Integrated Development Environments (IDEs)

An **integrated development environment (or IDE)** is a software application that helps to facilitate the writing of software. IDEs contain code editors, build tools, and debuggers all contained in one easy to use package. Many IDEs work with more than one programming language, giving you a tool that has a broad range of use.

You probably used to play with blocks when you were a kid (or still do, nothing wrong with that). IDEs are often used to do visual programming. Visual programming is a form of programming in which premade controls (buttons, text box, etc.) are used to build a form just like building a house with blocks. Ever wonder what the visual in Visual Basic and Visual Studio meant, well now you have a good idea what the Visual in Studio is.

IDEs are most often (but not always, as they can be used for many things) used to create GUIs or graphic user interfaces. A visual and hopefully user friendly front end of an application. But do not forget that an IDE can also be used to create many other things include console applications (command line), libraries, classes, etc... Now that we have had a look at the broad side of an IDE let us look into the general parts of an IDE while we are here (no fun you say, well…. do it anyway).

Table 1-1 displays the basic parts of most IDEs (not every single IDE is the same, these are just normally included basic parts).

Table 1-1

BASIC IDE COMPONENTS		
Source code editor	Build tools	Debuggers
Usually a text editor quite often with built in code completion that helps to create, modify and arrange code.	Automation and build tools that help to streamline the build process of software.	Built in utilities that help to track/trace bugs. Also giving you helpful errors and warnings about your source code.

The goal of an IDE is to streamline the process of software production maximizing programmer productivity. If you think about building software using a text editor, then using a build tool, and finally compiling and debugging, all built using different applications takes time. An IDE incorporates all of these tools together giving you an incredible productivity boost.

You may well notice that IDEs are often called or classified as other types of software applications. This may well be because certain tools are included/missing, or that the application has a more specific purpose, etc... A few examples can be seen in Table 1-2

Table 1-2

AN IDE BY ANY OTHER NAME....	
Abbreviation	**Description**
GIDE(game integrated development environment)	An IDE meant to create and design games. Include specialized libraries and tools for games application development.
SDK(software development kit)	Include necessary building blocks for developing applications. This includes frameworks, libraries, header files, as well as compilers, debuggers, and various other tools. Unlike an IDEs which often require an SDK to be installed, they can be used on their own.
RAD(rapid application development tools)	Often used interchangeable with IDE, and usually mean the same thing. Basically a set of tools that speed the process of software development, just like and IDE.

A closer look at IDEs

Now that we have look into the basic parts of an IDE let us take a look at a few more in depth examples. We will be taking a look at two IDEs for each language C++, Java and C# (.Net). One example will be a larger more complex IDE, while the other will be a smaller streamlined type of IDE. We will go over the same basic parts to help you build a little familiarity with each. So enough talking, start getting to the point.

Parts covered for each IDE

1. Creating a new project
2. Code our project/ common tools location
3. Running/Building projects
4. Location of debugging tools
5. Pros and cons of each IDE

C++ IDEs-Visual Studio, Code::Blocks

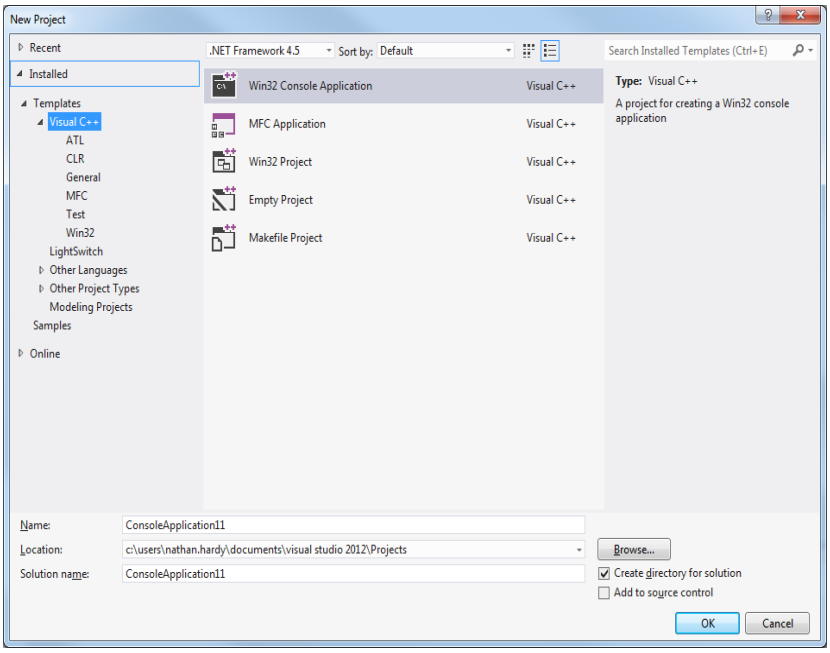
Visual Studio _____

The quintessential IDE if you are into .Net programming of any kind (and you like hugely bloated software). You might be surprised to find building a C++ application in Visual Studio is quite different from a C# application. We will dive into Visual Studio to look at the features (deeply buried or otherwise) and try to help you feel more at home there.

1. Creating a new project

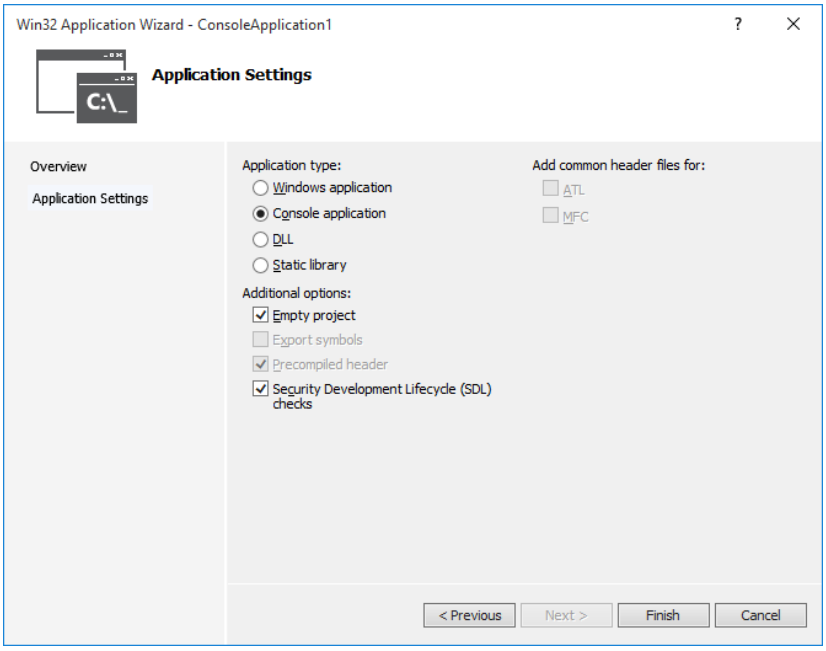
First let us start by creating a project (you can't code thin air after all) and see what all the fuss is about. Like all projects in Visual Studio, a new project starts with File->New->Project. Figure 1-1 shows us our options for a Visual C++ project

Figure 1-1



You may notice there are not as many options as a creating a basic VB or C# form. Making a GUI in C++ requires special tools (in Visual Studio this is translates as an MFC application or Microsoft Foundation Classes). But today we will create a blank console application and carry onward. You will notice that a new console application wizard opens up. We want to have some say in what goes into our program so we choose next instead of finish. You can see in figure 1-2 that we want to check empty project, leaving all the other options as they were.

Figure 1-2

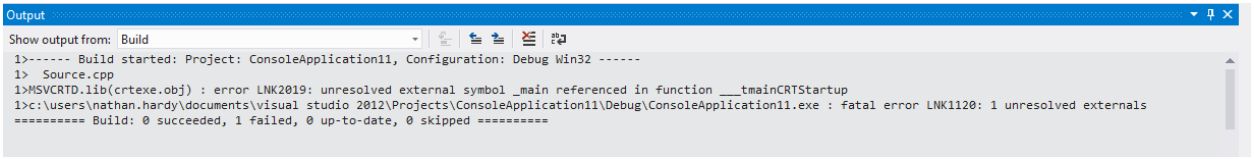


But wait a minute, nothing is here to program, so now what. Well we wanted an empty project and we got one all right. But if we right click on the program name in the solution explorer and go to Add->New item and choose a C++ file, there we go. Now you are free to program away to your heart's content.

2. Coding our project

Now we can start to build our program from here. One of the first things you may notice is that there are no easy drag and drop controls to add to our program (welcome to C++ console programming). Since we are creating a console application there are no GUI controls to drop in and help us along the way. In fact you are presented with completely blank file, no extra code at all unlike the VB forms you are used to. If you run a VB/C# windows form, the basic code is there and the window will pop up (empty but still works anyway). If you try to run your C++ program you get an error like in figure 1-3 (complaining about unresolved symbols).

Figure 1-3



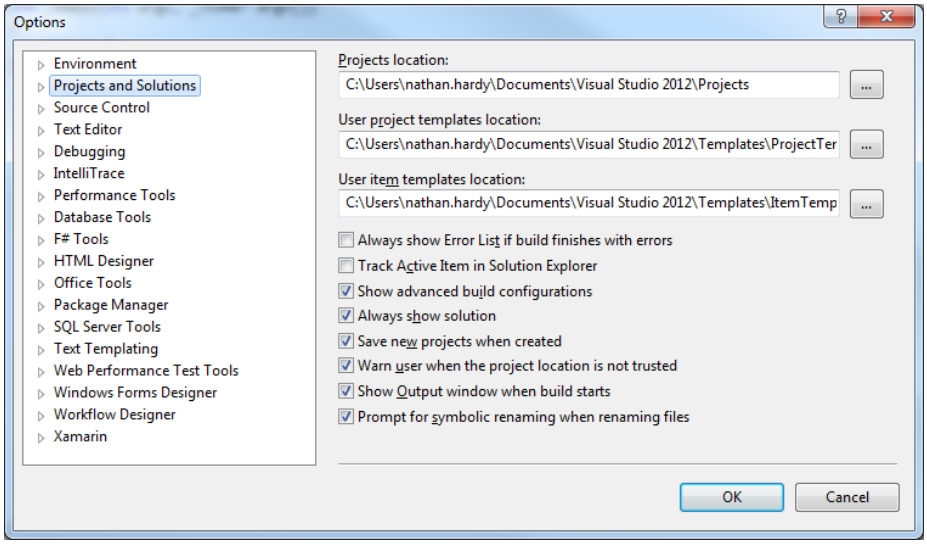
If you go into C++ programming such unhelpful error messages will be a normal occurrence so don't sweat it. What the Debugger is really trying to tell us is no code is there to run, so I will add some basic code for it to find like in figure 1-4

Figure 1-4

```
1  #include<iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      return 0;
8  }
```

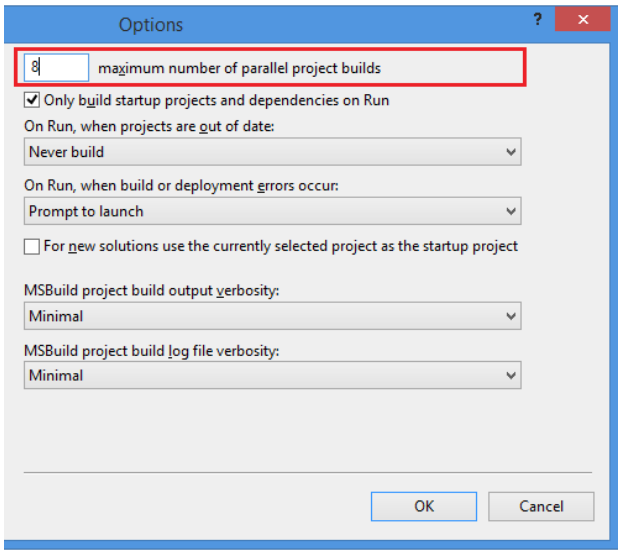
Since we have no component tools to get to know let's take a look at a few other features for Visual C++ that come in handy. First off when you start building fairly complex C++ projects you will begin to notice that compiling them takes quite a while. The more complex the longer they take, so speeding them up sure would be nice, right. So click on Tools->Options which will open up Visual Studios many, many options. Figure 1-5 shows you all the options that exist inside the recesses of our IDE.

Figure 1-5



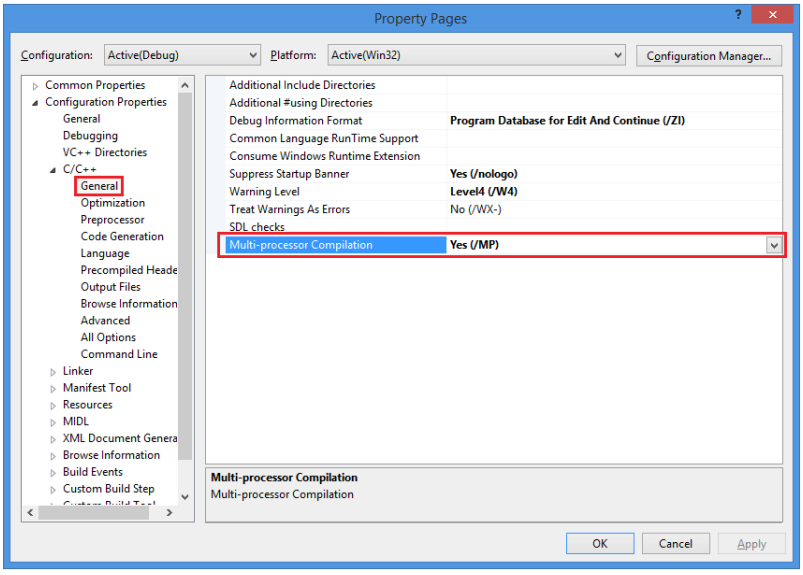
First off one the first options to check out the options inside of project and solutions (click the arrow). Figure 1-6 displays the sub-options inside that we will be tweaking or checking.

Figure 1-6



Notice the option that is highlighted, this allows you to increase the number of parallel builds, increasing this to around 8 allows applications to be built in parallel, speeding compilation times. Next, we will look at one more option for C++ that will help to speed things up a little. Figure 1-7 shows the options inside of Configuration properties->C/C++->General.

Figure 1-7



This particular option is turned off by default, so you will have to turn it on if you like. This option enables the use of multi-processor compilation (enabling the computer to use the 2, 4, 8 etc... cores to increase the speed of compilation). Oh and since we are here in options already Text Editor->All Languages->General and checking the line number box turns on line numbering in Visual Studio.

3. Running/Building

Now we can actually run our project, but you may notice the absence of the start button in your project. Instead this is replaced by a strange Local Windows Debugger button. Clicking this particular button compiles and runs your program. But wait a second, the program closes almost instantly, what is going on exactly. Well using the local debugger and having the no pause or input required means the program has run in full. You can get around this problem by going into Debug->Start without debugging.

4. Location of debugging tools

When building and programming C++ many interesting things come up, so make friends with the debugger (and save yourself a headache). Debugging will be covered later on (in another chapter) so we will only talk about a few basics and give you the location of such tools.

First let's talk about a few interesting things inside the build tab in Visual Studio, and by that I mean build, rebuild and clean solution. These options tend to pop up more in C++ than other languages, not that you can't use them for C#, etc…. Table 1-3 shows us the difference between the three.

Table 1-3

Build solution	Rebuild solution	Clean solution
Will perform an incremental build: if it doesn't *think* it needs to rebuild a project it won't. It may also use partially-built bits of the project if they haven't changed.	Will clean and then build the solution from scratch ignoring anything it's done before.	Will remove the build artifacts from the previous build. If there are any other files in the build target directories (bin and obj) they may not be removed but actual build artifacts are.

The debugging tab contains most of the other debugging tools (not surprising if you think about it). As this will be covered later on, we will move onward and upward.

5. Pros/Cons

Pros----------

1. Stable, well supported IDE
2. Good selection of tools and add-ons
3. Quite user friendly (after a slight learning curve)

Cons----------

1. Large learning curve for new users

2. Bloated software size (installing Visual Studio is getting towards 6+ gigs)
3. The code written by the IDE is sometimes faulty (not the code you write, but the automatic code generated by the IDE)

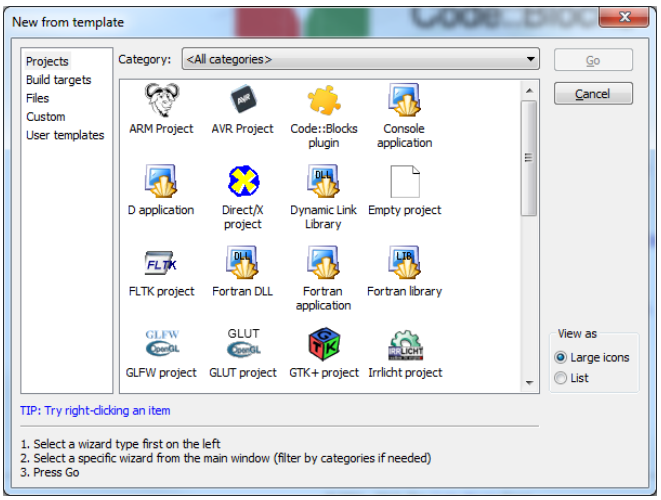
Code::Blocks_____

A free and open source cross-platform IDE that supports many different compilers. Largely geared towards C\C++ but has support for FORTRAN as well. Code::Blocks can be used for Windows, Linux and Mac OS X, as well as a few other platform choices.

1. Creating a project

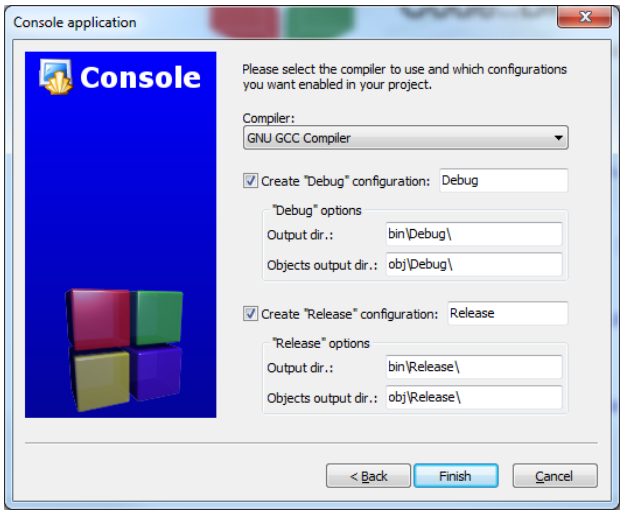
Code::Blocks has an interface not dissimilar to Visual Studio, so you should have a good idea of how to get around. So just like VS, creating a new project is accomplished be File->New->Project…. Figure 2-1 shows the templates available for your projects, this might be a little confusing as there are quite a few different options.

Figure 2-1



But keep in mind these are similar things you would find in Visual Studio, just not displayed all at once. We will continue on by clicking on Console application (top right choice), and begin creating a new project. Clicking on the go button takes us to the new console application wizard. Clicking on next leads us to choose which types of console application we want (C or C++), make sure C++ is highlighted and click next. The next screen allows us to name our application and file. Type in something and click next to continue. Figure 2-2 shows use the options for compilers, not something that Visual Studio gave us before. This is because Code::Blocks is cross-platform, or can be used on different OS's.

Figure 2-2



If you click on the drop down list of compilers you will see many, many options including the Visual C++ compiler. But we want to run the GCC compiler (standard compiler for Linux platforms) for our console. All the options are good so we continue by click on finish. Now clicking on the + sign next to sources shows us our C++ file (you may have to double click to see it).

2. Coding our project

Now we can code our project, although you may notice that Windows specific code does not work (as we are compiling with the GCC compiler). You may also notice that the basic program does not have to be paused, or ran without debugging when we run it. Coding in Code::Blocks is the same as Visual Studio, you simply type code inside the text editor and voila. Code::Blocks also features code completion allowing you to type in the start of a piece of code and have the IDE complete it for you. Our basic hello world application is ready to be ran/built. Again, since we are building a console application tools are not too important. One thing to note about Code::Blocks is adding new files is not as easy as Visual Studio (a new C++ file or header can be added by File->New->File).

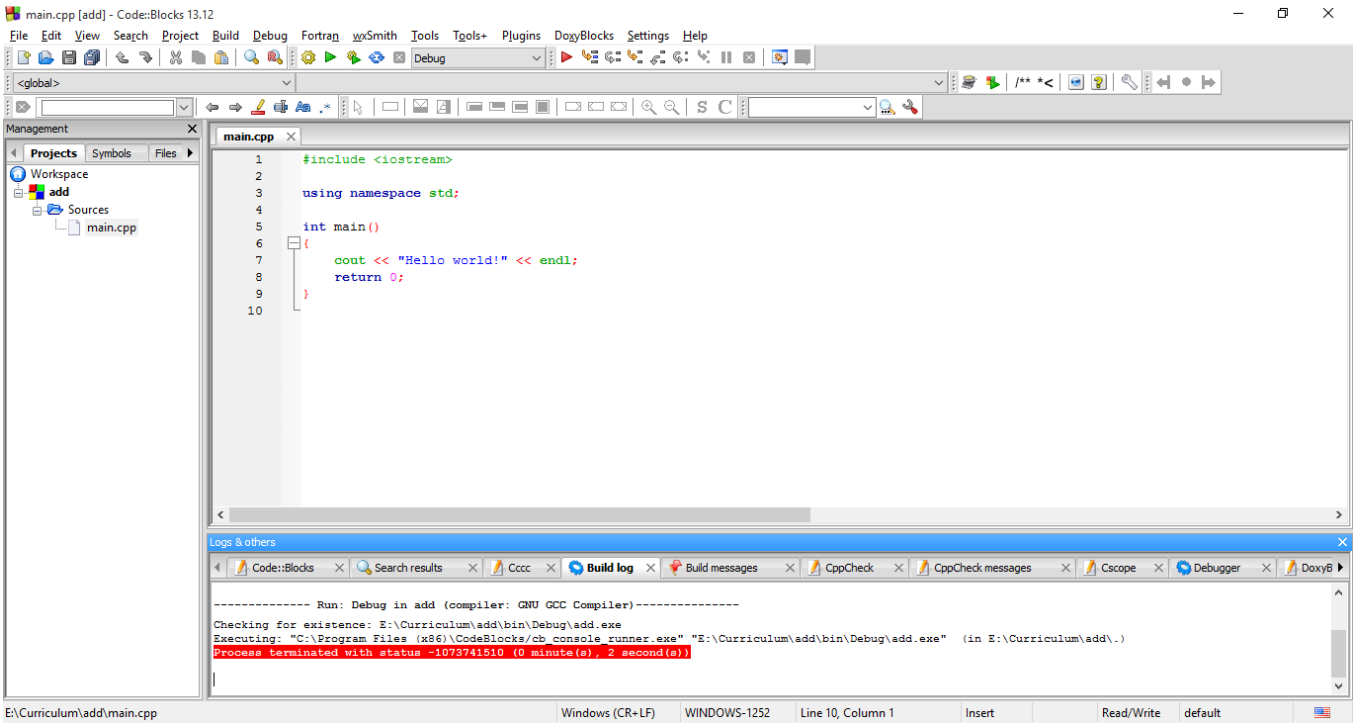
3. Running/Building

The distinction between build and run is less clear in Visual Studio, but is more defined in Code::Blocks. Again this can be traced back to the fact that Code::Blocks in meant to be cross-platform. So even if you decide to run your application without building it first, Code::Blocks warns you about this fact. Building and running are distinctly different, and will be covered more in the next module. For now clicking on the run arrow or going into Build->Build and run compiles and runs our basic program

4. Location of debugging tools

The first and most important debugging tool in Code::Blocks are the log windows located at the bottom of the screen. Figure 2-3 displays the log windows with the build log tab selected at the bottom of the screen.

Figure 2-3



Any and all messages (good or bad) that apply to your program will pop up in a log (build, debug, etc…). Other debugging tools, such as break points appear inside the debug tab just like Visual Studio.

5. Pros/Cons

Pros----------

1. Lightweight and easy to use
2. Cross-platform
3. Integrated with a GUI design framework (wxWidgets)

Cons----------

1. Only supports C++, FORTRAN
2. Not updated as frequently
3. Open source (this can be good, but also can be bad as people volunteer time resulting in delays, bugs, etc…)

Java IDEs-NetBeans, jGRASP

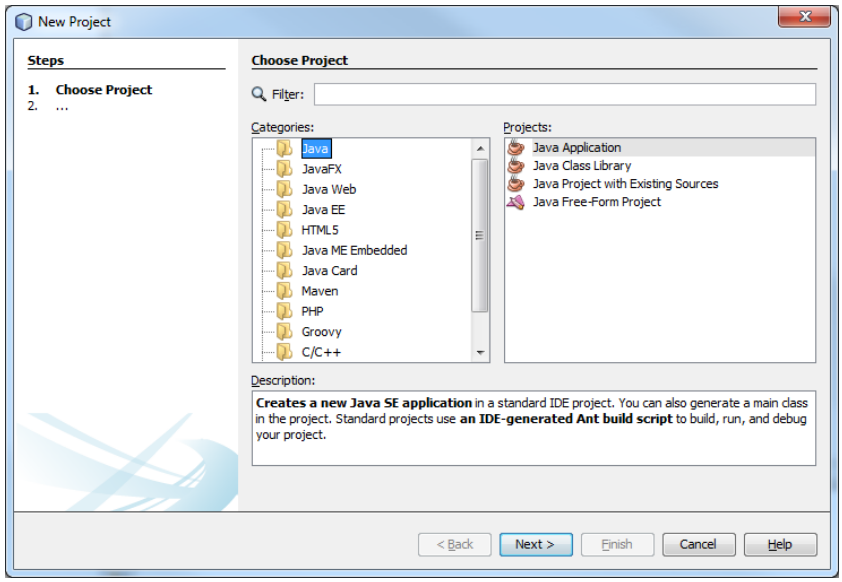
NetBeans_____

Now we move onto spilling the beans on NetBeans if you will forgive the pun. NetBeans is one of many good Java IDEs available for your coding needs. A highly customizable and complex program that can rise to meet just about any of your Java problems. NetBeans supports many languages besides Java such as C/C++, PHP, and HTML.

1. Creating a project

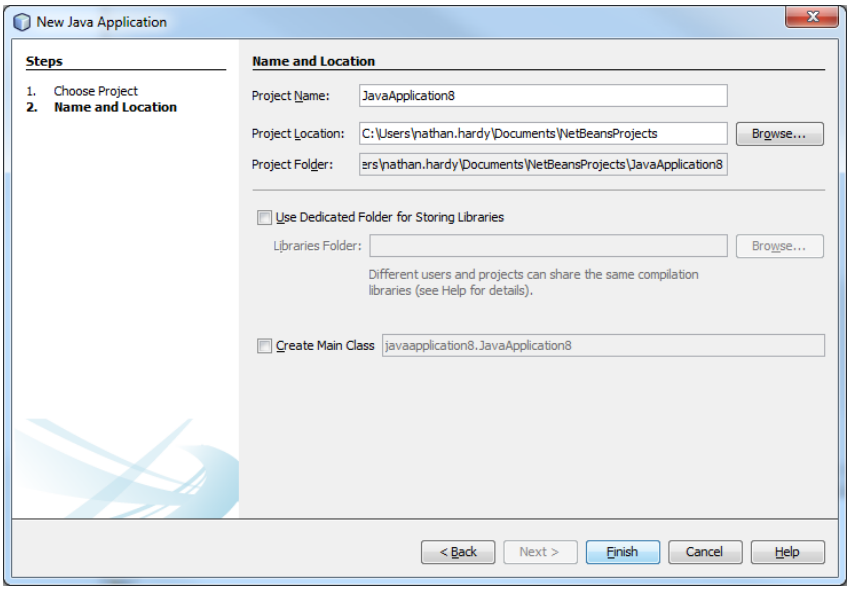
You may begin to see a pattern developing here if you look close (most IDEs look and behave the same on a basic level). So by now you can probably tell me how to create a new project (I'm waiting, go ahead). Well just for good measure I will continue to teach you anyway. First off just like any other IDE click on File->New->Project, this opens a new windows where you can choose your project type you want to use. Choose Java and Java application as shown in figure 3-1.

Figure 3-1



The next window wants you to specify a name and location of your project. You can name the project whatever you want and specify a convenient location. Make sure to uncheck the create main class check box as shown in figure 3-2, as we will be creating a GUI using NetBeans built in designer.

Figure 3-2

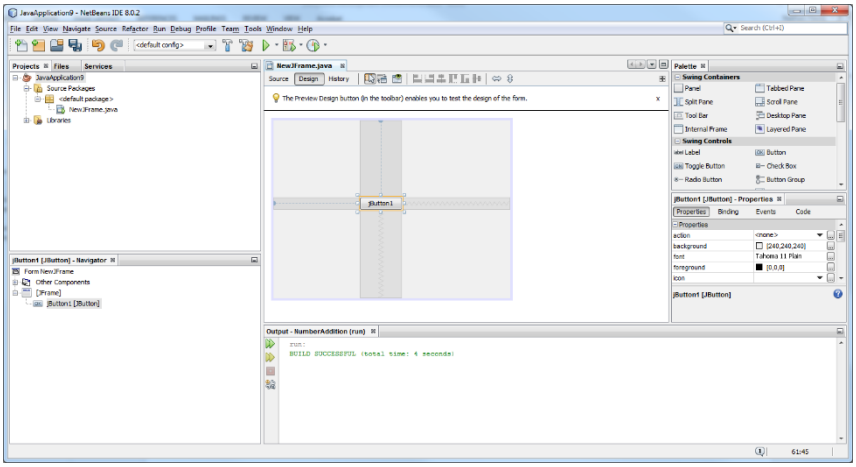


Continue onward by pressing finish, but wait nothing is here to code, so now what. Well we will pick that up in the next step.

2. Coding our project

Now we want something to add code to, so we need to right click on the name of our project in the projects window (upper left hand corner). Choose New->JFrame form and now we have a form to program on. Great now we can set up a basic hello world by first dragging a button off the palette (upper right hand corner) onto our form. Double clicking on the button creates an event handler in our code (Java version of button clicked event). Now click the source tab to see the code behind our form, figure 3-3 shows the windows mentioned (projects, palette) before clicking the source tab.

Figure 3-3



Locate your button event handler (in Java this is jButton1ActionPerformed, unless you renamed it) and add the following code.

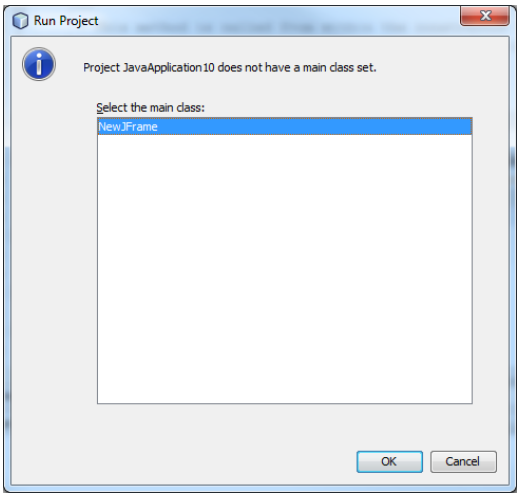
```
JOptionPane.showMessageDialog(rootPane, "Hello World");
```

Now that our program is ready to do something, we can move on to the next step.

3. Running/Building

Now we can start our program to see that sweet “Hello World” we all know and love. Getting there is easy, you can push the run button (looks like a triangle), or we can go into the run tab and choose run project. NetBeans will have a window that pops up complaining like in figure 3-4.

Figure 3-4



If you remember from the previous step we unchecked the “create main class checkbox”. Well now NetBeans wants to create one for us so it can run the project. Click ok and your application should pop up, click the button to see your code in action.

4. Location of debugging tools

Now we can continue to on and locate our debugging tools for yet another IDE (what complete and utter fun). Would you be surprised to find debugging tools inside of the debug tab (I know, so was I). Again, NetBeans like most other IDEs offer the same basic debugging tools. The debugging tab contains things like step over, into, and other options you use with break points. Another thing to take note of is the refactor tab (between source and run) in NetBeans. Refactoring isn’t directly a debugging tool, but does help find bugs. Refactoring is the process of restructuring existing code, without changing external behavior. Refactoring improves readability, reduces complexity, making it easier to locate bugs and problems.

5. Pros/Cons

Pros----------

1. Easy to adapt to your specific needs (huge choice in plugins)
2. Supports many different build tools (Gradle, Ant, Maven)
3. Includes samples for beginners (nice to have even if you are experienced)

Cons----------

1. Can be slow to load
2. Takes up more memory than lighter IDEs
3. Some advanced tools require research and/or training

jGRASP_____

A now for something completely different (but still and IDE after all). jGRASP is a tiny and lightweight Java IDE and is ideal for small quick coding needs (such as console applications). jGRASP can be used to produces CSDs (control structure diagrams) for Java, C/C++, Objective C, Python, etc… CSDs are just a document that shows the flow within source code (don't worry about that too much now). Keep in mind that jGRASP does not have a built in GUI designer (you can still program them, but you have to write all the code yourself). Now let's move on to the show and take a look at jGRASP.

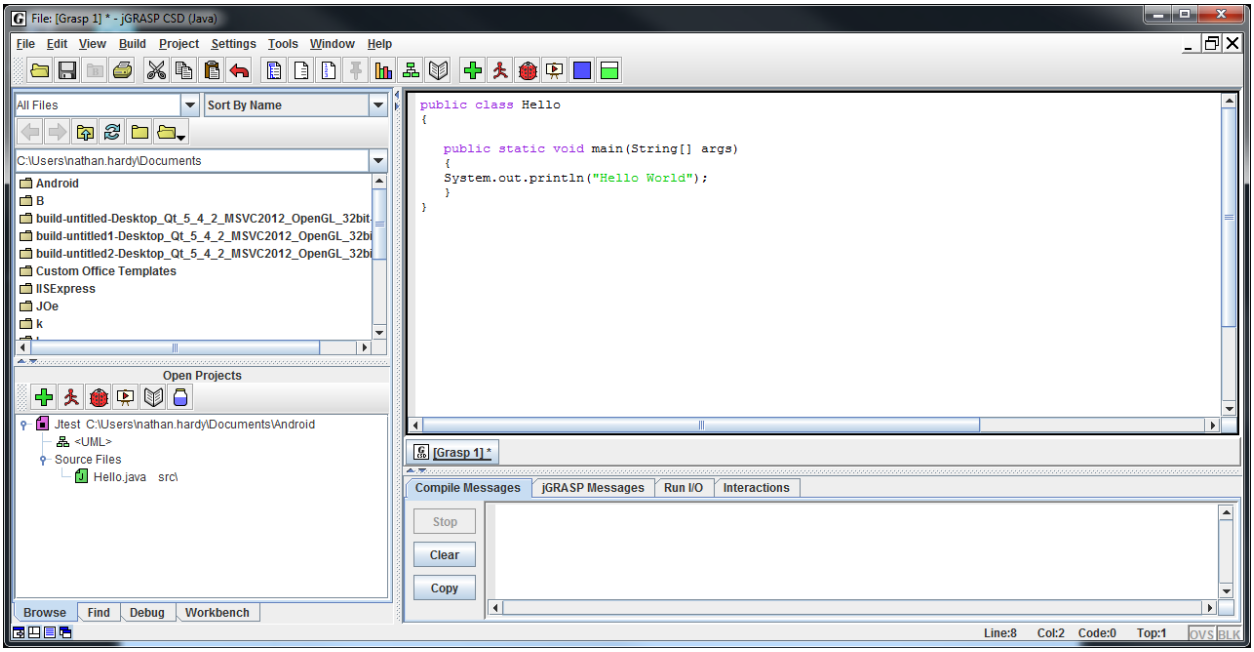
1. Creating a project

Creating a project in jGRASP isn't as straightforward as other they are in a few other more inclusive IDEs. We will begin by creating a file (not a project) to contain our code. Clicking on File->New->Java produces a blank file for the purpose of coding (what else would you do with it after all). jGRASP is a lightweight IDE and we will be building a console application, not a GUI.

2. Coding our project (file)

Now for the fun part you all enjoy more than anything, coding our Java file. We start out by typing in our class to contain our code (other IDEs often do this for you). A simple "public class Hello{" begins our class. Adding in "public static void main(String[] args){" continues our file by including a main function. Finally we add some code for our application to do "System.out.println("Hello world" and close each opening bracket. Figure 4-1 contains the completed file to view.

Figure 4-1



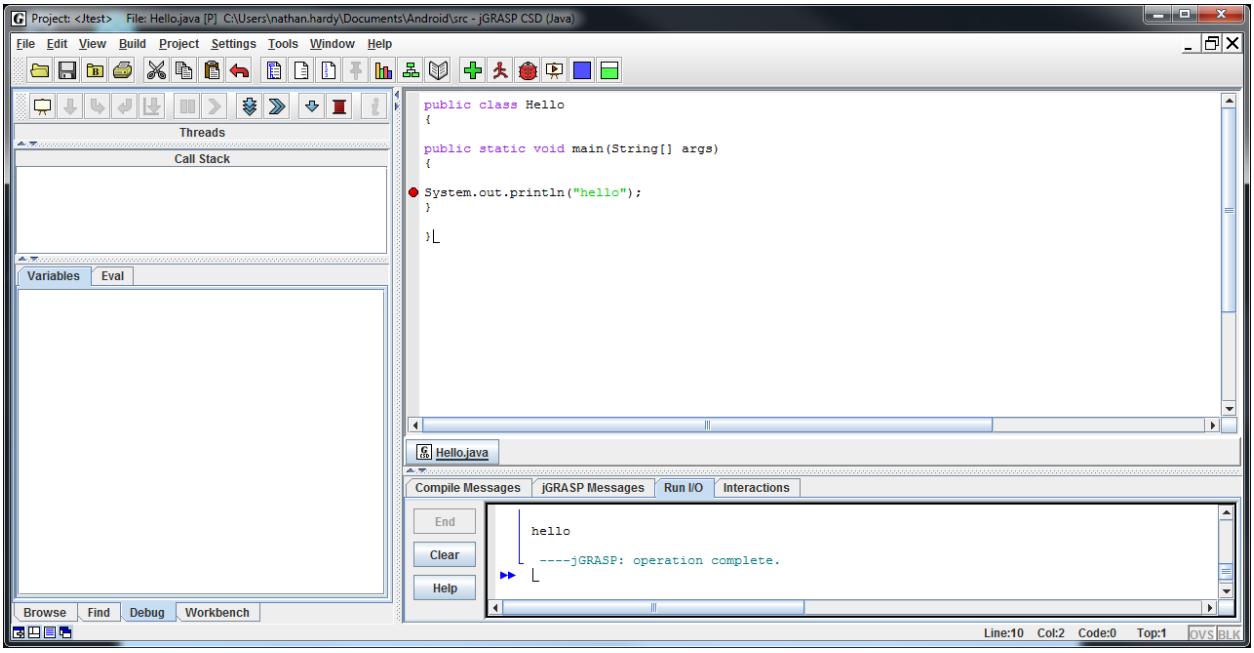
3. Running/Building

Well now that we have code what should we do with it? Build and run it would seem to be the most obvious solution to this problem. Clicking on the little green plus sign compiles our code (or clicking the build tab->compile). jGRASP should complain about saving it, go ahead, and the save as window will open. Clicking the ok button will rename our file after the class (in this case Hello). Now we can run our project to see another glorious instance of the “Hello World”. Now to run our code in jGRASP, we click on the little running man next to the compile button (makers of jGRASP must have a sense of humor). Or of course clicking on Build->Run accomplishes the same.

4. Location of debugging tools

jGRASP runs by default in debugging mode, you can turn this off if you really want. But why would you do that as it does not really hurt anything at all. You may notice the absence of the handy dandy debug tab in jGRASP. Indeed jGRASP is a more streamlined IDE and has no debug tab, does that mean there are no debugging tools but there are as most of your favorites are still there. Clicking into the Build tab reveals options like clean, debug, debug as application, etc. You can even set break points by clicking near the edge of your code. With a break point and clicking Build->debug, our program runs and stops at our break point. Figure 4-2 shows our program with a break point set.

Figure 4-2



5. Pros/Cons

Pros----------

1. Small install size
2. Easy to use for small projects
3. Can create CSD (if your into to that)

Cons----------

1. Less tool choices than larger IDEs
2. No code completion
3. No GUI builder

C#/.Net IDEs-Visual Studio, SharpDevelop

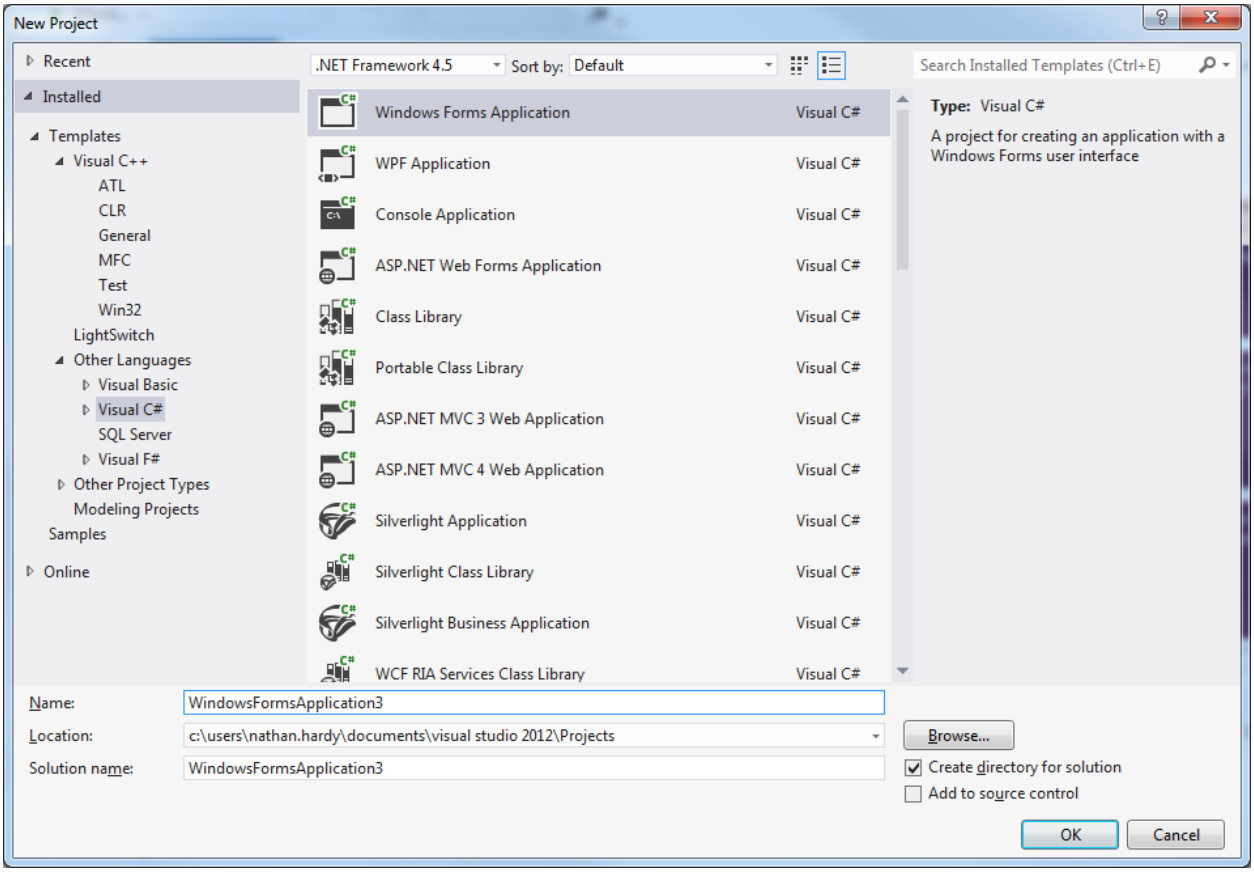
Visual Studio_____

Again we find all roads lead to Microsoft (well at least in the .Net world anyway). For C# programming that leads us back to Visual Studio. Programming in C# and Visual Studio is a very similar process to Visual Basic. We still have the power of .Net behind us making our coding life a little easier. Since we already talked about Visual Studio we can dive right into it.

1. Creating a project

Starting a project in Visual Studio is as simple as File->New->Project. But if you have Visual Studio set up for C++ or VB, our C# project options may be hidden inside of other languages. Let's start by choosing a Window form application and see where it takes us. Figure 5-1 shows us choosing windows form (Visual Studio is set for C++ and C# is hidden in other languages).

Figure 5-1

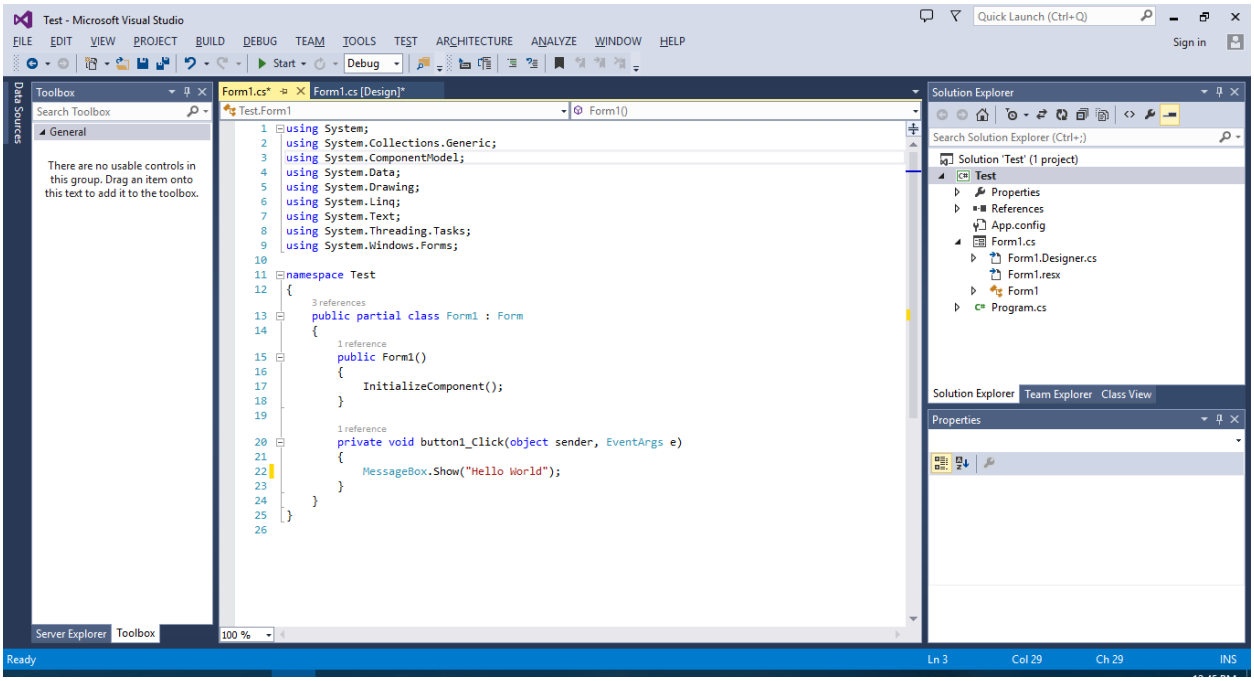


Clicking on ok creates our basic Windows form just like in Visual Basic. So let's move on to coding and have us some real fun.

2. Coding our project

We will drag a button onto our form to give our application a little something to do. Double clicking our button creates an event handler in which we will add some very original code. Inside of our button click event we will add the following code "MessageBox.Show("Hello World");" This causes our program to do something quite mysterious and interesting. When someone clicks the button, a message box pops up saying hello world (yeah I known, we done this 40 times by now). Figure 5-2 shows the final product if you care to take a look.

Figure 5-2



3. Building/Running our project

Now just like running a Visual Basic program, clicking on the start button runs our project. So nothing new to see here for us (except the C# code of course).

4. Location of debugging tools

Same tools just different language, so debug again is the prime location for all things debugging. But don't forget that build tab includes build, clean, rebuild and etc. Not as needed as when doing C++ programming but still good to know that they are there anyway.

5. Pros/Cons

Pros----------

1. Stable and well backed IDE
2. Many tools and plugins
3. Official IDE for Windows (maybe not a pro, but you can't discount this fact after all)

Cons----------

1. Massive install size
2. Not portable (or cross platform)
3. Can be intimidating for new users

SharpDevelop_____

A free and open source alternative to Visual Studio, not that there are many alternatives to choose from. Supports several languages such as VB, C#, Boo, IronPython, and IronRuby. SharpDevelop was meant to be an alternative to the express versions of Visual Studio. Sharp has equivalent features to almost anything found in Visual Studio Express. You will find that Sharp can do almost anything Visual Studio can do but with a more streamlined approach.

1. Creating a project

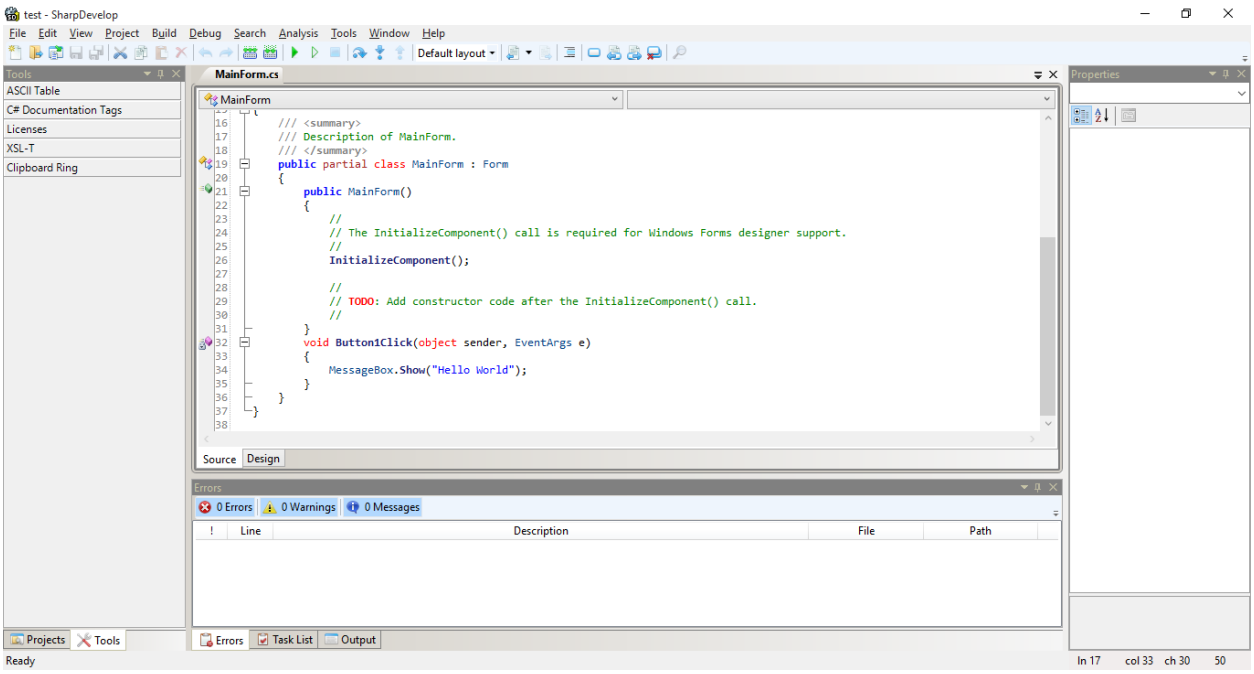
The Sharp interface is almost identical to Visual Studio so you should have no problem finding your way around. But since I am a nice tutorial I will help you get started anyway. Creating a new project in Sharp is the same basic concept as Visual Studio. Clicking on File->New->Solution creates our application, but wait don't we normally create a project instead of a solution. What's the difference anyway you may be asking. Well you can think of a solution as a container for projects, so a solution can have many projects. Sharp is kind of backwards in this instance, so we create a solution and carry on.

2. Coding our project

Next we continue forward by adding a button and the appropriate code to make it do something. You may notice that Sharp has a few quirks such as the source and design tab at the bottom of your form. These tabs let us switch between code and designer on the fly (something Visual Studio lacks in my opinion). Switch to the design tab and pull a button from the tools window (left side of screen, if you

don’t see hit View->Tools). Double clicking on the button produces the same effect as Visual Studio, by that I mean an event handler is created for your button. Switch to the source view and find your button event handler, placing in the same code as our Visual Studio C# example. Figure 6-1 shows SharpDevelop in with the source tab chosen and the code in place.

Figure 6-1



3. Running/Building project

Sharp is more like many other IDEs in that the run button is a green arrow (no not the super hero).But just like Visual Studio you can go into the Debug tab and hit run instead. The other arrow runs the program without debugging (in Visual Studio these options are hidden in the debug tab).

4. Location of debugging tools

Just like Visual Studio the debug options are hidden inside the debug tab (and build contains clean solution, rebuild solution, build solution). So nothing is really out of place or changed enough that you can’t figure it out.

5. Pros/Cons

Pros----------

1. Lightweight
2. Free and open source
3. Better implementations of certain tools, etc.

Cons----------

1. Less defined support
2. Not all features available
3. New features implemented slowly

Summary

In conclusion, IDEs are an important part of programming for any modern programmer. They make the life of anyone who produces code easier. Additionally they reduce time needed to produce useable applications for users. There many good reasons to use an IDE, but remember that they are not always needed for everything. After all you don't use a chainsaw to open a can of soup (you laugh but this is applicable to coding certain things with an IDE). So stop and think about your needs before you open up that IDE and start programming.