Eclipse.org CDT and Cygwin: A Tutorial on Installation and Functionality

Christopher T. S. Allen
Department of Computer Science and Statistics
University of Rhode Island - Undergraduate

Abstract

The purpose of this tutorial is to provide a step-by-step guide to installing the Eclipse.org CDT as well as the cygwin linux to windows API. Also covered in this tutorial is the basic functionality of C++ programming provided by the Eclipse.org CDT IDE. Such functionality includes: the editor, the debugger, and managed makefiles.

1 Introduction

In this tutorial you will find all that is necessary in order to create, edit, compile, and debug a C++ program making use of the tools provided by the Eclipse.org CDT IDE.

1.1 What is the Eclipse.org CDT?

To begin, the Eclipse.org IDE is a java-based, platform-independent integrated development environment. The Eclipse.org CDT is the C/C++ Developers Toolkit extension to the Eclipse.org IDE. The CDT provides Eclipse the capability to create and handle makefiles so that they do not need to be written by hand, as well as providing the option to make use of user-defined makefiles. The parsing engine used by the CDT allows for convenient syntax highlighting while the indexer provides the novice C++ programmer convenient references to different parts of their code in the form of auto-code completion while at the same time allowing the more advanced C++ programmer quick code completion for long references to various variables, which, in most cases, helps alleviate some of the tendencies towards sloppy coding habits. Unlike many other IDEs, this particular option can easily be turned on and off while you code, so that, should it annoy you, its functionality can easily be disabled.

1.2 What is Cygwin?

Cygwin is, to use a direct quote from their site:

> The Cygwin tools are ports of the popular GNU development tools for Microsoft Windows. They run thanks to the Cygwin library which provides the UNIX system calls and environment these programs expect.

Because Eclipse is a compiler/platform-independent IDE, a compiler, as well as other necessary tools such as make, need to be supplied. Cygwin does just that. It supplies us with the GNU toolset, which, among other things, includes the GNU C Compiler and the GNU make toolset, as well as a plethora of libraries that can be accessed and used. In addition to this, the cygwin environment provides excellent UNIX capability support. Meaning that programs built by the compiler supplied and using the libraries found in the cygwin environment, should, without any tweaking, build on just about any flavor of UNIX. This includes, but is not limited to, the Mac OS and the innumerable Linux distributions.

1.3 Why not some other IDE?

Given that the Eclipse.org CDT IDE has many tools necessary for C/C++ development easily accessible from the interface, it is the ideal IDE for multi-platform development. At the current time, a number of other cross-platform IDEs exist; however, none really, at least in the authors opinion, are on par with the cleanliness of the Eclipse.org CDT IDE. This is, of course, to say that this estimation is only within the realm of open-source IDEs. At the moment, the Eclipse.org CDT IDE's biggest competitor, in the realm of multi-platform supported IDE's, is the Code::Blocks IDE. Unfortunately, this IDE still has very poor support for the Mac OSX and Linux community and thus is not the ideal compiler for multi-platform programming.

2 Installation

Installation is possible on the various platforms that have been mentioned thus far; however, in the subsections to follow, the majority of the installation instructions are related to the Microsoft Windows Operating System[1].

2.1 Installing the Eclipse.org CDT

Requirements

- JRE 1.4 or greater

The installation of the Eclipse.org CDT IDE is rather straight-forward. To begin, head over to `www.eclipse.org/downloads` and select the **Eclipse SDK 3.2** option. From the list, select whatever ftp/http site works quickest for you, but make sure that the file you are downloading is indeed the Eclipse SDK 3.2 and not some other, older version! For users of an OS other than Windows, you will have to select the **Eclipse SDK 3.2** under *other downloads*. Once you have downloaded the **Eclipse SDK 3.2** unzip (or untar) it somewhere on your computer, preferably in line with where you generally keep your applications. Once it has been extracted start up the Eclipse.org IDE.

At this point, it will be necessary to install the appropriate tool-kit in order to use the Eclipse.org IDE for C/C++ development; however, this can easily be done through the Eclipse.org IDE with relatively little effort. From within the Eclipse.org IDE go to `Help->Software Updates->Find and Install`. From here, a dialog should appear. At this dialog select `Search for new features to install`. Once there, select `New Remote Site`. Give it some distinguishable name and for the url put `http://download.eclipse.org/tools/cdt/releases/callisto`. Click next and select the check box to select all CDT components. click next, read the license and if you agree to it, continue with the installation; however, should you disagree with the license then perhaps the Eclipse.org CDT IDE is not right for you. Once it finishes installing the CDT you will be prompted to restart the workbench. After the workbench restarts you will almost be ready to take advantage of the Eclipse.org CDT IDE [2].

2.2 Installing the Cygwin Environment

Unlike Linux and Mac OSX, Window's users have no direct access to a C/C++ compiler nor linker such as make, and so, must also install the Cygwin Environment before being able to make full use of the Eclipse.org CDT. To begin, get the installer from `http://www.cygwin.com` and launch it once it is finished downloading. Select a place to install cygwin at. It is recommended that you use the default path, but whatever your choice, make sure to remember where it is installed. All other choices provided in the installation are optional, but it is recommended that you use UNIX style line endings. Choose a mirror and once it has downloaded the setup.ini select

[1] That is not necessarily to say that the author condones the use of MS Windows OS, but for convenience has chosen to provide instructions for this particular OS as it is the least intuitive to install and make fully functional the capabilities of the Eclipse.org CDT

[2] Linux and Mac OSX users will not have to follow the rest of the installation. In fact, if you are a Mac OSX user or Linux user you have successfully installed the Eclipse.org IDE with CDT. Feel free to move on to the next section.

the arrow next to the **all** option and cycle through the available options until it says *install* next to *all*. At this point, go grab a cup of coffee or go do something constructive as the installation takes a very long time.

Once the installation is complete, there is one last thing to setting up the Cygwin Environment. Go to the `Control Panel->System->Advanced->Environmental Variables`. From here find the *PATH* variable and click on it to edit it. At the end of this variable's line add a semi-colon followed immediately by the path to your cygwin installation's *bin* folder. In a typical installation of cygwin, using the default path, at the end of your *PATH*[3] variable you should add:

```
;C:\cygwin\bin\
```

Hit ok and then apply. Now, open up a terminal to the command prompt by going to *Run* and typing *cmd*. At the command prompt type *gcc* and hit enter. If cygwin was appropriately added to your *PATH* variable then you should receive at the prompt *no input file*. At this point cygwin has successfully been installed.

3 Using the Eclipse.org CDT IDE

In this section some of the core capabilities of the Eclipse.org CDT will be demonstrated in a platform specific nature. Because of the vast number of capabilities of the Eclipse.org CDT, the sections have been broken down into the different utilities provided. If there are platform specific instructions the section will be further broken down into subsections corresponding to the platforms for which there are uniquely different instructions.

[3]This method has been known to cause some problems with other linux-based programs that have been ported to the Window's environment. Such programs known to cause problems are GAIM, a gtk+ based AOL instant messaging client, and GIMP, the GNU Image Manipulation Program

3.1 Creating Basic Executable Files

One of the most common goals of programming is to create some form of executable file that will, in some form or another, execute and produce the desired results. This first section on the usage of the Eclipse.org CDT is geared towards using a Managed Make C++ project and, regardless of your preference for an OS, should basically follow the same procedure.

To begin, open up the Eclipse.org CDT. Once it has loaded up, choose your workspace. Workspaces are the Eclipse.org IDE's method of handling projects. Typically, it is best to associate one folder(or directory) as a workspace for each type of programming environment you will be working with. So, for example, if you develop some programs that are Java based and some programs that are C++ based, then you should have two seperate workspaces, and thus two seperate folders (or directories) for these environment. Once you have selected your workspace you should be greeted by the Eclipse.org IDE welcome screen. Since we will be working with the CDT, from the menubar select `Window->Open Perspective->C/C++`. In the Eclipse.org IDE perspective are different layouts for the IDE's workbench. These perspectives change the way menus are displayed so that functions relevant to the perspective are displayed. For instance, under the Project menu heading you should now see an option for building and cleaning. For the time being, de-select the *Build Automatically* option as it will prove more harmful than helpful in this tutorial.

Now that we have the appropriate perspective open, go to `File->New->Project->Managed C/C++ Project` and in the dialog give the project a unique name. Hit next and, since we're making an executable file, the default options should suffice; however, at this point in the project creation wizard, you can choose what type of program you will be working on. There are several options including, `Exectuable (GNU on <OS>), Static Library (GNU on <OS>), and Dynamic Library (GNU on <OS>)`. Each of these options will be covered and explained in detail, but for now, just leave it on

`Executable (GNU on OS)`. Hit next, and for the `C++ Indexer` select the `Full Indexing` option. This option allows for auto-completion of code, akin to what is found in the standard Eclipse.org IDE as well as other popular IDEs such as Dev-CPP[4] and Code::Blocks.

Now, you should notice that the project appears in the `C++ Projects Window`, which is located just to the left of the `Editor`. Good programming practices would suggest that creating seperate folders (directories) for your headers and source files, so that seems the best place to start. Right-click on the project in the `C++ Projects Window`, or alternatively go to `File->New`, and select **Source Folder**. A **Source** folder is a folder that the Eclipse.org CDT uses as an alternate path to check for the code in which the **main()** function is located. Call this folder **Source** or **src**. Again, right-click on the `C++ Projects Window`, or, as per the alternate method of the **Source Folder**, go to `File->New`, and select **Folder**. Call this folder **Headers** or **Include**.

Imperative to being able to program in C/C++ are the files that make up your source code. Right-click on the **Source** (**src**) folder and select `New->Source File`. Give this source file some clever name that is reflective of either its functionality or classname. Because of the way the Eclipse.org CDT is designed, you need to include the file extention in the name. So, be sure to add a .cpp to your source file's name. Next, right-click on the **Headers** (**Include**) folder and select `New->Header File`. Again, give this header file a name reflective of either its functionality or classname. In this case, give it the same name as the **Source** file you just created, substituting a .h in place of a .cpp. These header files are where you should define all your function prototypes as well as any include files necessary for your prototypes to be defined[5].

[4]Incidentally, this IDE is no longer in development; however, Code::Blocks IDE offers full support of the popular Dev-Pak options that made this free IDE so popular

[5]Typically, any header files necessary for the actual implementation of the function prototypes should be added into the source file where these prototypes are defined. This helps to avoid including functions more than once.

3.2 CSC406 Jumpstart

At this point, I am going to diverge a bit from the format of this tutorial to provide you with everything that is essential for setting up your assignments in this class. This tutorial is far from complete; however, contained within this section you will be able to setup the framework provided by Dr. Herv, compile, and run this framework.

So, to begin, head over to the course web-site and download the framework. Next, open up the Eclipse.org CDT and start up a new project. Call this whatever you would like, but bear in mind these files will be the basis for many of your assignments and lab experiments, so choose a name to appropriately reflect this. Now, create a **Source Folder** and a **Folder** for the header files. Once those are created, drag the **simpleGLUT.cpp** file into the **Source Folder** and the **glPlatform.h** file into the **Folder** for the header files. Well formed projects will keep **header** files and **source** files seperate from one another, which is why I stress the creation of two folders rather than just one flat project.

Next, we need to let the Eclipse.org CDT know which paths to search in order to find libraries and header files. Right-click on the project in the **C++ Project Window** and go to **Properties**. At this point you should be greeted with a dialog that says 'Properties for ProjectName'. Select `C/C++ Build` and the dialog should then adjust to show you configuration details for the **debug** configuration.

3.2.1 Windows with Cygwin

For Windows users, click on the **GCC C++ Compiler** and select **Directories**. Click the **Add** button under the `Include Path -I` section and add the following to the dialog that appears:

```
C:\cygwin\usr\include\w32api
```

If you installed cygwin elsewhere, just replace **cygwin** in the above line with your path to the cygwin installation. Now, click the **Add** button again

and select `Workspace...`. Navigate to where your header folder is and click okay. Hit okay again and you should have a path similar to:

```
"${workspace_loc:/YourProject/include}"
```

This should take care of any header files you might need. Next, we need to add the appropriate libraries to our project. These libraries contain the actual implementations of GL, GLU, and GLUT. To add these libraries, go to the **GCC C++ Linker** and select **Libraries**. Under the **Libraries -l** section click add and type `opengl32`, and then hit okay. You will want to repeat this step for `glu32` and `glut32`. Afterwards, you should have three libraries under the **Libraries -l** header: opengl32, glu32, and glut32. We must handle one last thing before we can compile. The library search path must be defined. To do this, under the **Library search path -L** click add

```
C:\cygwin\lib\w32api
```

Click **apply** and hit **ok**. Go up to project select **Build project**. Let it build and then select `Run->Run...`. A new dialog should appear. Select your project under the **C/C++ Local Applications**. From here, click **Search Project**. It should find a .exe file. Hit **OK** and then hit **Run**.

That is it! You are good to go.

3.2.2 Mac OS X

For Mac OS X users, click on the **MacOS C++ Compiler** and select **Directories**. Click the **Add** button again and select `Workspace...`. Navigate to where your header folder is and click okay. Hit okay again and you should have a path similar to:

```
"${workspace_loc:/YourProject/include}"
```

This should take care of any header files you might need. Next, we need to add the appropriate libraries to our project. These libraries contain the actual implementations of GL, GLU, and GLUT. To add these libraries, go to the **MacOS X C++ Linker** and select **Libraries**. Under the **Libraries -l** section click add and type `GL`, and then hit okay. You will want to repeat this step for `GLU`. Afterwards, you should have two libraries under the **Libraries -l** header: GL and GLU. We also need the GLUT framework, but to do this is a bit different from the other two libraries. Under the **MacOS X C++ Linker** select **Miscellaneous**, select **Add** under **Other Objects** and include the following:

```
/System/Library/Frameworks/GLUT.framework/
Versions/Current/GLUT
```

We must handle one last thing before we can compile. The library search path must be defined. To do this, under the `MacOS X Library->Library search path -L` click add and type the following into the field:

```
/System/Library/Frameworks/OpenGL.framework/
Versions/Current/Libraries
```

Click **apply** and hit **ok**. Go up to project and select **Build project**. Let it build and then select `Run->Run...`. A new dialog should appear. Select your project under the **C/C++ Local Applications**. From here, click **Search Project**. It should find a .exe file. Hit **OK** and then hit **Run**.

That is it! You are good to go.