Subject: : AmigaOS4
Topic: : Using Visual Studio 2017 to Cross-Compile PowerPC Amiga OS4 Code via Cygwin-based adtools Toolchain
Using Visual Studio 2017 to Cross-Compile PowerPC Amiga OS4 Code via Cygwin-based adtools Toolchain
Author: : stonecracker
Date: : 2017/10/6 7:26:08
URL:

Using Microsoft Visual Studio (VS) Community 2017 as an IDE to Develop/Cross-Compile PowerPC (ppc) Amiga OS4.x (OS4) Binaries Using a Cygwin-GCC/G++-based Amiga Toolchain (cyg-adtools) on Windows 10

[First Published 2017-10-06. Updated to explicitly show where tab spacing should be in make files, plus a note to avoid spaces in file/directory names, and a note about the symlinks used. Updated to show use of "-gstabs" option in makefile instead of "-gdbg", and also a link to @kas1e's nice tutorial on using gdb on OS4. Updated to describe adding other Include and Library search paths in makefile for 3rd party or other headers/libs.]

[Updated 2019-10-26 -- Note, there are 3 related posts here that cover cross-compiling for Amiga OS4 from Windows:

1) A post on how to build a Windows-hosted toolchain (http://www.amigans.net/modules/xforum ... hp?topic_id=7623&forum=25)

2) A post on how to use the toolchain built by the first post, from a Windows / cygwin command line (http://www.amigans.net/modules/xforum ... m=25&topic_id=7654&order=)

3) The post you're reading now, on how to use the toolchain built in the first post from inside Visual Studio for Windows (http://www.amigans.net/modules/xforum ... 9&forum=25&post_id=108233)]

Prelude:

This guide is long. Very long. It's not long because it's difficult or even time consuming to accomplish its goal, which is to get Visual Studio to use your nice new Cygwin-hosted adtools. It isn't. It's actually really fast/easy -- once you know what you're doing. I actually can get a new cross-compiling Amiga OS4 source project set-up/going in Visual Studio inside of 30 seconds.

The "know what you're doing" is the lengthy part, and that's what I'm hoping to impart onto you, to help save you the insane time it took me to catch every tiny little nuance of making this work, because some of it was absolutely counter-intuitive and unexpected.

To accomplish this setup for all your future projects, you need to gain some knowledge on the workings of VS

and how it interacts with Cygwin and the Cygwin-hosted adtools toolchain.

So, I've attempted to craft a truly complete and entirely self-contained guide. That means it's rather verbose. But hopefully you only need this guide (plus the pre-requisite one) to get everything going. Hopefully, you won't need to read 100's of posts and chains of conflicting/out-of-date information.

Why bother?

Because VS is slick, and makes AmigaOS/OS4 coding (any coding, really) that much more enjoyable and easy. Unless/until a great native/Amiga IDE is released, cross-compiling is not only a good idea -- it's mission critical. I say this because creating a great/comfortable cross-compiling toolset a pre-requisite in my mind to keeping/attracting developers for OS4 because the more barriers we have (i.e., forcing the legions of Windows/Mac/Linux-based developers to abandon their favourite non-Amiga dev environments, just to develop for the beloved AmigaOS), the less developers we have.

I, myself, was very much in the same boat -- just trying to get either a native or cross-compiling toolchain going was a painful ordeal. I (like all other devs who've stuck around or are just now coming back to the OS) had to make a choice -- burn a bunch of hours to get this going smoothly, or stop trying and abandon a 25-year reunion with my all-time-favourite OS, or ... burn an _insane_ amount of time just getting cross-compiling toolchains and IDE's to work and write about it so others wouldn't have to.

VS and the cyg-adtools package can offer an Amiga OS4 C/C++ developer a really powerful, modern, robust, heavily developed, and as-up-to-date-as-you'd like, cross-compiling Integrated Development Environment on Windows 10. There are countless plugins constantly being developed for VS, and since VS Community 2017 (and probably beyond) are now given away by Microsoft at zero cost, VS has been exploding in popularity. A wise move, in my opinion, to help Microsoft stem their losses in the massive global development community who may otherwise have little (or zero) need to develop on, or for, Windows.

VS definitely is _not_ the lightest-weight / smallest-footprint / fastest-running / easy-to-use IDE in the world. It _is_ one of the most powerful and popular ones -- and not just for C/C++ developers anymore, and also not just for Windows developers anymore.

Really the only thing that modern developers would like that is missing from this guide is the ability to have VS and its tools act as a tightly-integrated front-end for remote debugging (ex: VS running VisualGDB or WinGDB or remote gdb on Windows, with a remotely-connected instance of gdb running in your Amiga environment). The missing piece to get remote debugging working is how to make a virtual serial or SSH connection from the Amiga environment to a Windows one. I think it's possible. I just haven't worked on the issue. If I do figure it out, I'll post another guide on that subject as well.

This guide was developed using VS Community Edition 2017. I assume that all commercial/paid-for editions of VS would work as well. Additionally, this entire setup will also probably work just fine on Visual Studio Code 2017, maybe verbatim, if using Visual Studio Code on Windows.

But, that aside, I think it's really interesting that this entire setup (or at least the concepts) could work on Visual Studio for Mac and Linux. If so, then a new guide is very much warranted as the instructions would definitely have to change given that Mac and Linux don't use Cygwin at all (they don't need to ...the gcc tools would be native), so the set-up would be somewhat different. That is, you _wouldn't_ be using VS Code 2017 on Mac/Linux with the cyg-adtools package, you'd instead be generating and using an entirely different adtools package for Mac or Linux gcc/g++, and using makefiles from Visual Studio Code to access those cross-compiler binaries. At any rate, hopefully this guide can help with that if anyone tackles that effort.

Lastly, these instructions were supposed to be part two of two: part one being how to integrate the cyg-adtools package with the Eclipse IDE on Windows. However, my attempts to make that work were so awkward and difficult that I realized it would just be faster/easier to use a good developer's text editor and a Cygwin command shell. In other words, because of my horribly bad run-ins with Eclipse's setup environment refusing to use the adtools cross toolchain in Cygwin, and being really finicky with makefiles, I found that having no IDE whatsoever was far superior to trying to get Eclipse and cyg-adtools to work. It's almost certainly _not_ Eclipse's fault per se, I actually think it's a good IDE, I just couldn't figure it out.

The setup described here, by contrast, works really well, and borrows heavily from an older article on this subject, found at:

http://www.amigacoding.de/index.php?topic=18.0

If you _don't_ want to use an IDE at all, and instead just your favourite Windows editor, your Cygwin/UWin bash shell, and the cyg/uwin-adtools, then ignore this guide entirely and instead check-out:

http://www.amigans.net/modules/xforum ... m=25&topic_id=7654&order=

Pre-Requisites:

Before using _this_ guide, you should first read/complete the _other_ guide I wrote on building the Cygwin-based "adtools" package directly from the adtools source repository on github.

All of these instructions here fundamentally assume you've followed my other guide. Check out:

http://www.amigans.net/modules/xforum ... hp?topic_id=7623&forum=25

The impetus for this guide is that I discovered the exact same problem with this subject matter as in my prior one, namely there is nothing but outdated information about using Windows Integrated Development Environments (IDE)'s with any Cygwin-based AmigaOS/OS4 adtools cross compiler toolchains.

In this guide, I instruct on how to pick up where my other guide left off ... that is, I discuss how to bring my other guide's resulting Cygwin-hosted adtools cross-compiling chain into VS in an almost-seamless/almost-complete manner. Literally the only thing missing that I could see with this set-up is a functioning remote debugger to remotely run gdb on your Amiga OS4 test environment from your VS workstation.

I cover only the Cygwin toolchain here. Although my prior guide showed how to build the adtools toolchain on UWin (Bash on Ubuntu on Windows/Linux Subsystem (LXSS)/Windows Subsystem for Linux (WSL)), this guide doesn't show how to actually use that particular toolchain from inside Visual Studio. That's the subject of another guide I may yet produce (or not -- folks, feel free to take up that task).

Knowledge about GNU make/gmake/make files is helpful. This guide does, afterall, try to continue the paradigm of porting as much of the standard GCC/G++/gmake/make-based Linux C/C++ development methods/processes into the Windows environment as we can. Central to that is the ability to create some simple "make" files.

If you choose to use Visual Studio to create/edit Amiga source packages, then it's important to realize that your source package might be used by other developers (or yourself) in some development environment that is _not_ VS. So, to avoid the _guaranteed_ collisions/problems that would otherwise happen if we were not careful, these instructions specifically do NOT use the standard "makefile" naming conventions. Instead, we use the filenames "makefile.vs" and "makefile-dbg.vs" so that any developer who's expecting their standard command-line toolchain calls to work with non-VS/standard makefiles named "makefile" _won't_ get tripped up by you using VS in your development efforts.

By extension, this means if you intend to distribute your Visual Studio-developed source package to anyone, because other developers may not be using VS, you'll probably need to create both a standard make file named "makefile" AND your VS-specific "makefile.vs" and "makefile-dbg.vs" make files described herein.

Indeed, this is THE central idea of this guide: it's the VS-specific makefiles, (and some VS settings) that do the heavy lifting to force VS to actually use your cyg-adtools toolchain to create AmigaOS binaries.

And remember, the price to pay for using Visual Studio is that these makefiles and settings _must_ be created and configured for _every_ Amiga OS4.x cross-compiled project that you want to manipulate inside VS. It's a small price to pay, in my mind.

VS + cyg-adtools is, truly, slick and powerful. I don't give Microsoft much credit very often. They deserve kudos for making Visual Studio Community 2017 both excellent and free.

How-To:

Phew. Enough intro. Let's get to it.

TIP: In everything below, avoid using spaces in any directory name or file name. Save yourself some sanity because the resulting errors can be infuriating.

1) If you've not already done so, follow all the steps needed to create your Cygwin gcc/g++ setup and the Cygwin-hosted adtools toolchain from source. Go to the top of this guide and follow the link there if needed.

2) Install VS Community 2017. Once installed, run it, and add the VS "Linux Development with C++" Toolset using the Tools->Get Tools and Extensions menu option. Close VS.

3) Add the Cygwin "bin" directory to the Windows path. To do this, access the Windows 10 path settings by clicking on:

Windows Start Menu Button->Settings (the cog/wheel icon).

In the "Find a Setting" search box, type "view advanced" and the "View Advanced System Settings" auto-complete option should show. Click on it.

Then, the System Properties dialog box will display.

Click the "Environment Variables..." button.

In the System Variables sub dialog area, you should be able to scroll in that box and find the environment variable named "Path". Click on "Path" to highlight it, then click on the Edit button.

Add the "C:\cygwin64\bin" directory to the bottom of that pop-up Path Edit dialog box.

Click OK on all the dialog boxes you just launched.

You definitely want to reboot your machine to ensure this system path is now being used.

4) After reboot and login, start Visual Studio again.

5) Create a new project (File->New Project).

In the New Project dialog box, in the left-hand side Project Type selection tree (not labeled as such) select:

Installed -> Visual C++ -> General

In the middle pane, select Makefile Project.

Enter a name/directory of your project. For this guide, create a Project named "CygVsMakeProject1", and save it to VS's default $HOME\source\repos directory.

Note that VS 2017 contains "Projects" inside collections called "Solutions". A Solution can have many Projects. By default VS will create a Solution named the same as the first Project you create. That is, in VS's Solution Explorer window, you'll have a "CygVsMakeProject1" Solution that contains a single project also named "CygVsMakeProject1". Don't confuse the two in the following instructions.

6) You'll next be presented a large dialog box asking about your Debug Configuration Settings and about your Release Settings. These are 2 profiles you can use to effect 2 different types of builds, one with debug symbols inserted into the binaries and one without.

We'll definitely want to take advantage of VS's nice ability to do that, which means we'll use 2 different, but very similar, makefiles (see below) for this purpose.

In the Debug Settings dialog box:

The makefile referenced in these settings ("makefile-dbg.vs") _includes_ the compiler options for adding debug symbols to the object files/executables.

For both the "Build command line" and "Rebuild command line" fields:

```
make -f makefile-dbg.vs 2>&1 | sed -e 's/\(\w\+\):\([0-9]\+\):/\1(\2):/'
```

For the "Clean command Line" field:

```
make -f makefile-dbg.vs clean
```

For the "Output (for debugging) field:"

$(ProjectName)-dbg

Leave the other fields blank for now. Click Next.

Release Settings dialog box:

This is essentially the same settings as the Debug settings, but we need to reference a make file that _excludes_ any debug symbols from being inserted into binaries. This makefile is called "makefile.vs"

Uncheck the "Same as debug configuration" selection box.

For both the "Build command line" and "Rebuild command line" fields:

make -f makefile.vs 2>&1 | sed -e 's/\(\w\+\):\([0-9]\+\):/\1(\2):/'

For the "Clean command Line" field:

make -f makefile.vs clean

For the "Output (for debugging) field:"

$(ProjectName)

Leave the other fields blank for now. Click Finish.

7) Time to create the 2 makefiles referenced above. To understand the files you're about to create, know that in this guide, we'll be creating a small program called "easy" whose source filename is "easy.cpp". "easy" is a trivial, but really good, Amiga-specific example to prove that we have everything needed to create a true Amiga program, not just some standard portable C/C++ binary that doesn't use any AmigaOS-specific functions. The program itself is super-simple, when run, it just flashes the Workbench screen.

And now the interesting knowledge bits....

Very interestingly, our Cygwin-based makefile/make environment is a mix of both Windows/Cygwin execution spaces and command formatting. For example, the makefiles are written in pure POSIX style, complete with forward slashes for directory names, and respecting such things as Cygwin symbolic links/etc (but _not_ Windows backslashes and _not_ Windows junctions/symlinks).

Yet we still have Visual Studio running in a standard Windows execution space and it either directly or indirectly via the makefiles, calls such things as "make" and "ppc-amigaos-g++" and "rm" commands from the Windows environment -- not a Cygwin bash shell. "make" and "rm" aren't at all Windows commands, yet this works. It's why we needed to place the C:\cygwin64\bin directory into the Windows System Path variable so Cygwin could

do its magic. And, despite all this mix of Windows/POSIX commands/filesystem namespaces, Windows execution spaces, etc. it all flows effortlessly. Cygwin makes all that possible/nearly automatic.

But you, good Amigan, need to keep some things straight in your head.

For Visual Studio anything -- in all VS dialog boxes, settings fields, non-makefile scripts, plugins, configuration settings, etc., use only standard Windows conventions/naming/filenaming/backslashes/"C:"/etc. (but never forward slashes or Cygwin-created symlinks)

Inside makefiles -- use only POSIX standard conventions as understood by Cygwin, like forward-slashes in directory names, and use of Cygwin/bash-created symbolic links, but never back slashes or "C:"

This is important and is easy to confuse.

This means that although in VS, you'll reference a given source tree or file or directory using standard Windows syntax, those very same files and directories that you'll be hammering away at with your cross compiler toolchain inside the makefiles must be referenced differently.

Now, back to the instructions...

In VS, with the newly-created project open and selected (remember, you must have the Project selected, not the Solution, when doing this), do the following:

Project -> Add New item -> Utility -> Text File

In the resulting Text File dialog box:

Change the default filename shown from "Text.txt" to "makefile.vs". Make sure you _don't_ have "makefile.vs.txt" as the filename. You can save that file to its default location.

Repeat these steps immediately above to add another text file, and call it "makefile-dbg.vs". You can save that file to its default location.

In the left-most pane (VS's Solution Explorer window) you should now see 2 child objects of the CygVsMakeProject1 project you just created. Those 2 child objects should be the 2 makefile text files. They should be in the Project directory, not some subdirectory like the "source" directory. If they are in a subdirectory, move them back up to the Project directory.

8) Let's edit each of the makefiles now.

Note, I'm assuming you've followed my prior post on creating the adtools chain from source, which put the binaries in the location referenced below, and have also created symlinks that are relied-upon below.

Please note!!!! Those are TABS in the second/indented line beneath each of "all" and "clean". They're not spaces, you must use a tab indent. Don't forget those!

In the VS editor pane, you should have "makefile.vs" already open, copy/paste the following into that file (remembering that you should check for, or just manually put in, a tab in those indented lines):

all:

<put a tab here> /usr/local/amiga/adtools/bin/ppc-amigaos-g++ -o easy easy.cpp -Wall
-I/usr/local/amiga/adtools/ppc-amigaos/SDK/include

clean:
<put a tab here> rm easy

Similarly, in the "makefile-dbg.vs" editor window, copy/paste the following into that file (remembering that you should check for, or just manually put in, a tab in those indented lines):

all:
<put a tab here> /usr/local/amiga/adtools/bin/ppc-amigaos-g++ -gstabs -o easy-dbg easy.cpp -Wall
-I/usr/local/amiga/adtools/ppc-amigaos/SDK/include

clean:
<put a tab here> rm easy-dbg

Save both the "makefile.vs" and "makefile-dbg.vs" files.

Some words about both makefiles for your own knowledge/future use.

1st, note the use of POSIX syntax throughout. This includes forward slashes and a reliance on a symlink created earlier if you followed my pre-requisite guide.

2nd, note the absolute Cygwin-based path/filename to reference the specific Amiga g++ cross compiler executable.

3rd, note the -I option, which is how I can specify the includes search path so that the amigaos-g++ compiler knows where to find the Amiga and other headers I'd like to use. The way I've set this up, Visual Studio doesn't actually communicate much with the makefiles except to invoke them; that's why they have to be so involved/detailed in things that might otherwise be passed-in as parameters in any other IDE/makefile environment. IN ORDER FOR YOU CODE TO COMPILE/LINK, YOU ABSOLUTELY MUST ENSURE THIS MAKEFILE INFORMS THE COMPILER OF YOUR INCLUDES SEARCH PATH! Use a '-I' for every directory needed. That might mean reallllly long commands with lots of long '-I' statements but that's fine.

Similarly, though not shown in this example, just like the '-I' option, you need to add '-L' options to tell the linker what directories to search for libraries to link against. Has the same syntax as the '-I' option, and use another '-L' for every directory that needs to be searched.

4th, these are stupid-simple makefiles. You can make much, much, smarter ones than this. There are lots of ways to avoid lots of long '-I' and '-L' entries for example.

But I've shown this as-is because the information here is the _minimum_ you need to have these makefiles work with VS and your Cyg-adtools. You might be able to use passed-in parameters from Visual Studio instead of stupid hard-coding of things like the include path or even filenames of anything. You might also be able to use environment variables to avoid any hard-coding of filenames/paths in a makefile. You probably can use parameterized substitutions to allow for a super-smart 100% generally re-usable makefile that you just copy/paste into 100% of your Visual Studio Amiga projects without a moment's thought. Please feel free to do so and post your improvements in comments to this guide.

9) Now, let's to the simple thing, copy/paste-in some code into a file named easy.cpp.

Create easy.cpp by again selecting the Project named CygVsMakeProject1 in the Solution Exploerer window.

Then, choose Project -> Add New Item -> Visual C++ -> C++ File (cpp)

In the filename dialog, keeping the same default directory supplied by Visual Studio, change the file name to "easy.cpp" and click "Add"

In the Solution Explorer, you should then see a single node in the "Source Files" tree, which is the one file you just added called "easy.cpp"

10) Copy/paste the following into the "easy.cpp" file's edit window.

```
/* easy.cpp: a complete example of how to open an Amiga function library in C/C++.
* In this case the function library is Intuition. Once the Intuition
* function library is open and the interface obtains, any Intuition function
* can be called. This example uses the DisplayBeep() function of Intuition to
* flash the screen.
*/

#include <proto/exec.h>
#include <proto/intuition.h>

struct Library *IntuitionBase;
struct IntuitionIFace *IIntuition;

int main()
{
IntuitionBase = IExec->OpenLibrary("intuition.library", 50);

// Note it is safe to call GetInterface() with a NULL library pointer.
IIntuition = (struct IntuitionIFace *)IExec->GetInterface(IntuitionBase, "main", 1, NULL);

if (IIntuition != NULL)          /* Check to see if it actually opened.   */
{                              /* The Intuition library is now open so  */
IIntuition->DisplayBeep(0);  /* any of its functions may be used.     */
}

// Always drop the interface and close the library if not in use.
// Note it is safe to call DropInterface() and CloseLibrary() with NULL pointers.
IExec->DropInterface((struct Interface *)IIntuition);
IExec->CloseLibrary(IntuitionBase);
}
```

11) Save the easy.cpp file. Use the "Save all" button in the VS toolbar, just in case you missed saving a bunch of your work to this point.

You're (finally) ready to build/compile/test.

12) Let's test this set-up!

In the profile dropdown (in the toolbar, just underneath the "Debug" and "Team" menu items), you should see the options "Release" and "Debug".

Beside that drop-down is a platform selection dropdown. Ignore that one.

Now, choose "Debug" from the profile dropdown.

Then choose Build -> Build CygVsMakeProject1

Check out the Output window at the bottom right of the VS screen.

If all is good, in a second or two, you should see...

1>/usr/local/amiga/adtools/bin/ppc-amigaos-g++ -ggdb -o easy-dbg easy.cpp
-I/usr/local/amiga/adtools/ppc-amigaos/SDK/include
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========

Then, from the profile drop-down, choose "Release".

And again, choose Build -> Build CygVsMakeProject1

Which should show in the Output window...

"1>/usr/local/amiga/adtools/bin/ppc-amigaos-g++ -o easy easy.cpp
-I/usr/local/amiga/adtools/ppc-amigaos/SDK/include
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =========="

Which means you now have 2 AmigaOS OS4 PowerPC executables, one named "easy" and another named "easy-dbg" in the Project's directory (ex: C:\<path to user's Windows home dir>\repos\<SolutionName>\<ProjectName>)

13) Go test "easy" and "easy-dbg" by running those executable files in your AmigaOS environment from a command shell. If the Workbench screen flashes once each time you run those programs, you're good to go!

You now have a slick cross-compiling Visual Studio 2017 environment for you to develop away.

You can update the adtools compiler chain at any time, just by following the instructions in my prior post, and if you follow those directions closely (especially the symbolic linking I mention throughout), then the only changes you'll to make to keep Visual Studio "in the loop" is to update your Windows-side symlink (the "adtoolsln" link) described below to allow VS's IntelliSense to keep using the latest header files for its syntax/content lint'ing.

More on this below.

What About Existing Code?

All these examples above have created new files from scratch. But what about all those Amiga OS4 programs and existing code trees? With resources/libraries/etc?

This is rather beyond the scope of this guide. But as a pointer, you can't just copy source packages into some VS project directory and assume Visual Studio knows about the files. Look into Visual Studio's "Project -> Add Existing Item" option or "File -> Open Folder..." functionality.

What About Debugging?

As noted above, the only thing missing from this guide is a nicely-integrated, single-step, source-level, remote debugging interface.

While that's certainly true unless/until I (or someone reading this) solves the remote gdb connectivity issue between your OS4 test environment and your Visual Studio development PC, manual debugging using gdb on the OS4 environment works like a charm.

If you'd like to know how to do this, a nice shout out goes to @kas1e for his tutorial on using gdb in an Amiga environment, which you can find at:

http://www.os4coding.net/blog/kas1e/gdb-beginners

If you follow his tutorial, you'll find that the debug versions of the binaries built from this setup here will work beautifully. Suggest that if you do use gdb, that you also copy to your Amiga OS4 test environment, all the source files used to build your OS4 binaries, into the same directory where your binaries are copied -- that way gdb can easily reference your source as needed.

And, Finally, Getting IntelliSense to Work

One of the most valuable features of Visual Studio is its IntelliSense code completion/monitoring/profiling abilities....But it's useless if it doesn't know the code/headers it's supposed to be using.

If you look at the "easy.cpp" code file in its editor in Visual Studio, you'll see lots of red underlines and warning symbols and errors. That's because we don't currently have the CygVsMakeProject1 project set-up correctly to tell IntelliSense where to find all the AmigaOS headers (well, actually, any headers used by the .cpp file in question, not just AmigaOS SDK ones).

Let's fix that.

For your sanity, in anticipation of future updates to your adtools chain, create directory symlink in Windows by launching an Administrator Command Prompt. (Find your Command Prompt item in your Windows start menu, right-click on it, then right-click again on the "Command Prompt" sub menu item, and choose "Run as Administrator").

Then:

cd c:\cygwin64\usr\local\amiga

And create a Windows/NTFS synbolic link (which you'll definitely need to redo every time you build a new version of the adtools chain so you're always pointing to the latest version of the headers).

mklink /D adtoolsln adtools-ppc-cyg64-20170623-404

[Note, use "adtoolsln" and _not_ "adtools" as your link name, because if you've followed all my instructions to this point, you already have an "adtools" symlink/junction file -- one that was created and used in the Cygwin setup to begin with].

Close that Command Prompt window.

Return to Visual Studio

Then with the CygVsMakeProject1 Project selected in the Solution Explorer:

Select Project -> Properties -> NMake

In the resulting dialog box, in the "Configuration" drop-down chooser, select "All Configurations"

Then, still in that dialog select NMake -> IntelliSense -> Include Search Path

And enter the following into the IntelliSense's Include Search Path value field:

C:\cygwin64\usr\local\amiga\adtoolsln\ppc-amigaos\SDK\include\include_h;C:\cygwin64\usr\local\amiga\adtoolsln\ppc-amigaos\SDK\clib2\include;C:\cygwin64\usr\local\amiga\adtoolsln\ppc-amigaos\SDK\newlib\include

And you'll definitely want to add any other include directories you'd like to have IntelliSense know about. Basically, whatever include directories you want in your make files for a given project, you'll probably want to have here, using Windows path/filename syntax.

But remember, all of this is just for the editor's error/warning highlights. None of these settings affect the Include search path of the actual gcc/g++ compilers. Again, the makefile include search paths probably will mirror the IntelliSense Include search paths -- albeit with different POSIX syntax.

***** That's all folks *****

In the end, it's actually quite fast/easy to do everything here. You'll probably repeat all these steps like I do inside of 30 seconds when you need them.

I just figured you'd want all the backgrounder/underlying knowledge needed so you can adjust to suit your needs in the future.

Some credits:

In the original posting which I drew a bunch of this information from (see above for full link URL), the author EDanaII, when they posted their message back in 2012, which itself was a repost, had a note of thanks at the end of the message. I'm copying that verbatim here: "I'd like to thank SG2 for his initial help in getting this working, and Hans for helping me publish the article!"

And now, my turn. I very heavily edited and of course heavily added-to the original post from EDanaII. I don't know EDanaII, but thanks go out to him/her for their work -- and additional big thanks to the original poster which EDanaII reposted from. Let's hope this helps to keep the effort going.

Search keywords:

Cross
Cross-compile
Cross-compiler
Cross-compiling
compile
compiler
compiling
adtools
Win
Windows
Bash
Ubuntu
Cygwin
Amiga Development Tools
Developer
Intel
x86
x64
PPC
PowerPC
Power PC
OS4
Ami
AmigaOS