

How to Integrate CUDA with Visual C++

If you want to program CUDA, first you will have to install latest nVidia CUDA Driver for your Graphics Hardware which supports desired CUDA version. Then you will have to install the CUDA Toolkit which includes the CUDA Compiler, Include file, lib file and binary files to develop your CUDA application.

The bin folder under your CUDA installation location (most probably C:\CUDA), you can see nvcc.exe which helps you to compile the CUDA program. If you give, nvcc my_cuda_filename.cu, the compiler will compile the source file and creates the executables (a.exe).

But in Windows world, most of the developers are much satisfied with the IDE Visual Studio. So may have to leave the world of command line compilation and source editing in favor of improving our productivity. If we can integrate the CUDA development to Visual Studio IDE, that's pretty nice no? In one of my previous post, [I said about enabling syntax highlighting for CUDA files under Visual Studio](#).

Now let's check how we can support CUDA compilation under Visual Studio. CUDA compiler has a dependency with C++ compiler. It supports either Visual C++ 7.1 or 8.0. Currently it doesn't support Visual C++ 9 (VS 2008).

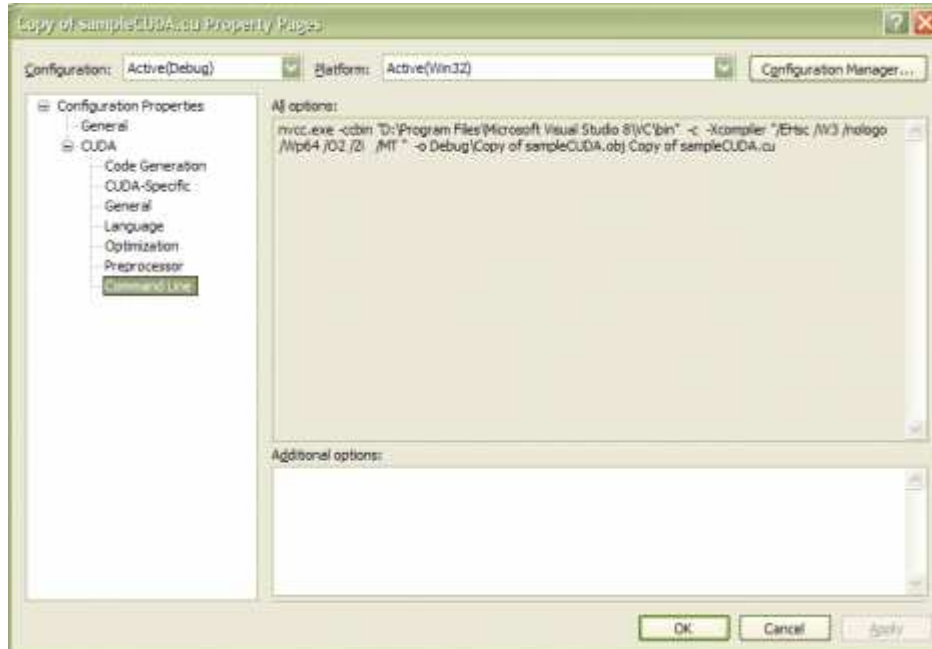
Method 1 – Install CUDA Build Rule for Visual Studio 2005

There's a painless method by installing a [custom build rule for Visual Studio 2005 developed by JaredHoferock](#). A cool installer is available to do the necessary settings. When you add the ".cu" files to your application, it shows a dialog box to select the rule you installed. Press "OK" button.



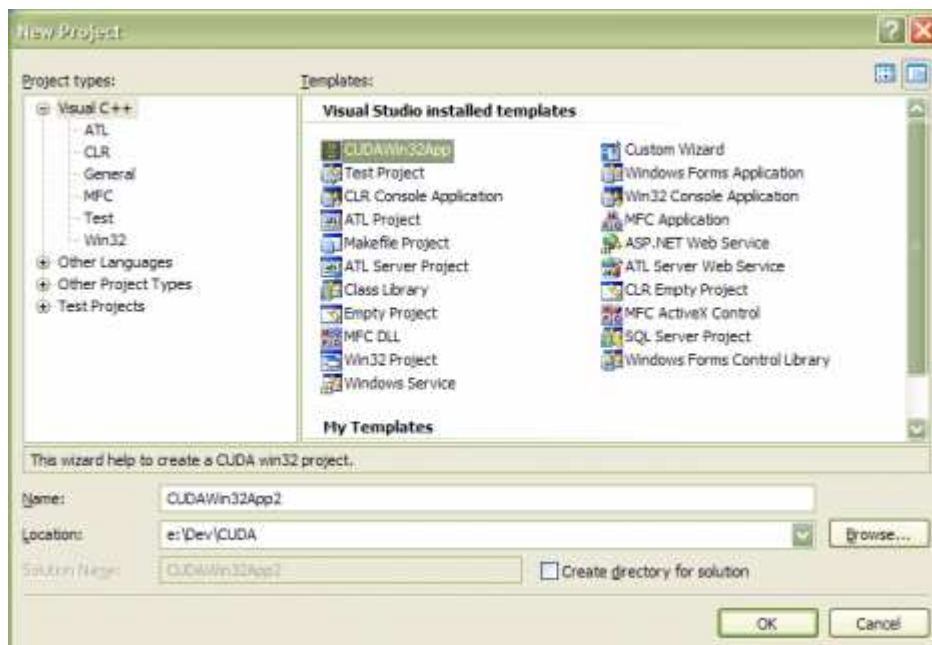
Select CUDA Build Rule

You can control the per-file (.cu file) configuration for your application by selecting the properties of the file from your solution explorer. See the figure below.



Detailed Build Configuration under Wizard

The installer also adds new Project Type to create your new CUDA project which does the necessary settings for your application.



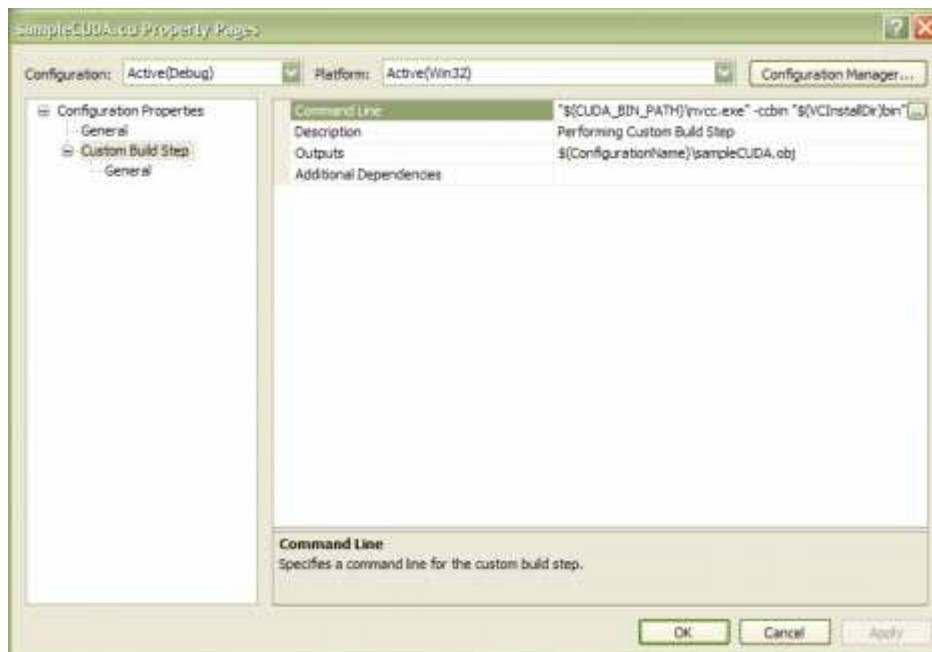
Wizard to create new CUDA Project

Finally you can put the required CUDA libraries (e.g CUDA Runtime library – cudart.lib) in Project->Properties->Linker->Input->Additional Dependency.

I think the installer won't work for Visual Studio 2003. I think you can manually copy the rule file from %Program Files%\Microsoft Visual Studio 8\VC\VCProjectDefaults\cuda.rules and specify this as the rule file for compilation. Anyway try it yourself.

Method 2 – Manually Configure by Custom Build Event

Without Visual Studio, we can use nvcc compiler for compiling the CUDA files with various command line parameters. So the same setup we can use with Visual Studio as well. One of the advantages is that, we can use various Visual Studio Specific variables to identify various file locations.



Custom Build

1. Select the CUDA source file from the solution explorer and take properties.
2. Select "Custom Build Setup" from the tree
3. You can select the active configuration from "Configuration" combo box.
4. Specify the following options under "Command Line"

```
"$(CUDA_BIN_PATH)\nvcc.exe" -ccbin "$(VCInstallDir)bin" -c -D_DEBUG -DWIN32 -D_CONSOLE -D_MBCS -Xcompiler /EHsc,/W3,/nologo,/Wp64,/Od,/Zi,/RTC1,/MTd -I"$(CUDA_INC_PATH)" -I./ -o $(ConfigurationName)\<your_cu_filename>.obj <your_cu_filename>.cu
```

The above configuration is to compile the source under debug configuration. For a release configuration, you will have to remove the symbol definition, `_DEBUG` (-D switch is to define symbols). Also the above command line ask to link with the static-debug version of C-Runtime library by specifying `/MTd` option. In the case of release build you will have to modify it to `/MD` or `/MT`. Please check the various [C-Runtime versions](#) from MSDN and apply accordingly. Thus a sample command line for release version may look as follows.

```
"$(CUDA_BIN_PATH)\nvcc.exe" -ccbin "$(VCInstallDir)bin" -c -DWIN32 -D_CONSOLE -D_MBCS -Xcompiler /EHsc,/W3,/nologo,/Wp64,/Od,/Zi,/RTC1,/MD -I"$(CUDA_INC_PATH)" -I./ -o $(ConfigurationName)\<your_cu_filename>.obj <your_cu_filename>.cu
```

If you need to add Device Emulation option, you add `-deviceemu` switch in the command line. Thus you the compiler will generate code for GPGPU Emulation library. You can understand more about each switches from MSDN itself. The information about the other variables used has been specified at the end of this post.

5. You can specify your own description for the custom build event under "Description" text box. (See the figure)
6. Finally you have to specify the output file name under "Outputs"

```
$(ConfigurationName)\<your_cu_filename>.obj
```

7. If you have included any CUDA files (probably will be having extension `.cu` or `.cuh`), you can specify it in the "Additional Dependencies option

In the command line string, we've used some visual studio specific build macros. [You can see the entire list of Visual Studio Build Macros in MSDN.](#)

8. Finally build your project and now see the output of CUDA compilation in your output Window.

Adding a new build configuration

Sometimes you have to build your file with CUDA emulation for many debugging purpose. [Please check MSDN to know more about adding build configuration.](#)

I'll explain how to call a CUDA Program from a C++ file in the next post. Thanks for your patience. I'm a newbie in CUDA. If you've better ideas and suggestions, please share with me through your comments.