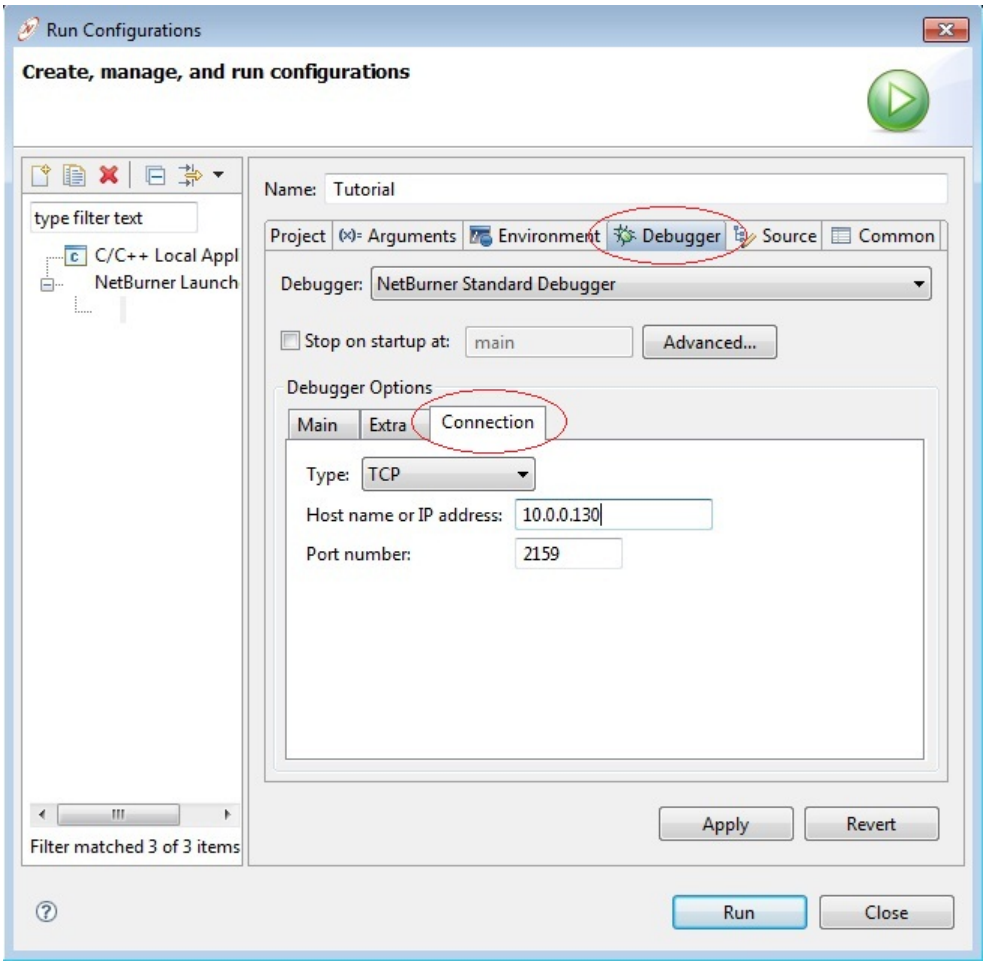


At this point, if you do not see the device in the list, then check your hardware setup or get a TA for help.

Select the Netburner device and click okay. The Target IP field should now contain the IP address of our netburner board.



Now click on the "**Debugger**" tab. Make sure the "**Netburner Standard debugger**" is selected in the "**Debugger**" list box. Click on the "**Connection tab**" in the debugger option. In the Hostname or IP address box, replace the content "**Input NetBurner IP**" with the IP address of your Netburner board. The address should be the same as the one entered in "**Target IP address**" under the project tab that we did a few step earlier.



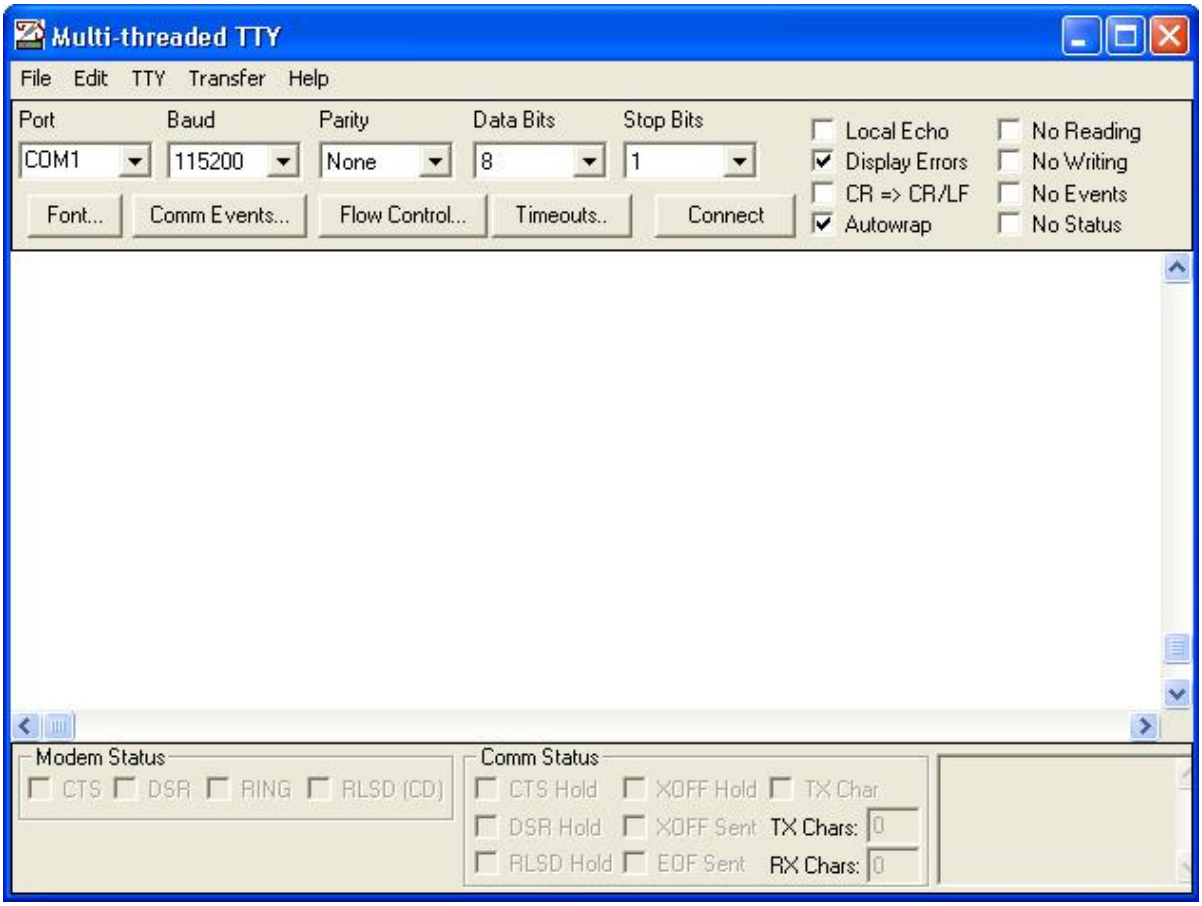
Click the "**Run**" button to compile the application and to download it to the hardware. The "**percent complete**" dialog box will flash briefly and then disappear indicating that the compile and download was a success.

Note: If there is an error with the download, please check to make sure that there is proper communication between the Netburner board and computer via the network. If you cannot resolve this issue, ask a TA for help.

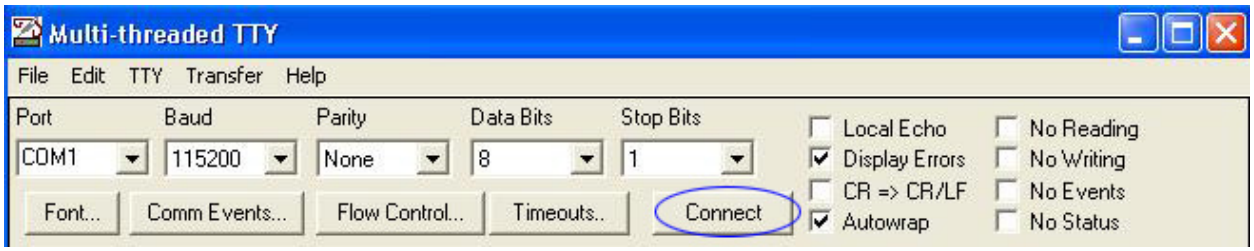
Using MTTTY

Once the application is downloaded to the Netburner board, it will start running within a few seconds. To monitor our application since it involves printing some text through the serial cable, we have to use a serial communication program. Netburner has provided MTTTY. Other programs exist such as HyperTerminal for Windows and Minicom for Linux. To launch MTTTY, you can either double click the MTTTY shortcut icon on the desktop.

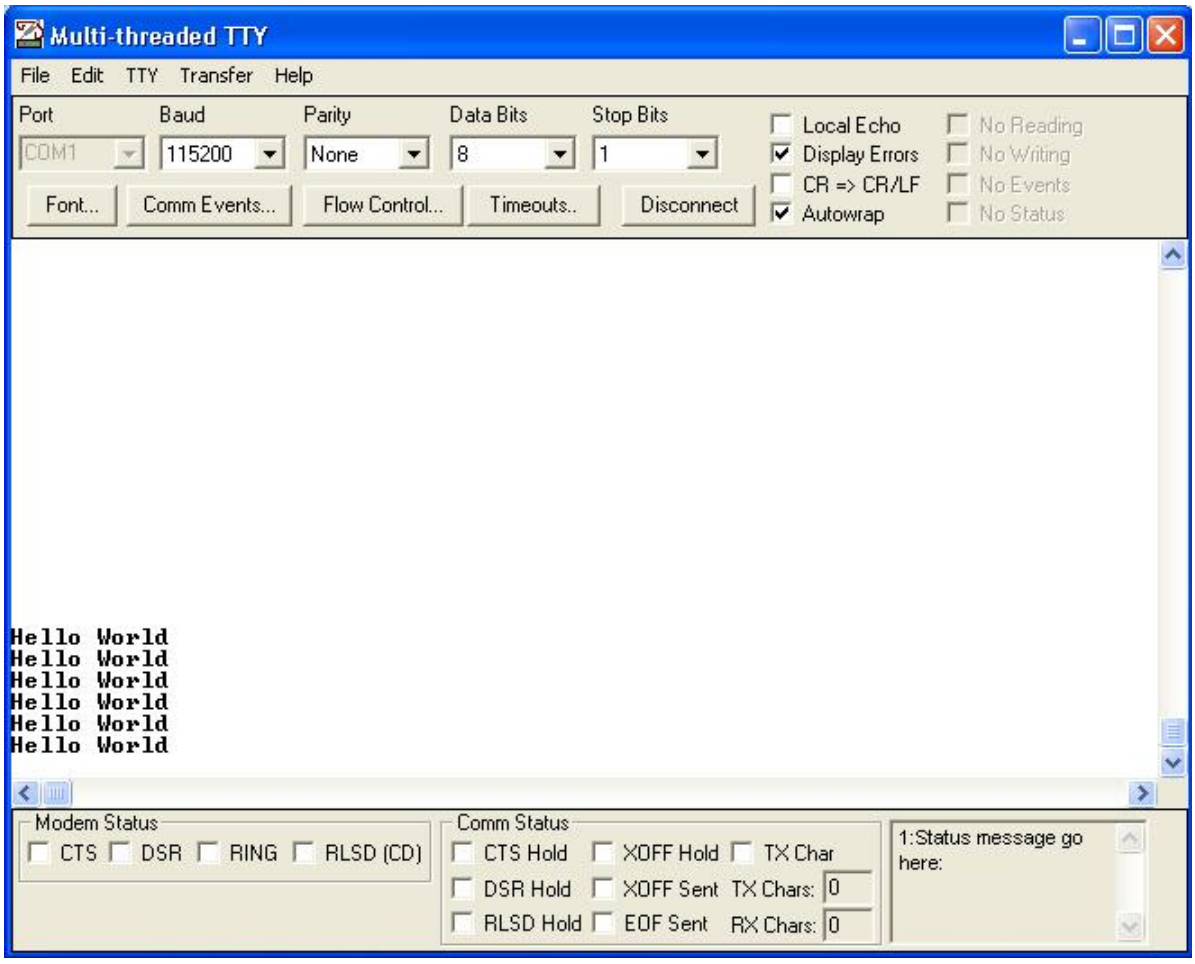
Start → All Programs → Netburner NNDK → MTTTY Serial Terminal



Do not change the parameters in "**Port**", "**Baud**", "**Parity**", "**Data Bits**", and "**Stop Bits**". Changing the parameters will result in improper serial communication with the Netburner board. Click on the button "**Connect**" to establish connection.

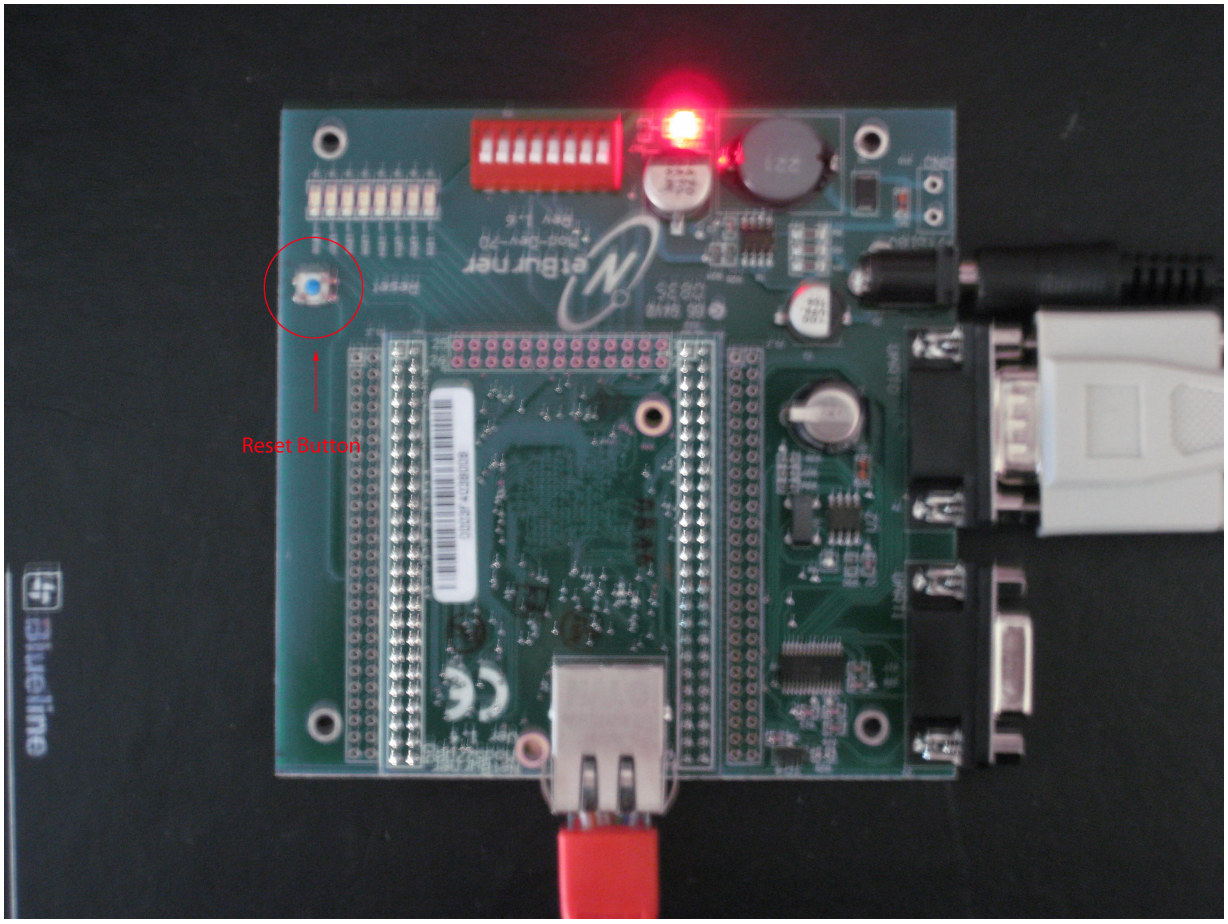


After connection with the board is established, we should see our application running with the text "**Hello World**" displayed on the monitor repeating.

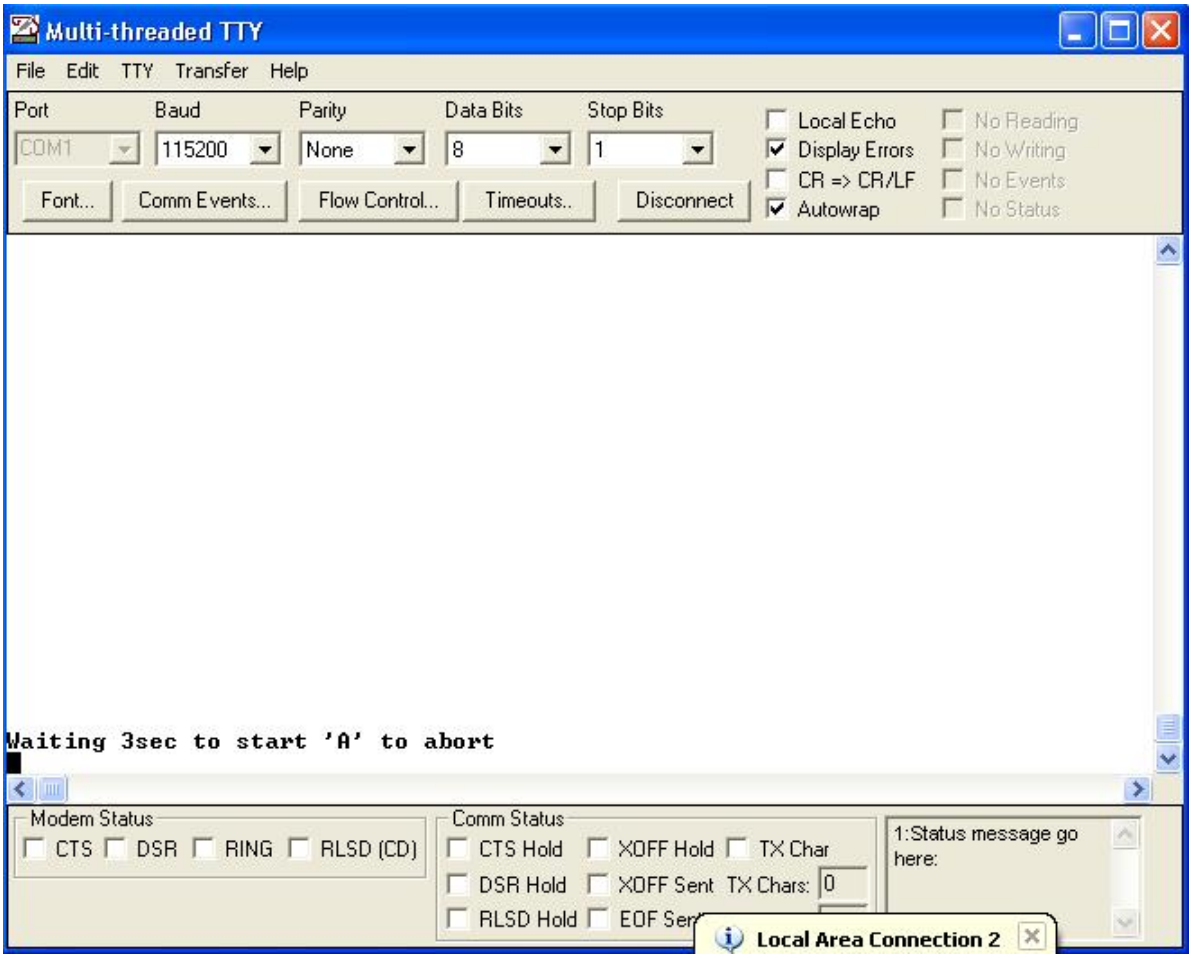


Downloading the application manually through MTTTY

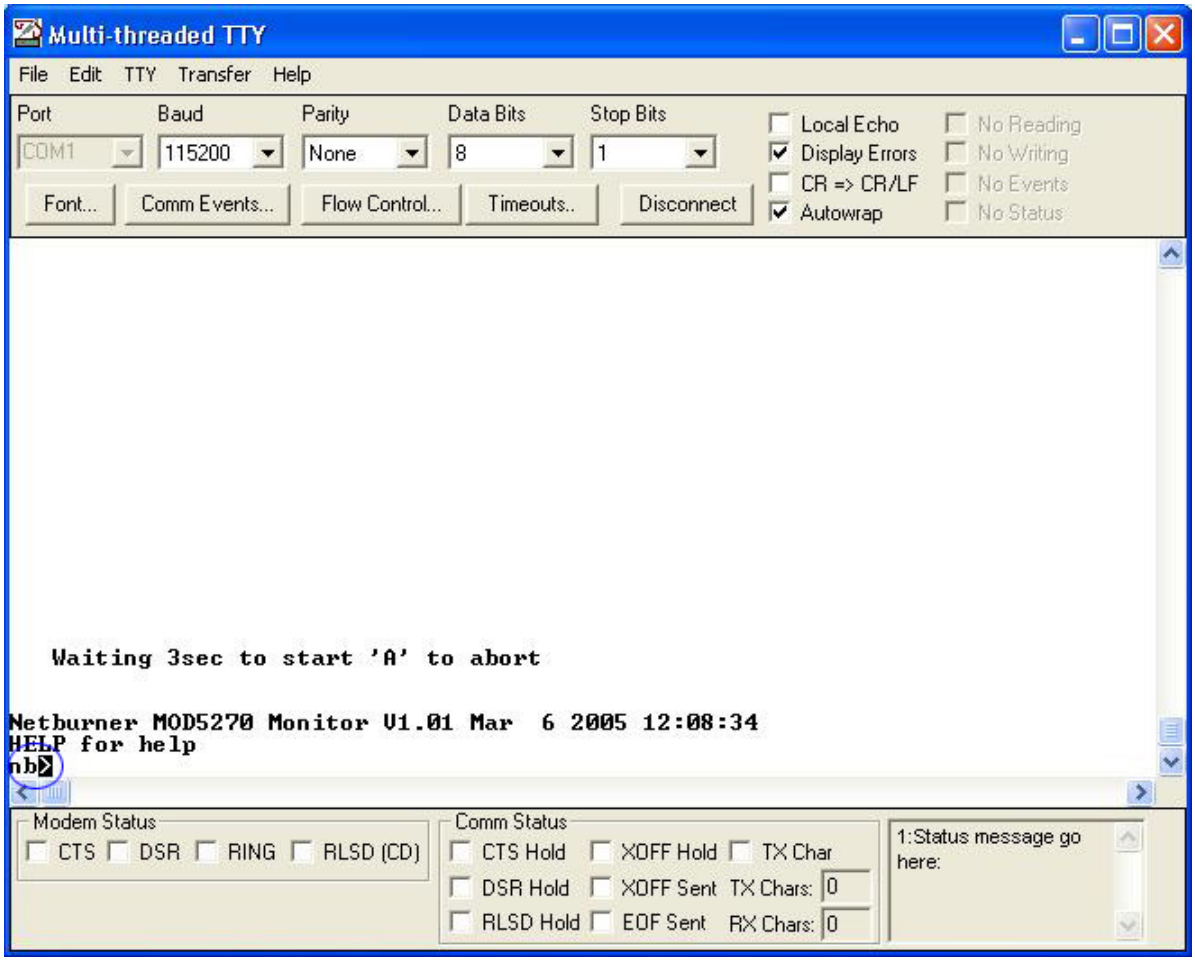
To download the application manually, click on the blue reset button on the Netburner Board.



Once the reset button is pushed, you will have 3 seconds to click type "**A**"(Shift+a) to enter into the boot monitor program rather than run your application program.



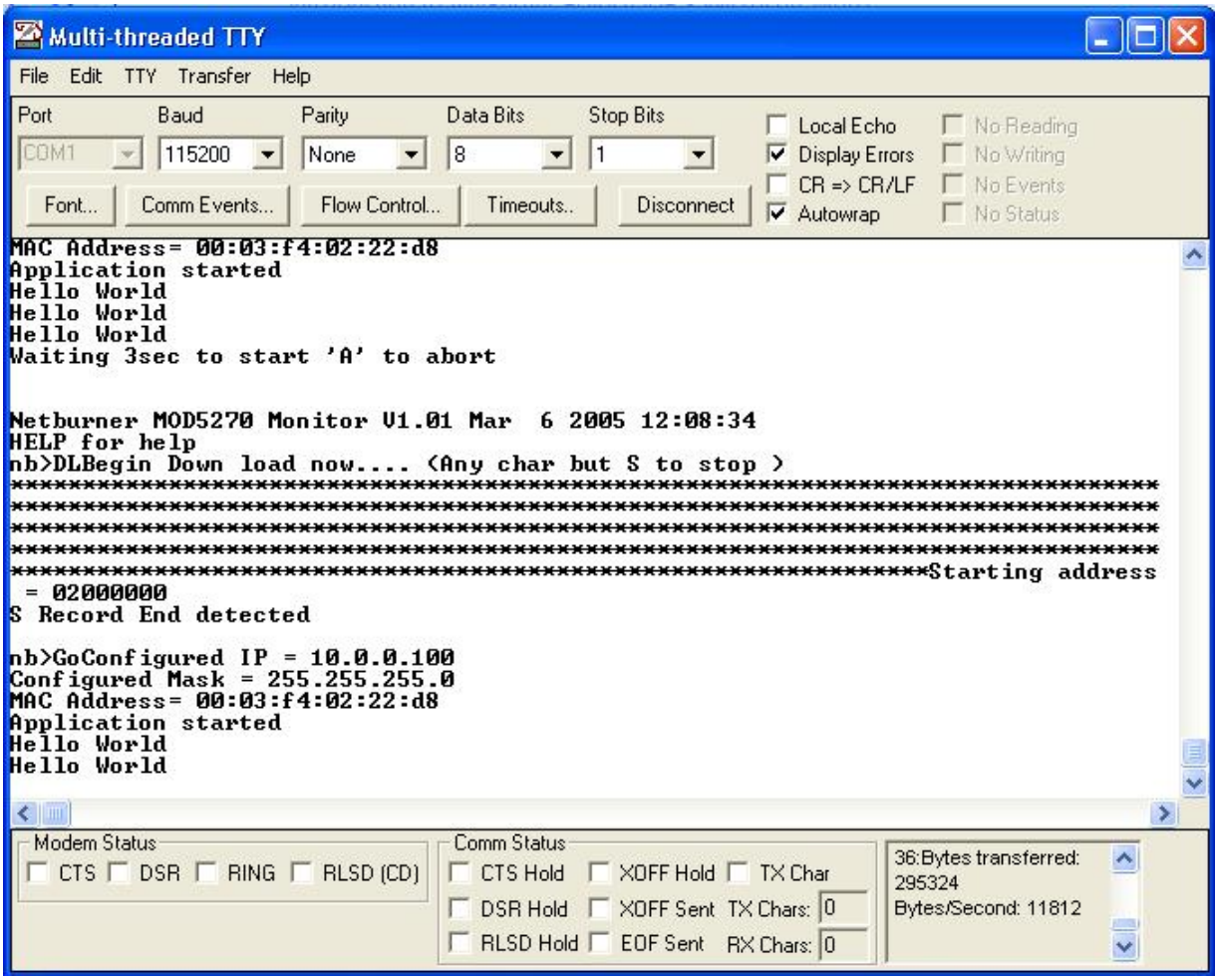
The Netburner boot monitor prompt (i.e. nb>) will appear indicating that the boot monitor program is now running on the Netburner board.



SDRAM Download

At the nb> prompt in the MTTTY window type the command **DL** (DL stands for download), then press the **Enter** Key. Next, send the **.s19** file to the board. You can either send the file by clicking **F5** or select **Transfer** → **Send File(Text)**. A send file window will appear. Navigate to the directory **C:\Nburn\NBEClipse\Workspace\Tutorial\Release** and select **tutorial.s19**

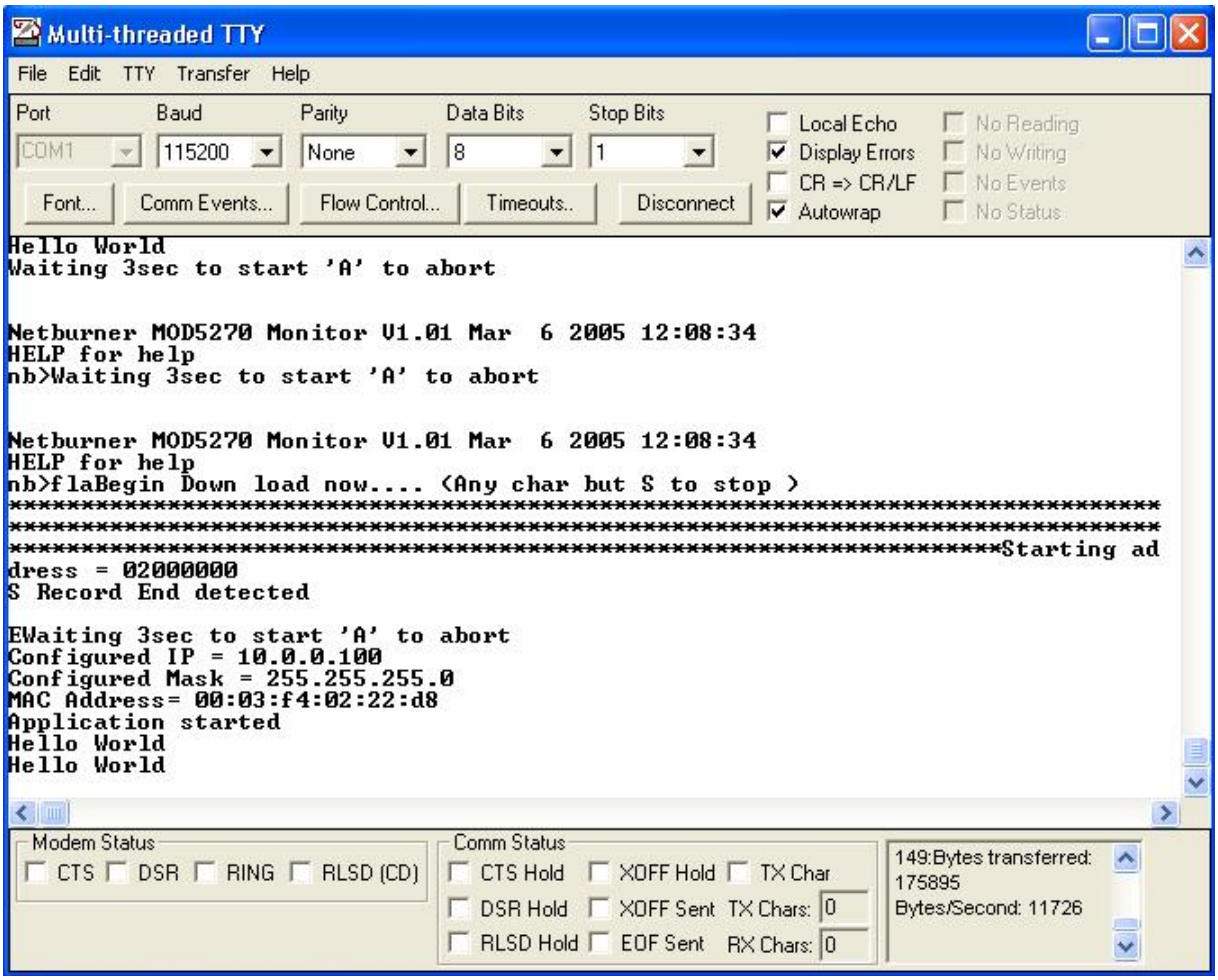
As the download progresses, "* * * * *" characters will appear in the MTTTY window, and the progress bar on the lower left hand side of the MTTTY window will move towards the right. Once the download has finished, type the command **'Go'** at the nb>prompt and press the Enter key to run your application.



Flash Download

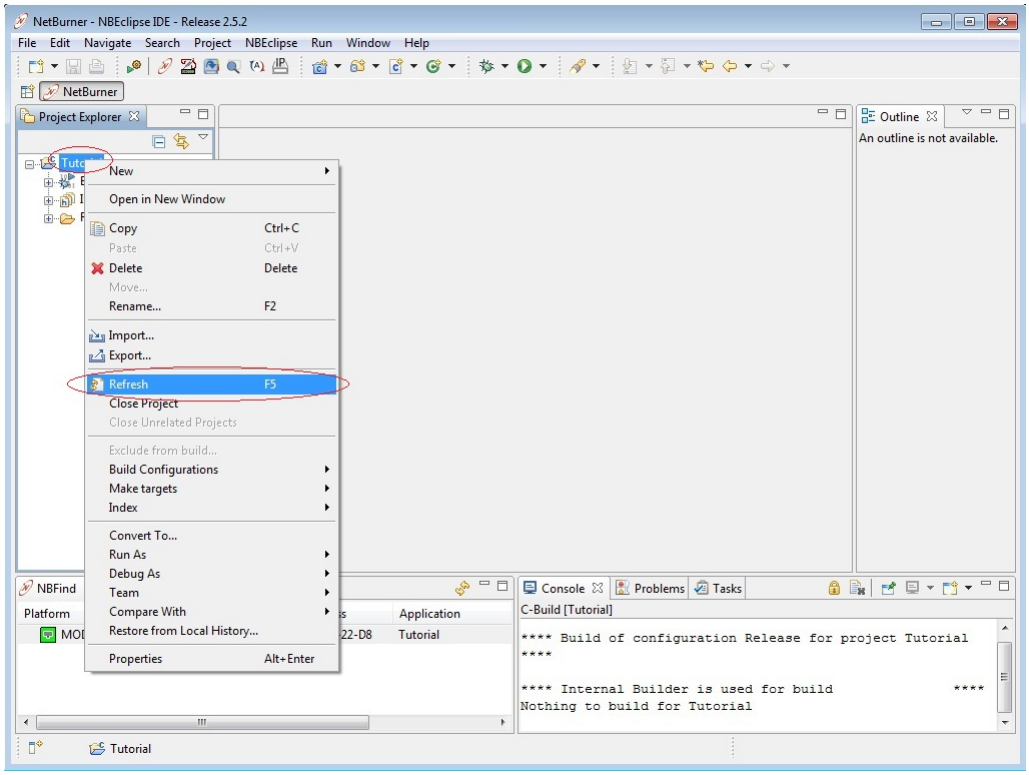
A flash download is very similar to a SDRAM download. At the nb> prompt in the MTTTY window type the command **FLA** (FLA stands for Flash application), then press the **Enter** Key. Next, send the **_APP.s19** file to the board. You can either send the file by clicking **F5** or select **Transfer 'Send File(Text)**. A send file window will appear. Navigate to the directory **C:\Nburn\NBEClipse\Workspace\Tutorial\Release** and select **tutorial_APP.s19**

As the download progresses, "* * * * *" characters will appear in the MTTTY window, and the progress bar on the lower left hand side of the MTTTY window will move towards the right. Once the download has finished, the Boot Monitor will automatically reprogram the application area of the flash and restart the Netburner board. If the program does not restart automatically, you can type the **BOOT** at the nb> prompt and hit **enter** to manually run your program application.

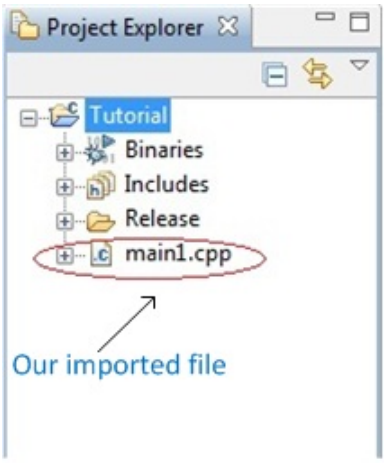


Importing files

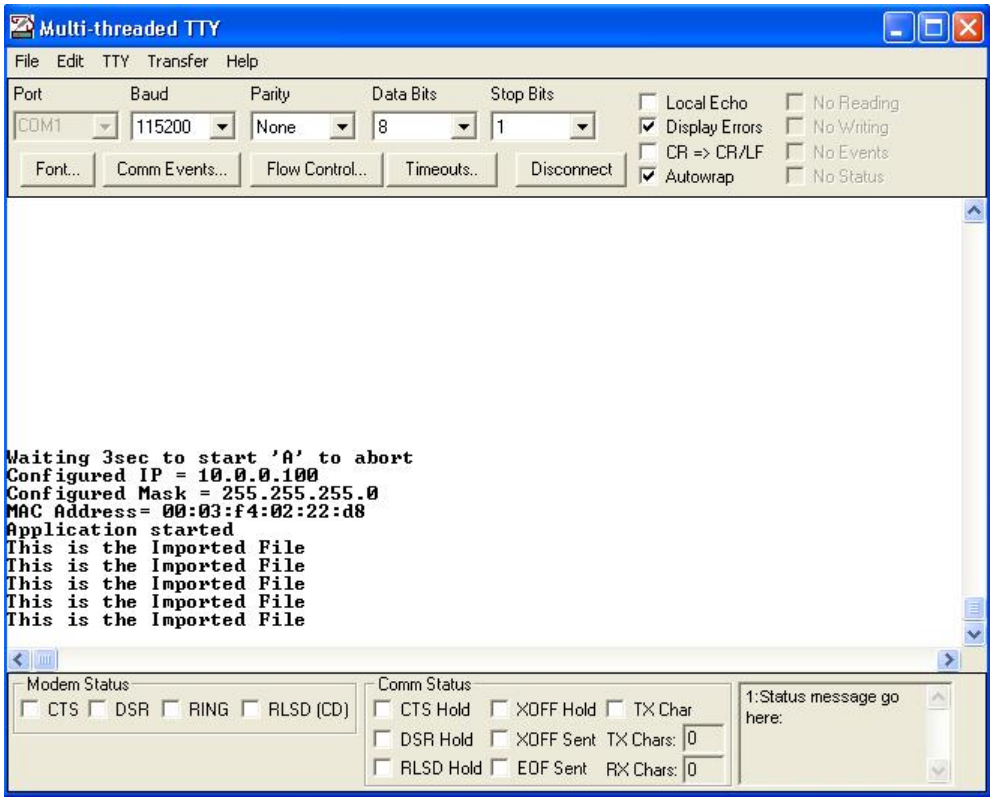
Delete the "**Main.cpp**" file from your Project. Download the modified source code called **main1.cpp**. Place the file in the directory where your project is stored. **C:\Nburn\NBEClipse\Workspace\Tutorial** To import the file **main1.cpp**, right-click on the "**Tutorial**" project in the "**Navigator**" window on the left side of the screen, and select "**Refresh**" from the pop-up menu.



After clicking refresh, you should see "**main1.cpp**" added in the "**navigator**" window.



Compile and download the new application to the board and observe what the new text is displayed in MTTTY. You can either method discussed earlier in the section "**Downloading to Netburner board**" to do this. The screen shot below shows the modified text.

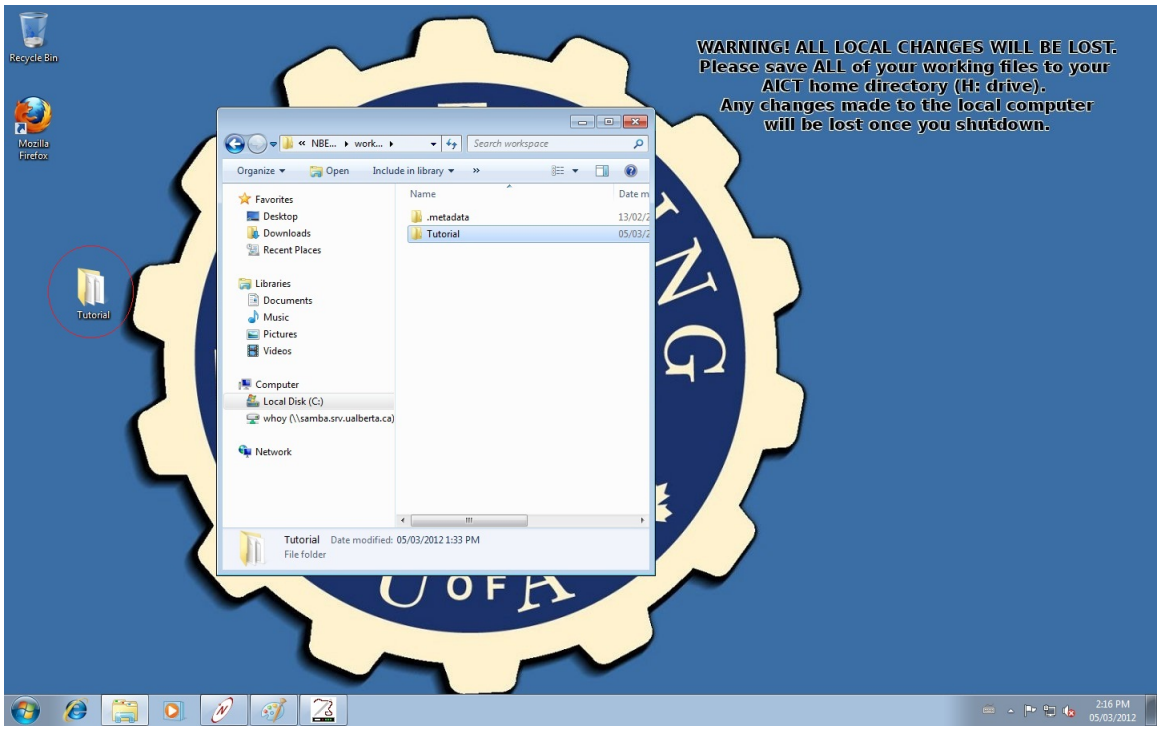


Exporting projects

Navigate to the directory where your project files are stored.

C:\Nburn\NBEClipse\Workspace

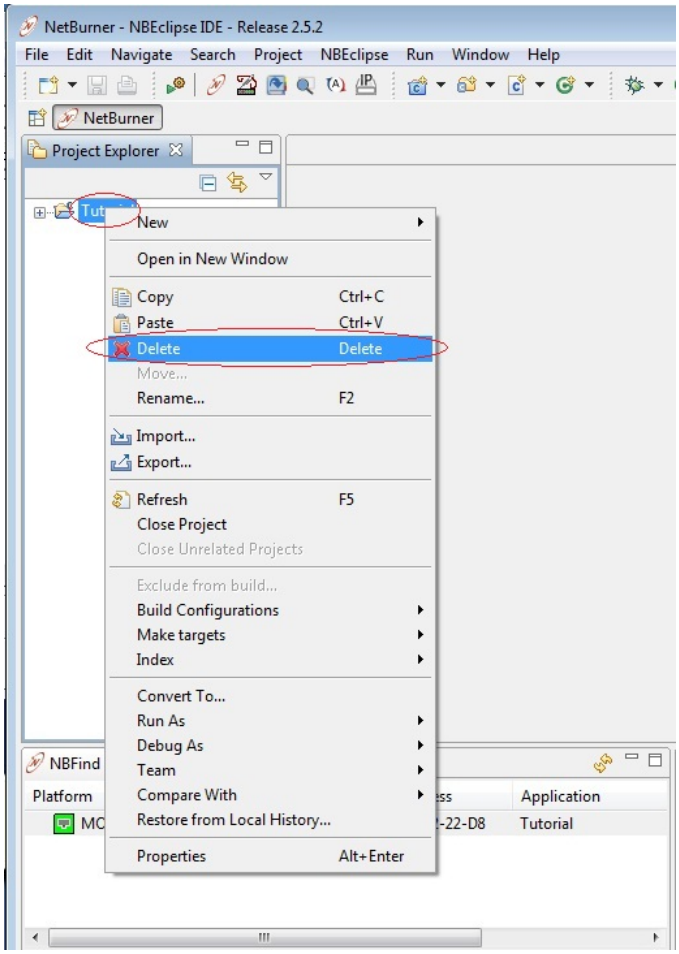
Copy the entire **'Tutorial'** folder and place it either on the desktop or in your ECE home directory.



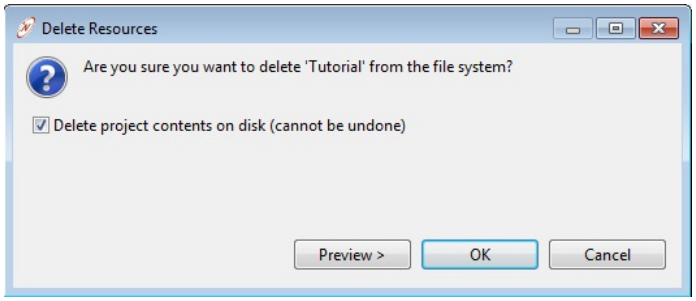
Netburner Eclipse has an **'Export'** option but I don't reccommend using it because it causes problems.

Importing projects

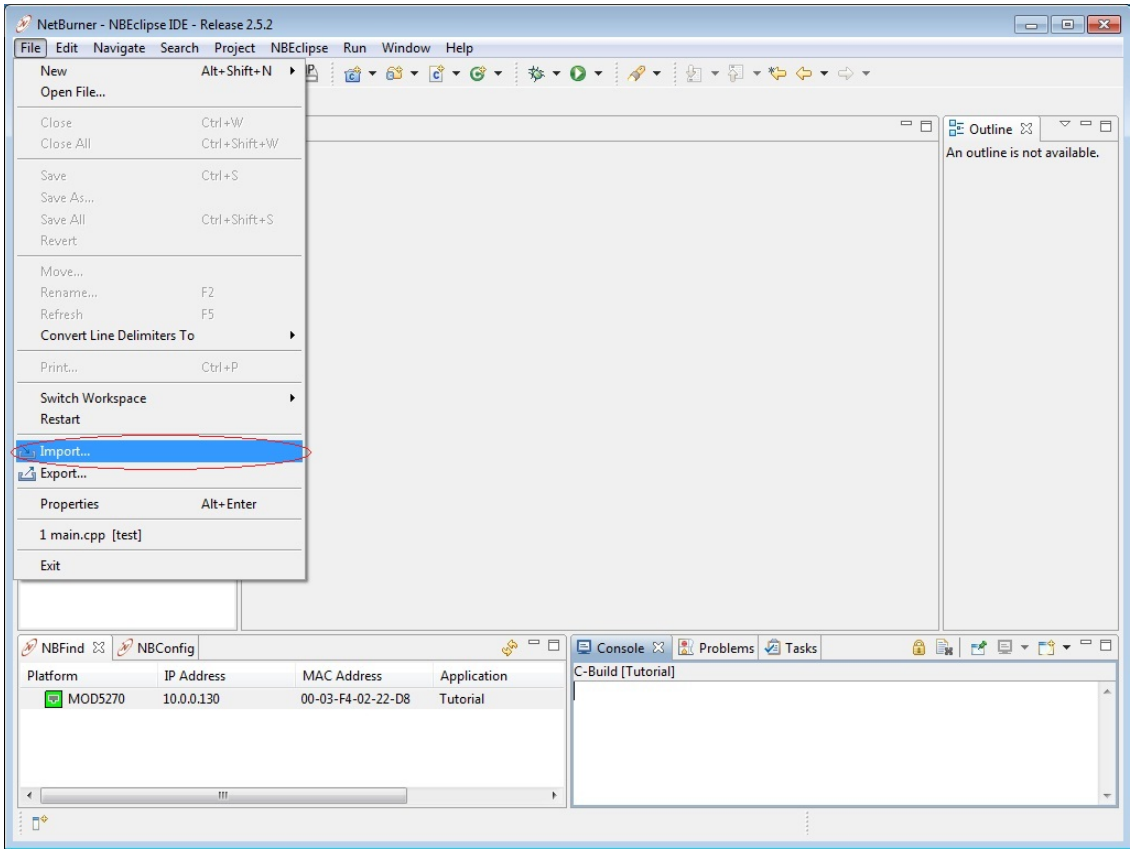
Close the MTTTY program and delete the Tutorial project. You can do this by either deleting it in Elicpse or by deleting the Tutorial folder in the Workspace. In Eclipse, highlight your project folder, right click and select delete.



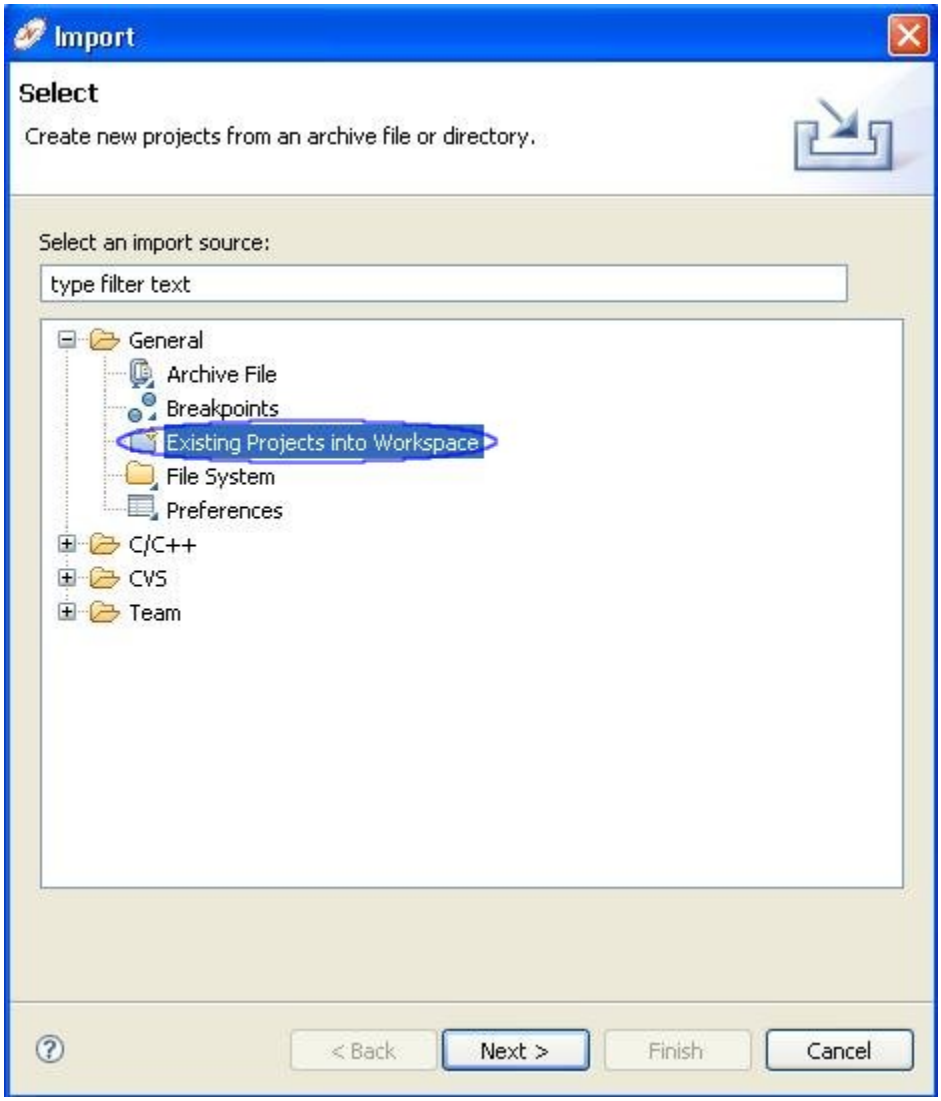
Click on the checkmark box to delete the project permanently. Before you perform this step, make sure that you have your project backed up somewhere.



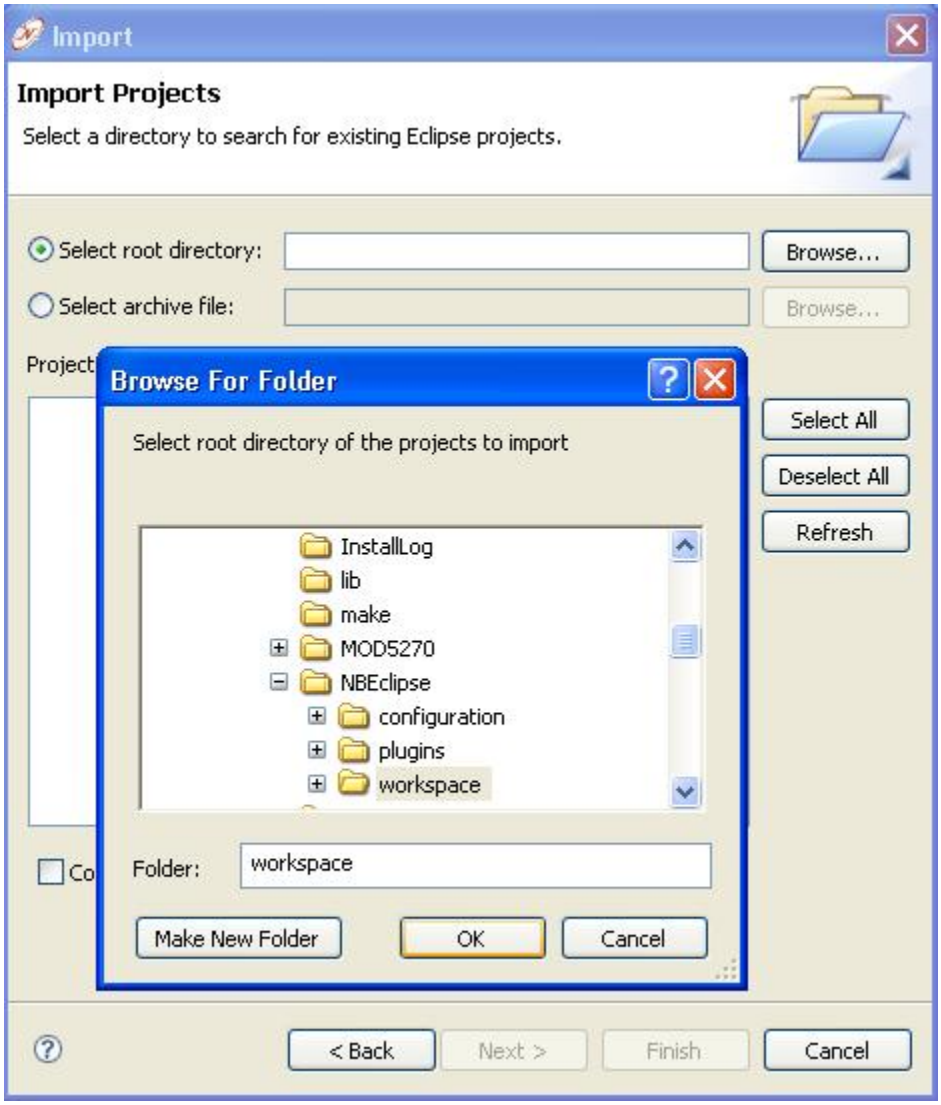
To restore our project, simply copy the backup version from the desktop and place it back into the Workspace directory. Select the **'import'** option in Elipse.



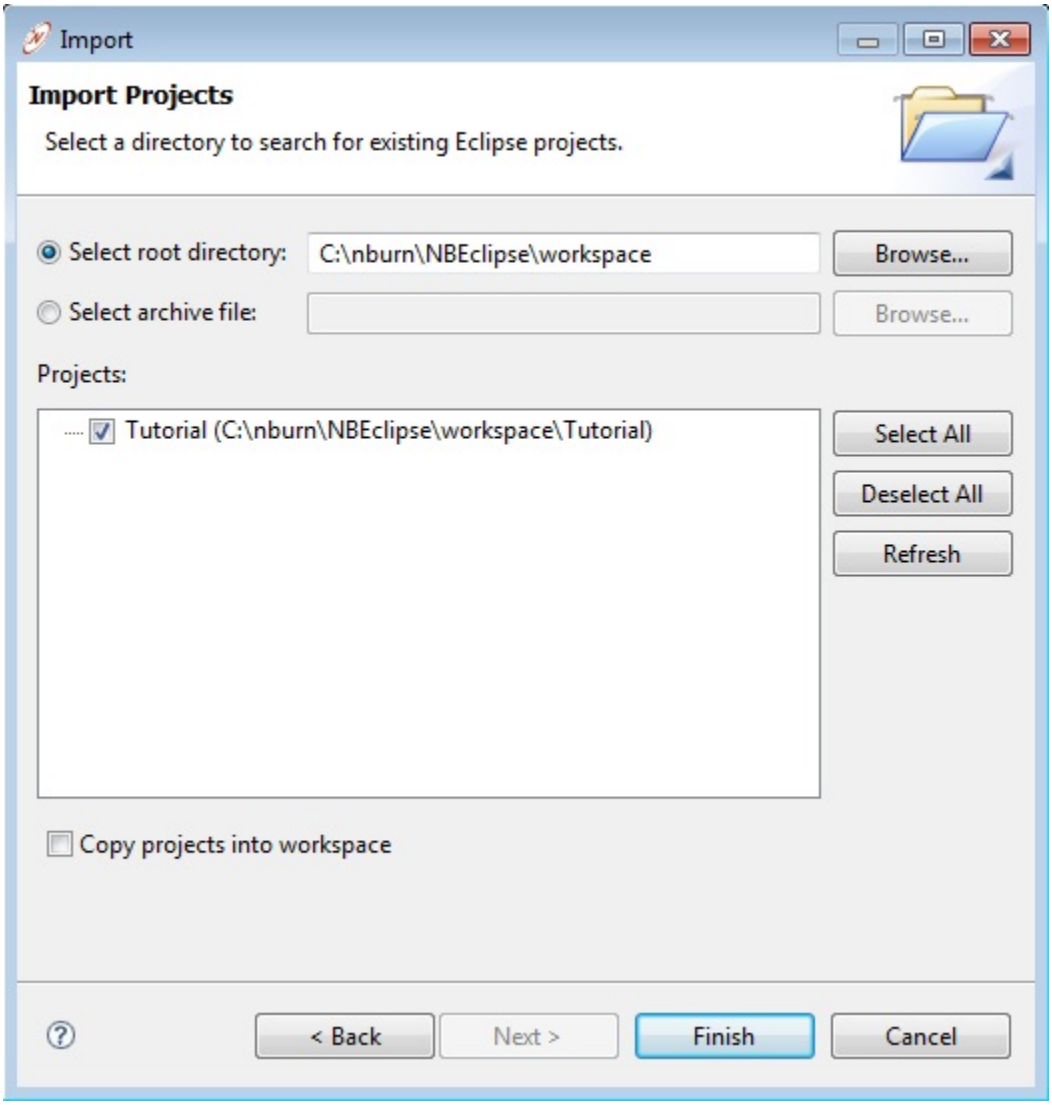
Select **'Existing projects into Workspace'**



In the **'Select root directory'** click **'Browse'** and choose the Workspace folder. Click **'OK'**



Click **'Finish'** to complete the import



Debugging

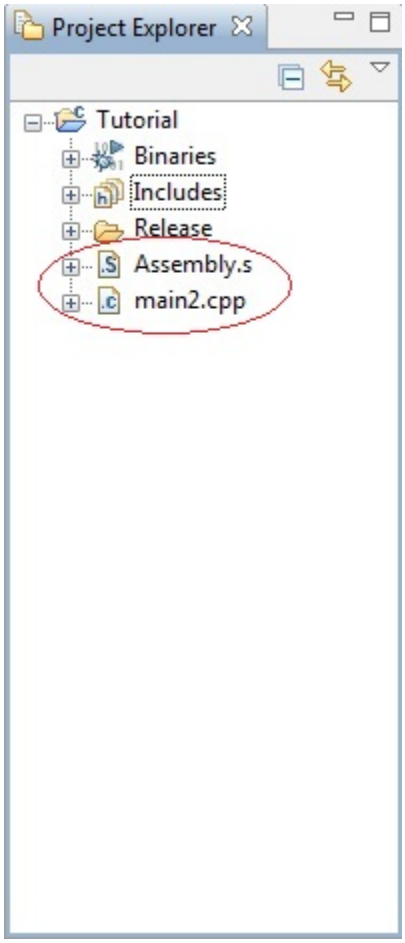
To add Network debugging capabilities, you must include the NetworkDebug.h header file and one of the following two functions, **InitializeNetworkGDB_and_Wait()** or InitializeNetworkGDB(). This option is usually selected when creating our project with the NetBurner Application Wizard. If you recall earlier in the tutorial, the option "**Network Debugging**" was checked off during the configuration process in "**Network Project Options**".

main1.cpp

```
#include <startnet.h>
#include <autoupdate.h>
#include <NetworkDebug.h>


extern "C" {
void UserMain(void * pd);
}

const char * AppName="Tutorial";

void UserMain(void * pd) {
    InitializeStack();
    OSChangePrio(MAIN_PRIO);
    EnableAutoUpdate();

    #ifdef _DEBUG
    InitializeNetworkGDB_and_Wait();
    #endif
```

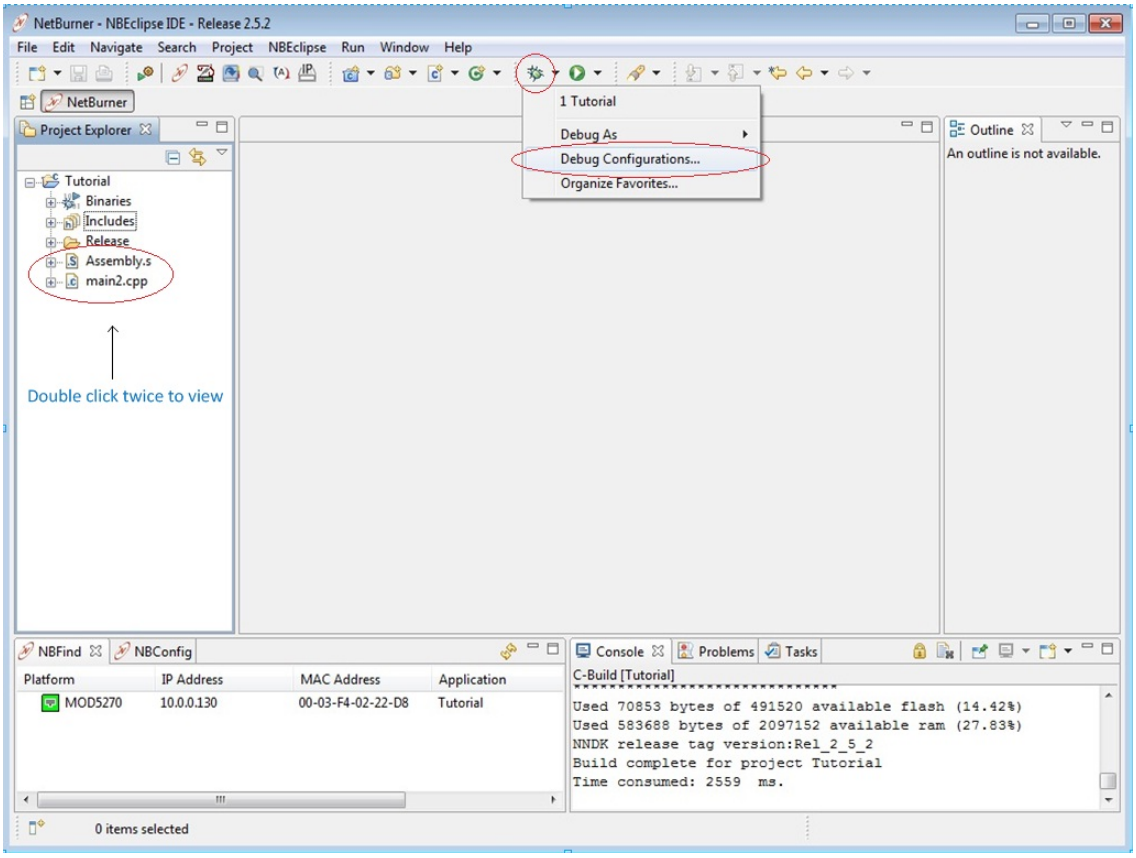
Running the Debugger

For our example, please download the following files **Main2.cpp** and **Assembly.s** and place them in your project directory. Deleted "**Main1.cpp**" from your project.

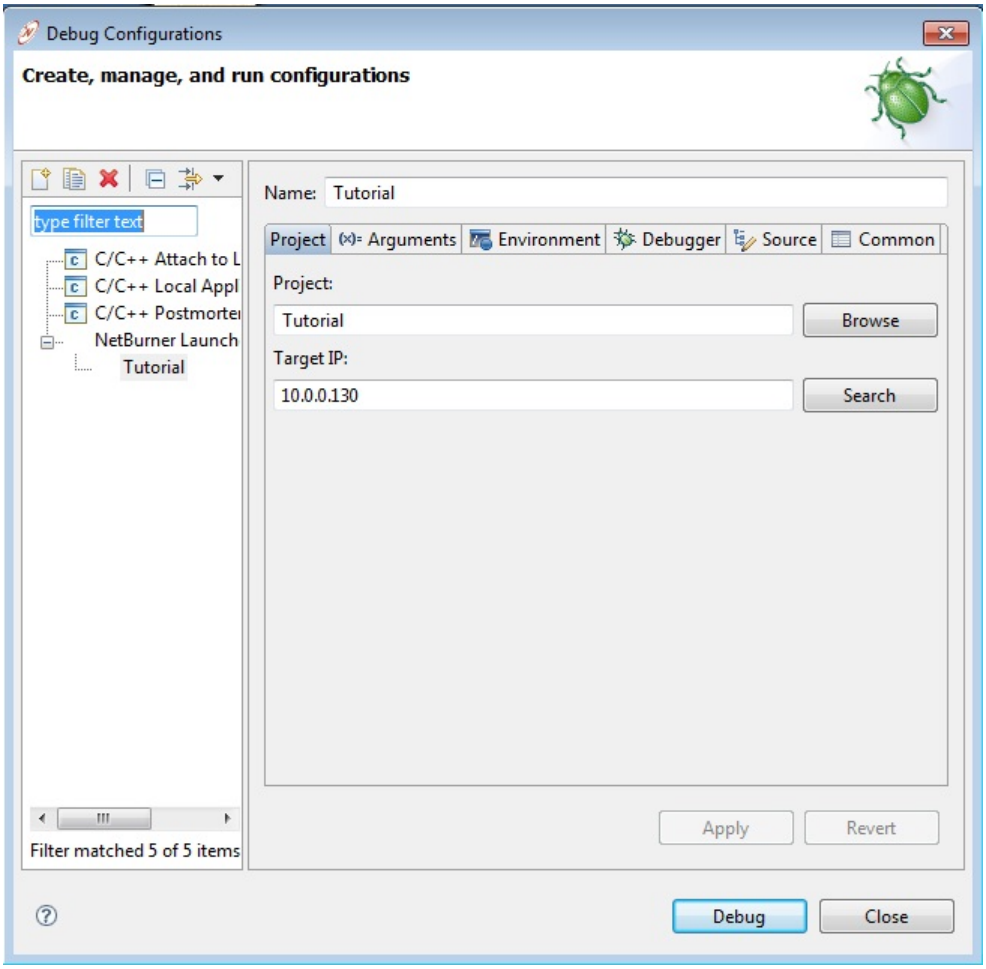


Running the Debugger

Double click on "**main2.cpp**" and "**assembly.s**" to view the source code. To start a debug session, we will use the "**debug**" combo box on the NBEclipse tool bar.

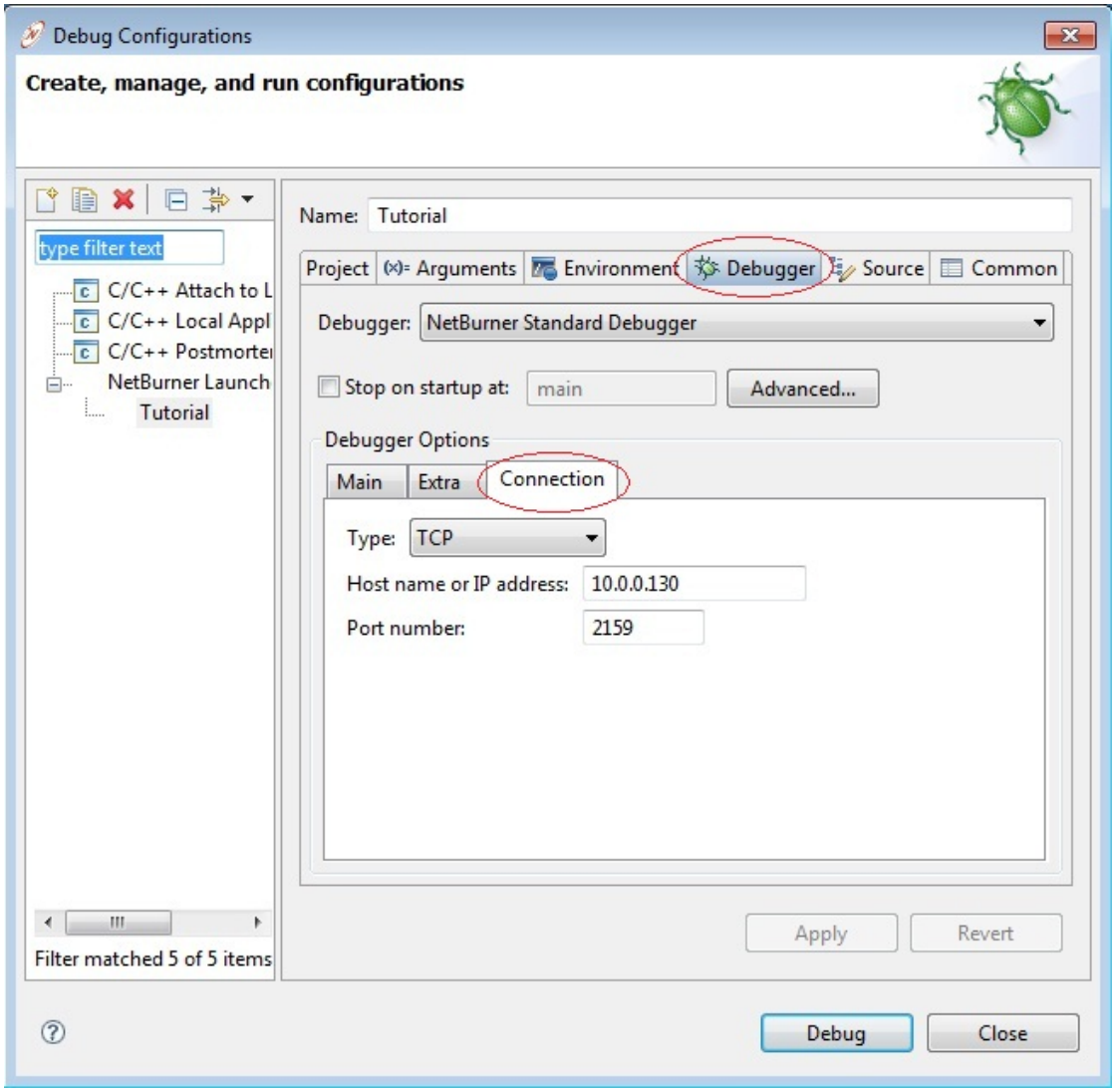


Selecting "**Debug**" will display the "**Create, manage, and run configurations**" dialog box.

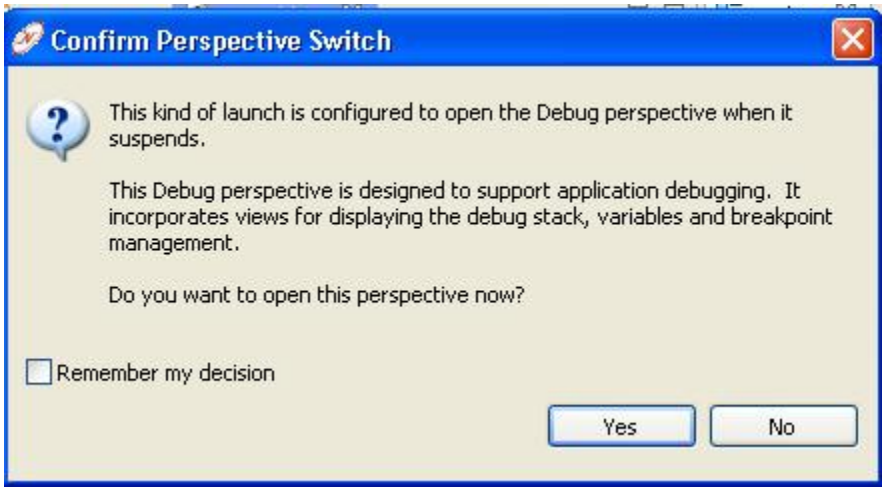


In the "**Target IP:**" textbox, the IP address should already be entered. If the field is blank, enter it manually or click on "**Search**".

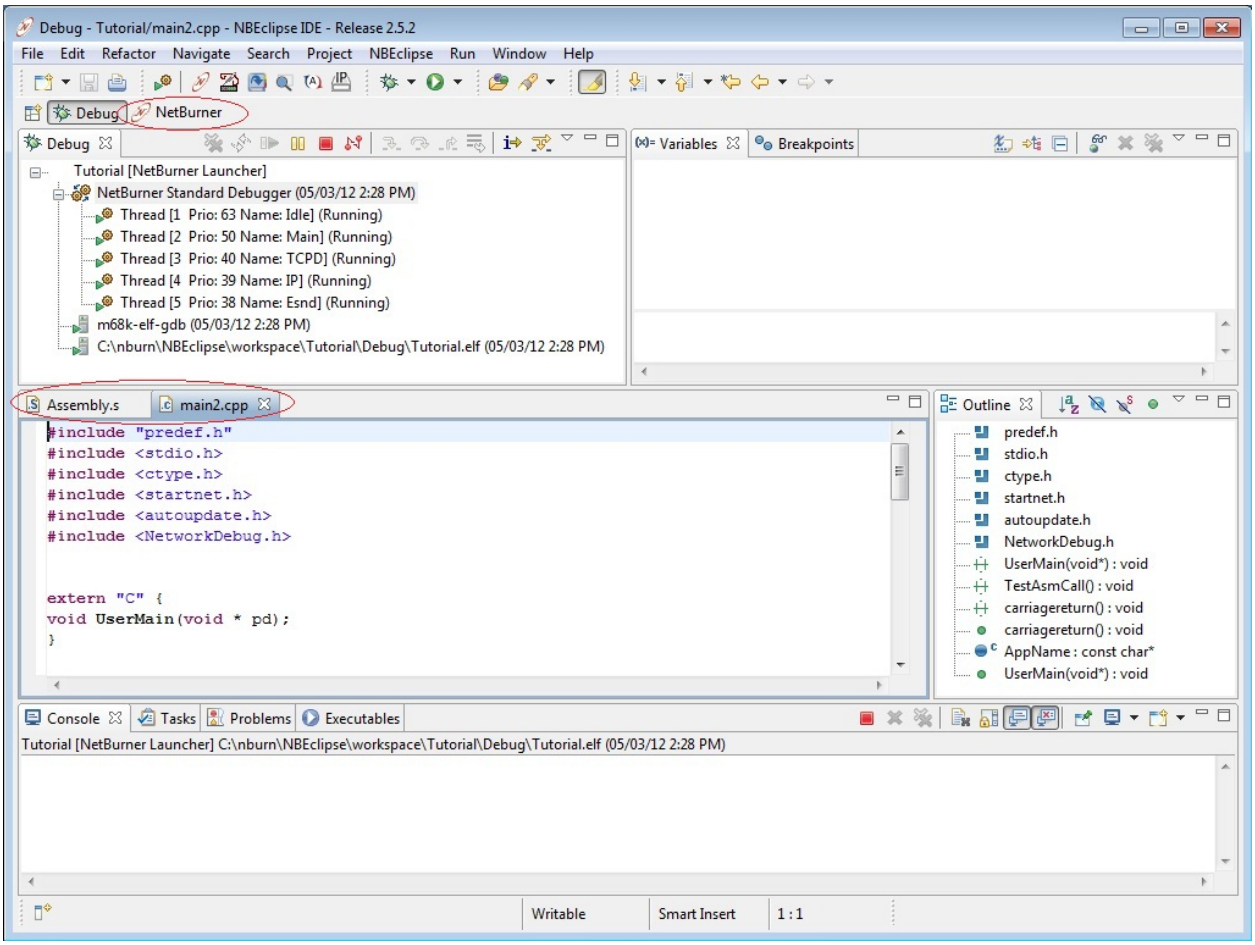
Click on the "**Debugger**" tab and click on the "**Connection**" tab. Once again, the IP address should already be entered. If the field is blank, enter it manually.



Now that the proper parameters are entered, click on "**Debug**" to start our debug session. A "**confirm perspective switch**" dialog box will pop up. Click on "**Yes**" to continue.

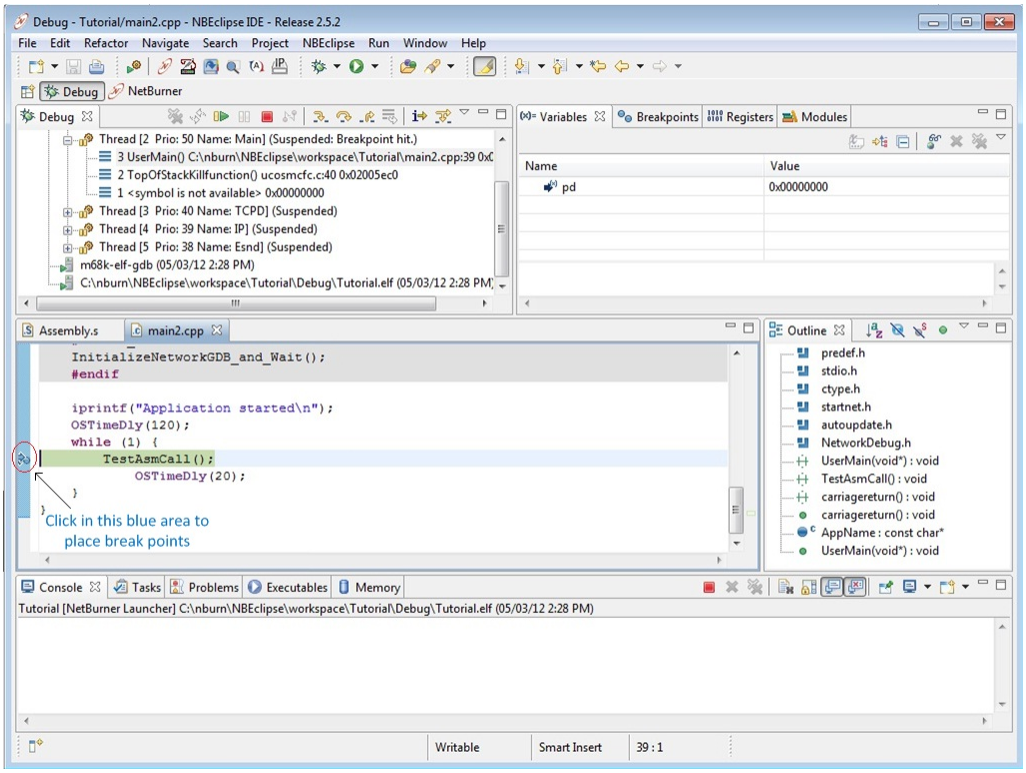


A screenshot of the "**Debug Perspective**" is shown below.



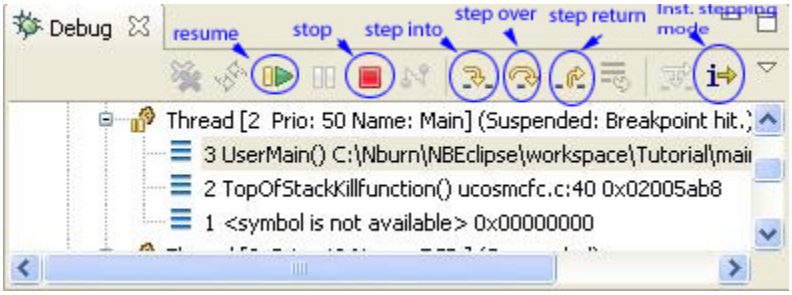
If "**main2.cpp**" and "**assembly.s**" are not present, click on the "**Netburner**" tab and open them in the "**Navigator window**" by double clicking on the files. Then return to the Debugger window.

Breakpoints are set by double-clicking in the far-left gray-colored margin next to the source code lines in the edit window. Click on the "**main2.cpp**" to display it and place a breaker next to the line "**TestAsmCall()**" by double clicking the gray are next to it. Double clicking twice on the breakpoint will get rid of it.

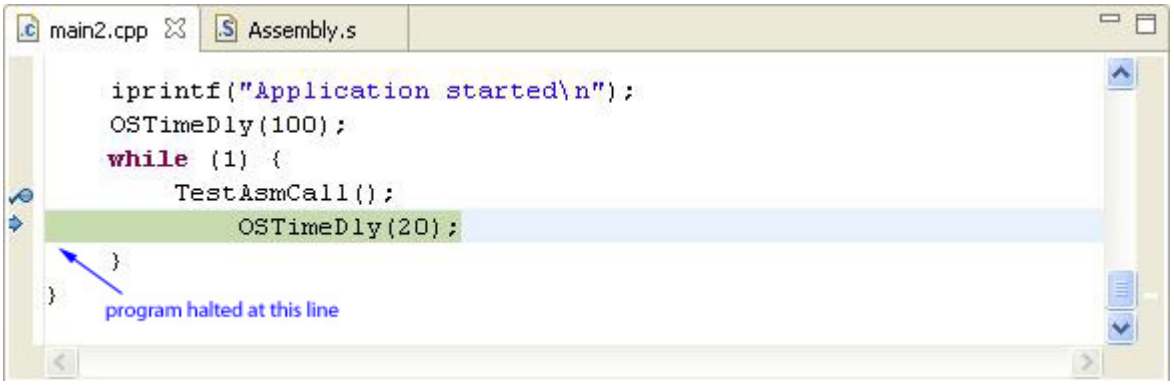


Once a breaker is placed, the program will halt at that line and the "**Debug**" perspective will update its windows and variables to display the current state of the application.

The first active icon is the green "**Resume**" button in the "**debug**" window. Clicking on it will resume the application until it hit the next breakpoint. The red "Terminate" icon ends the debug session. The yellow arrows from left to right are "**Step Into**", "**Step Over**", and "**Step Return**". These enable you to single step to the next line (step over), step into a function call (step into), or run until the current function call returns (Step Return).

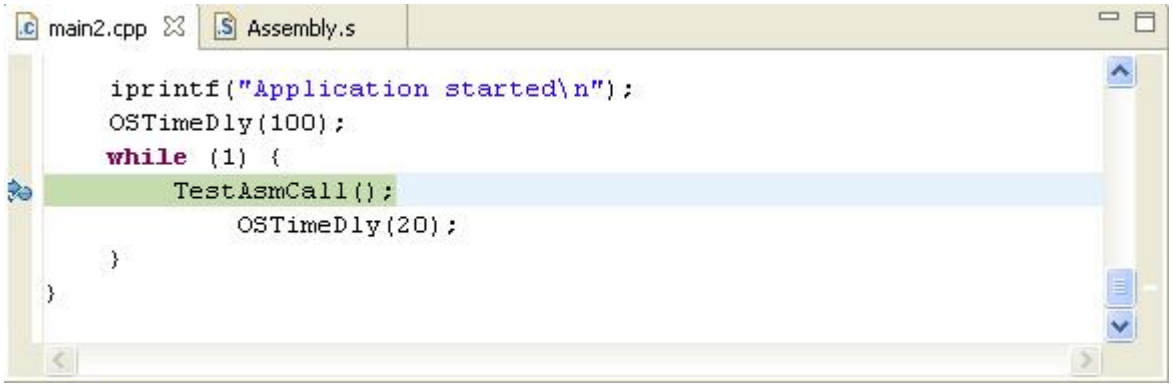


Click on the "**Step over**" to see what happens.



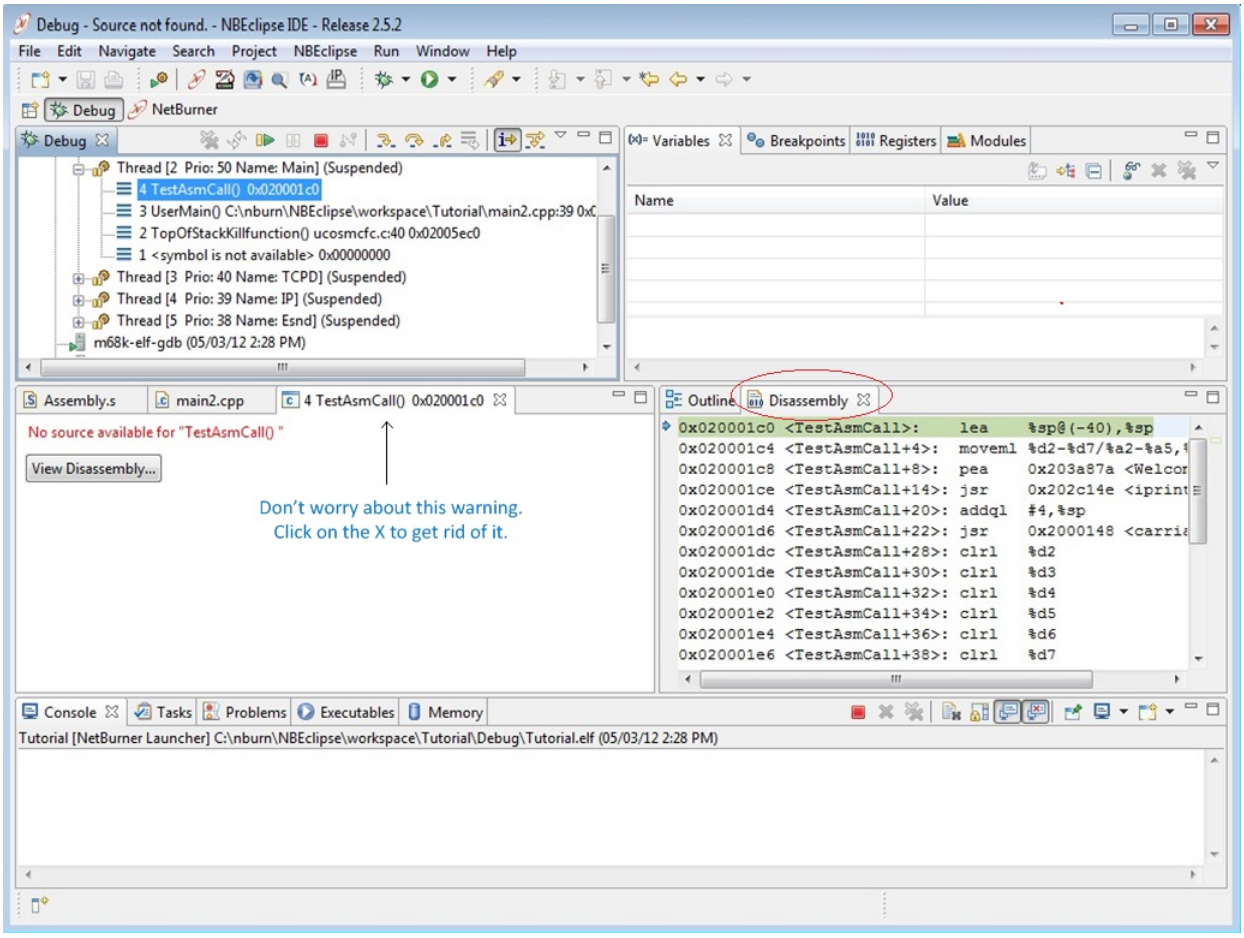
Notice, that our program is now halted at "**OSTimeDly(20)**" which is the line after "**TestAsmCAll()**". You can tell where you are in the program by locating the arrow in grey shaded area or the line of code that is highlighted in green.

Resume our program by clicking the green "**Resume**" button.

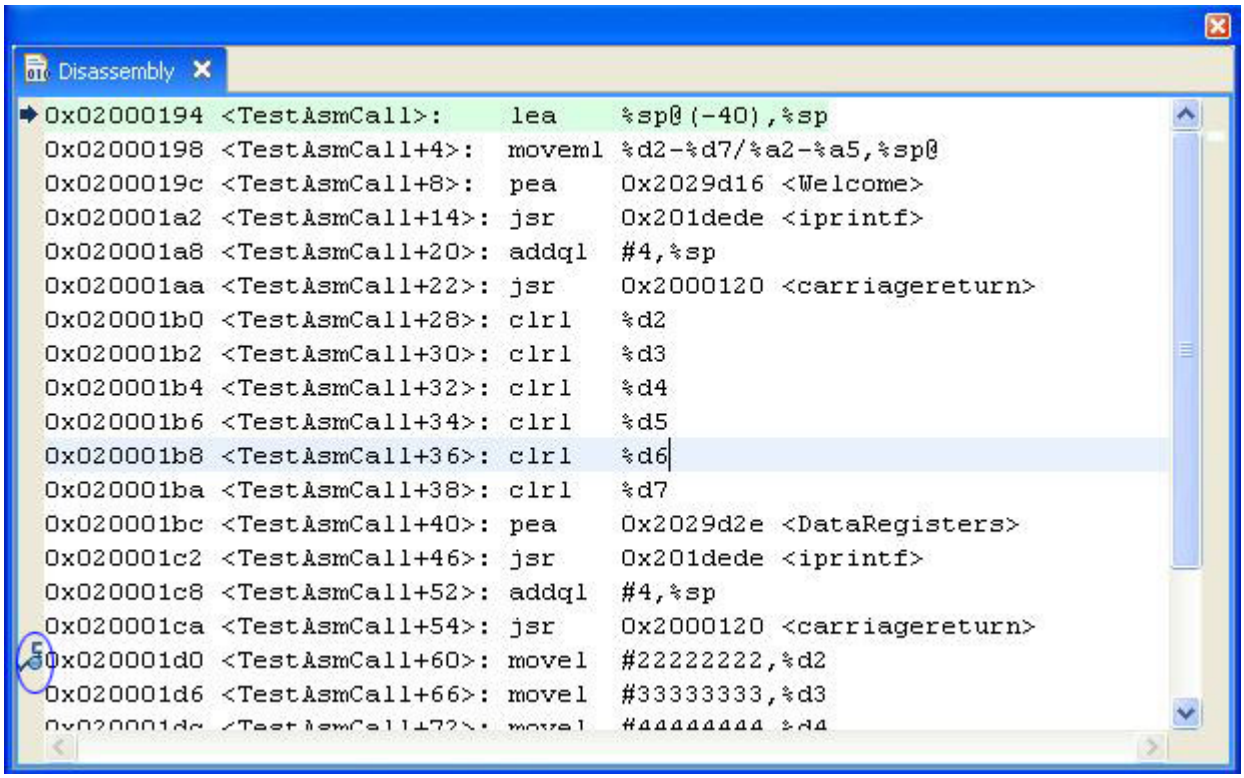


Notice, that our program is once again halted at "**TestAsmCall();**" The program ran once and has halted back at the breakpoint. Rather than step through this line of code, step into it by clicking on the "**instruction stepping mode**" and then the "**step into**" button. We should of step into our "**assembly.s**" file. In addition, the "**disassembly**" window should pop up with the machine/assembly language code.

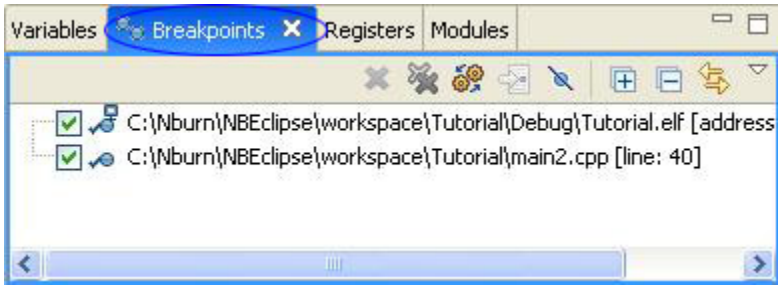
A screen shot is shown below.



Resize the Disassembly window lengthwise to the left or by double clicking it twice. We should now be able to view the contents more clearly. We can step through our machine language code one line at a time by clicking the "**Step over**" button. Try it a few times to get comfortable with it. You can also place breakpoints in the "**disassembly**" window by double clicking on the grey area next to the line where you want to halt the program at. Place a breakpoint at the line "**move.l #22222222, D2**". The screenshot below shows how do it.



As an aside, to confirm that we have two breakpoints, you can click on the "**breakpoints**" tab to see if it's present. There should be two breakpoints, one at "**TestAsmCall**" and one at "**move.l #22222222, D2**".



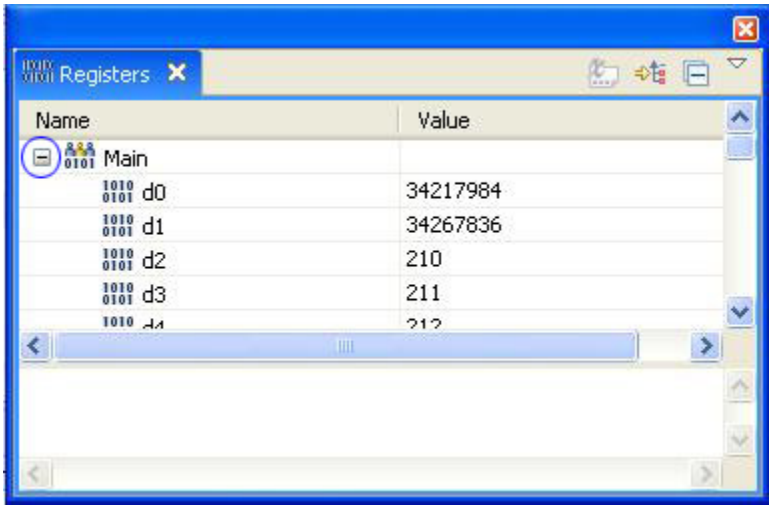
Now click on the green "**resume**" button a few times to observe what happens. You should see the program halt at each breakpoint. This is how we debug our program by placing breakpoints in our code where we want to stop our program and stepping through line by line of our code.

Register and Memory Content

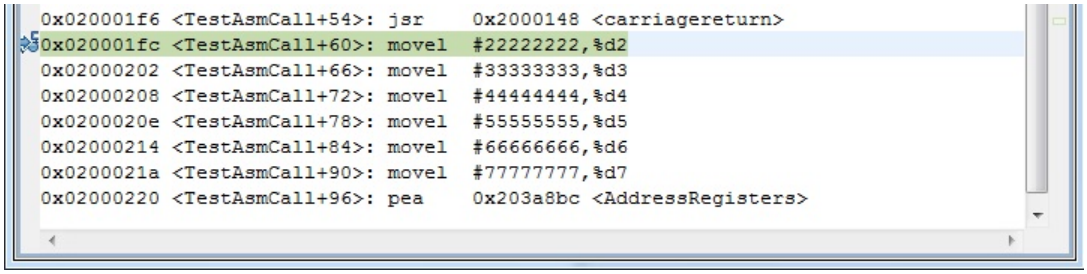
Now that we know how to halt our program at specific points in our code, there are tools available for monitoring the Register and Memory Content.

Register

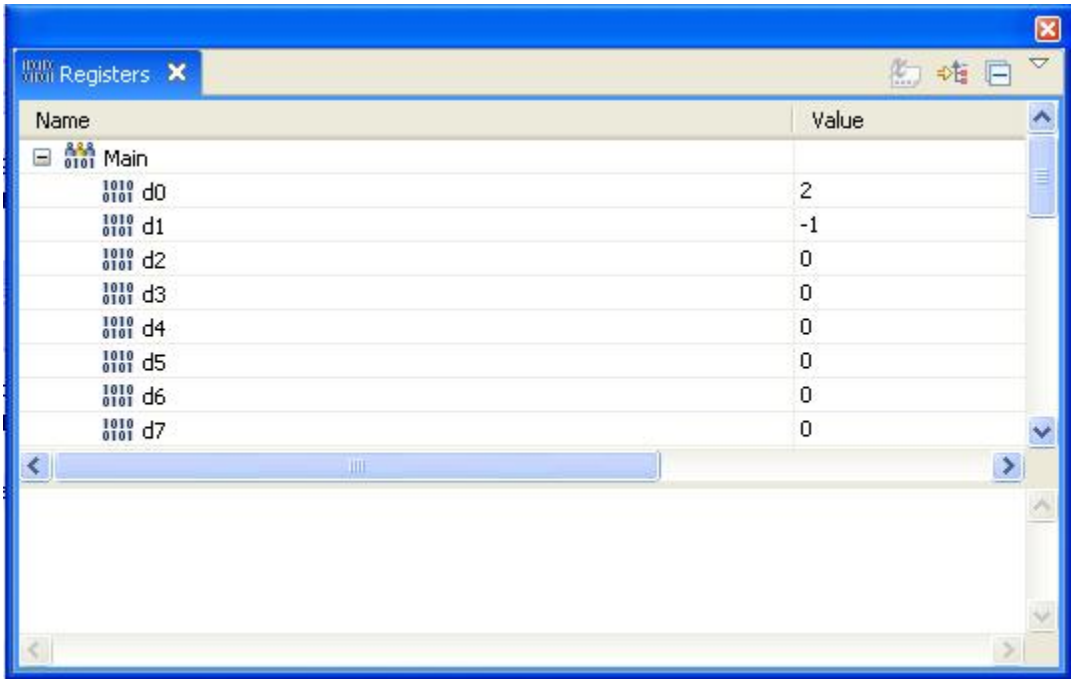
To see the register contents, click on the "**Register**" tab. Click on the '+' beside the name "**Main**" to expand it. As you are stepping through your code, the contents of the registers will get updated accordingly.



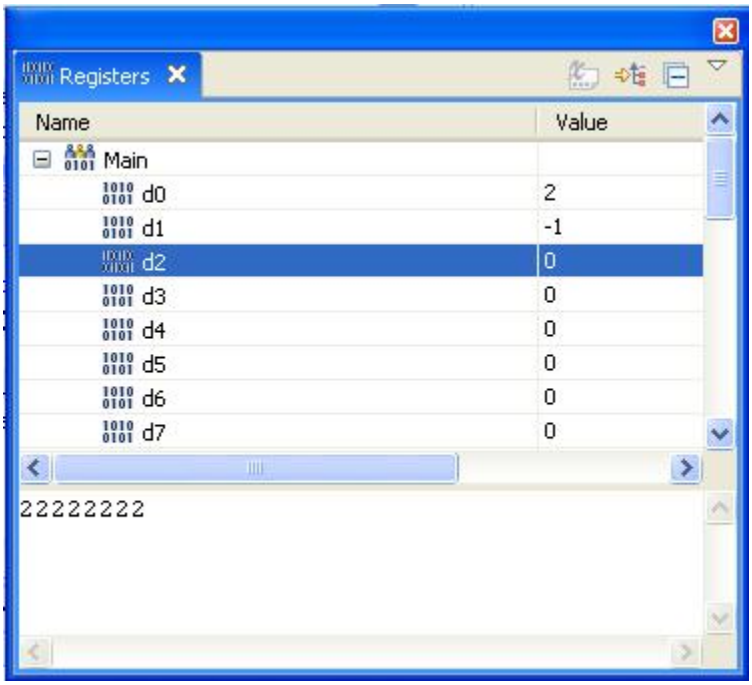
Step over each line of code and watch the register value change. We had previously placed a breakpoint at "**move.l #22222222, D2**". Run the program until you reach this breakpoint.



Now bring up the Register Window. Registers D2-D7 should have been cleared to values of zero.



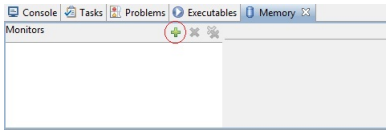
Now try stepping over one instruction at a time. Step over the instruction "**move.l #22222222, D2**" and monitor the Register D2.



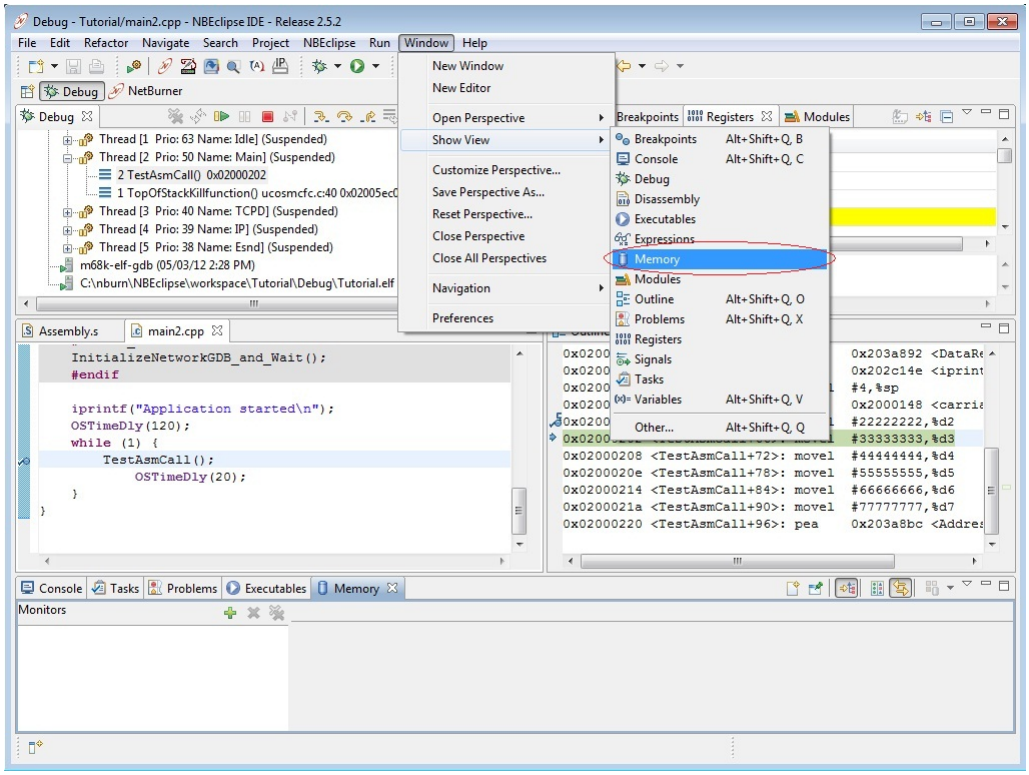
Notice the value of D2 is now displaying "**22222222**".

Memory

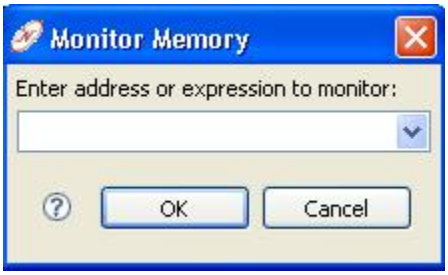
A "**memory**" box should open up in the bottom left hand corner.



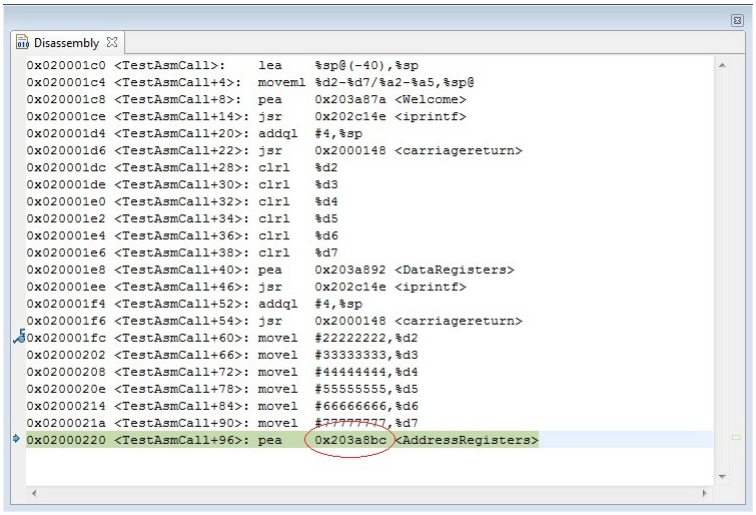
If you do not see it, you can bring it up by clicking on "**Window\Show View\memory**"



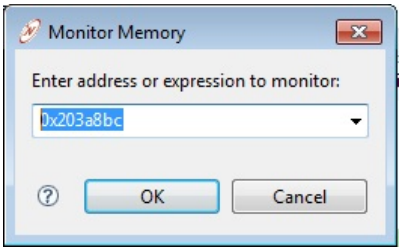
Click on the green "+" sign to add the memory address in which you want to monitor.



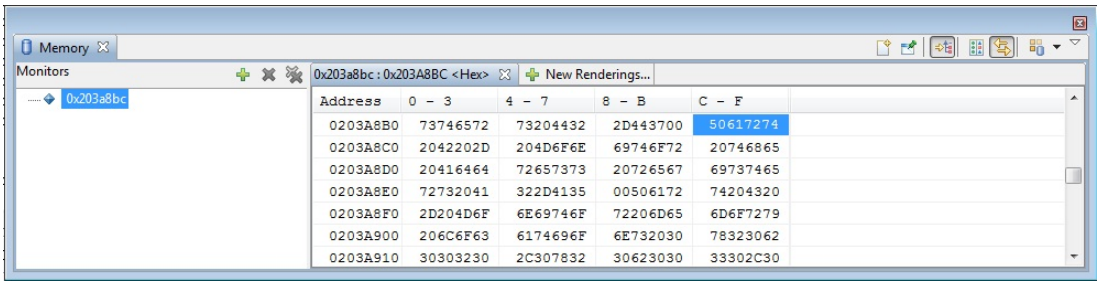
In our example, we will monitor the memory address of where our data is stored. In order to do that, we must first locate where our data is stored. Open up the "**Disassembly**" window where our machine code is located and find out what address "**AddressRegisters**" is located at. In this example, the data "**AddressRegisters**" is located at memory address "**0x2029d58**".



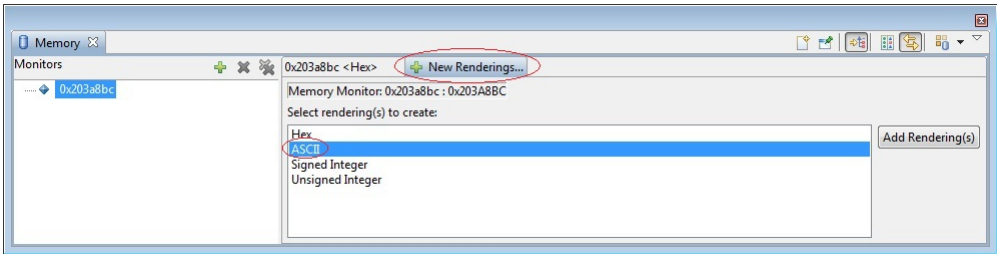
Enter this value in the memory monitor box.



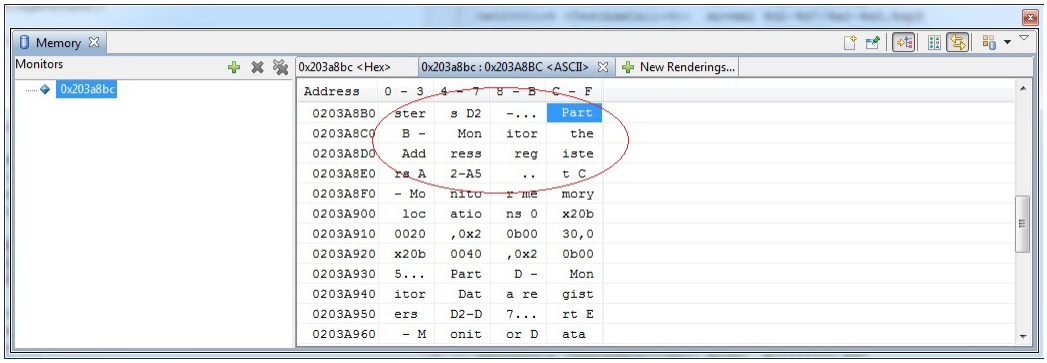
Click "**OK**" to confirm. Our memory address should now be displayed in the memory monitor along with the contents.



In our example, the memory address 0x2029d58 contains that data "**50**"'. Although this may not seem apparent, but the data "**0x50**" represents the ASCII character "**P**". To see the data more clearly, click on the "**New Rendering**" tab and select "**ASCII**" .



The data is now displayed in Ascii format instead of Hex. In our example we can see the text "**Part B - Monitor the address registers**"



This concludes our tutorial.

Appendix

```
1.          #include “predef.h”
2           #include <stdio.h>
3           #include <ctype.h>
4           #include <startnet.h>
5           #include <autoupdate.h>
6           #include <NetworkDebug.h>
7
8           extern “C” {
9           void UserMain(void * pd);
10          }
11
12          const char * AppName=“Tutorial”;
13
14          void UserMain(void * pd) {
15                  InitializeStack();
16                  OSChangePrio(MAIN_PRIO);
17                  EnableAutoUpdate();
18
19                  #ifdef _DEBUG
20                  InitializeNetworkGDB_and_Wait();
21                  #endif
23            iprintf(“Application started\n”);
24            while (1) {
25            OSTimeDly(20);
26                      }
27            }
```

Lines 1-6 specify the include files. These are library functions that must be included in order for use in program. Example, “stdio.h”’ defines standard input and output functions such as “iprintf”

Line 8-10 tells the C++ compiler to declare the UserMain() function as a “C” type function call. This is done to allow straight C programming as opposed to C++

Line 12 - Defines the application name

Line 12 - Defines the application name

Line 14 - Declares the UserMain() function.

Line 15 - Initializes the TCP\IP stack

Line 16 - Sets the task priority to the defined value of MAIN_PRIO, which defaults to 50.

Line 17 - Enables the code development AutoUpdate feature using “make load”

Line 19-21 - Configures the application to accept a debugger connection.

Line 23 - Prints text to monitor