NVIDIA CUDA Installation Guide for Microsoft Windows

Installation and Verification on Windows

DU-05349-001_v12.1 | February 2023

Table of Contents

Chapter 1. Introduction

CUDA® is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).

CUDA was developed with several design goals in mind:

- Provide a small set of extensions to standard programming languages, like C, that enable a straightforward implementation of parallel algorithms. With CUDA C/C++, programmers can focus on the task of parallelization of the algorithms rather than spending time on their implementation.
- Support heterogeneous computation where applications use both the CPU and GPU. Serial portions of applications are run on the CPU, and parallel portions are offloaded to the GPU. As such, CUDA can be incrementally applied to existing applications. The CPU and GPU are treated as separate devices that have their own memory spaces. This configuration also allows simultaneous computation on the CPU and GPU without contention for memory resources.

CUDA-capable GPUs have hundreds of cores that can collectively run thousands of computing threads. These cores have shared resources including a register file and a shared memory. The on-chip shared memory allows parallel tasks running on these cores to share data without sending it over the system memory bus.

This guide will show you how to install and check the correct operation of the CUDA development tools.

1.1. System Requirements

To use CUDA on your system, you will need the following installed:

- A CUDA-capable GPU
- A supported version of Microsoft Windows
- A supported version of Microsoft Visual Studio
- The NVIDIA CUDA Toolkit (available at http://developer.nvidia.com/cuda-downloads)

The next two tables list the currently supported Windows operating systems and compilers.

Table 1. Windows Operating System Support in CUDA 12.1

Operating System	Native x86_64	Cross (x86_32 on x86_64)
Windows 11	YES	NO
Windows 10	YES	NO
Windows Server 2022	YES	NO
Windows Server 2019	YES	NO
Windows Server 2016	YES	NO

Table 2. Windows Compiler Support in CUDA 12.1

Compiler*	IDE	Native x86_64	Cross (x86_32 on x86_64)
MSVC Version 193x	Visual Studio 2022 17.2	YES	YES
MSVC Version 192x	Visual Studio 2019 16.11	YES	YES
MSVC Version 191x	Visual Studio 2017 15.9 (RTW and all updates)	YES	YES

* Support for Visual Studio 2015 is deprecated in release 11.1.

x86_32 support is limited. See the x86 32-bit Support section for details.

For more information on MSVC versions, Visual Studio product versions, visit https://dev.to/yumetodo/list-of-mscver-and-mscfullver-8nd.

1.2. x86 32-bit Support

Native development using the CUDA Toolkit on x86_32 is unsupported. Support for running x86 32-bit applications on x86_64 Windows is limited to supporting existing applications on GeForce GPUs. To this end only CUDA Driver will have 32-bit support. CUDA Runtime, Math Library, Compiler and Developer Tools will not support 32-bit.

1.3. About This Document

This document is intended for readers familiar with Microsoft Windows operating systems and the Microsoft Visual Studio environment. You do not need previous experience with CUDA or experience with parallel computation.

Chapter 2. Installing CUDA Development Tools

Basic instructions can be found in the Quick Start Guide. Read on for more detailed instructions.

The setup of CUDA development tools on a system running the appropriate version of Windows consists of a few simple steps:

- Verify the system has a CUDA-capable GPU.
- Download the NVIDIA CUDA Toolkit.
- Install the NVIDIA CUDA Toolkit.
- Test that the installed software runs correctly and communicates with the hardware.

2.1. Verify You Have a CUDA-Capable GPU

You can verify that you have a CUDA-capable GPU through the **Display Adapters** section in the **Windows Device Manager**. Here you will find the vendor name and model of your graphics card(s). If you have an NVIDIA card that is listed in http://developer.nvidia.com/cuda-gpus, that GPU is CUDA-capable. The Release Notes for the CUDA Toolkit also contain a list of supported products.

The **Windows Device Manager** can be opened via the following steps:

1. Open a run window from the Start Menu
2. Run:

```
control /name Microsoft.DeviceManager
```

2.2. Download the NVIDIA CUDA Toolkit

The NVIDIA CUDA Toolkit is available at https://developer.nvidia.com/cuda-downloads. Choose the platform you are using and one of the following installer formats:

1. Network Installer: A minimal installer which later downloads packages required for installation. Only the packages selected during the selection phase of the installer are downloaded. This installer is useful for users who want to minimize download time.
2. Full Installer: An installer which contains all the components of the CUDA Toolkit and does not require any further download. This installer is useful for systems which lack network access and for enterprise deployment.

The CUDA Toolkit installs the CUDA driver and tools needed to create, build and run a CUDA application as well as libraries, header files, and other resources.

Download Verification

The download can be verified by comparing the MD5 checksum posted at https://developer.download.nvidia.com/compute/cuda/11.6.2/docs/sidebar/md5sum.txt with that of the downloaded file. If either of the checksums differ, the downloaded file is corrupt and needs to be downloaded again.

To calculate the MD5 checksum of the downloaded file, follow the instructions at https://support.microsoft.com/kb/889768.

2.3. Install the CUDA Software

Before installing the toolkit, you should read the *Release Notes*, as they provide details on installation and software functionality.

Note: The driver and toolkit must be installed for CUDA to function. If you have not installed a stand-alone driver, install the driver from the NVIDIA CUDA Toolkit.

Note: The installation may fail if Windows Update starts after the installation has begun. Wait until Windows Update is complete and then try the installation again.

Graphical Installation

Install the CUDA Software by executing the CUDA installer and following the on-screen prompts.

Silent Installation

The installer can be executed in silent mode by executing the package with the `-s` flag. Additional parameters can be passed which will install specific subpackages instead of all packages. See the table below for a list of all the subpackage names.

Table 3. Possible Subpackage Names

Subpackage Name	Subpackage Description
Toolkit Subpackages (defaults to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1)	
cuda_profiler_api_12.1	Driver and Runtime APIs for ProfilerStart/Stop.
cudart_12.1	CUDA Runtime libraries.
cuobjdump_12.1	Extracts information from cubin files.
cupti_12.1	The CUDA Profiling Tools Interface for creating profiling and tracing tools that target CUDA applications.
cuxxfilt_12.1	The CUDA cu++ filt demangler tool.
demo_suite_12.1	Prebuilt demo applications using CUDA.
documentation_12.1	CUDA HTML and PDF documentation files including the CUDA C++ Programming Guide, CUDA C++ Best Practices Guide, CUDA library documentation, etc.
nvcc_12.1	CUDA compiler.
nvdisasm_12.1	Extracts information from standalone cubin files.
nvjitlink_12.1	nvJitLink library.
nvml_dev_12.1	NVML development libraries and headers.
nvprof_12.1	Tool for collecting and viewing CUDA application profiling data from the command-line.
nvprune_12.1	Prunes host object files and libraries to only contain device code for the specified targets.
nvrtc_12.1 nvrtc_dev_12.1	NVRTC runtime libraries.
nvtx_12.1	NVTX on Windows.
nvvm_samples_12.1	NVVM samples.
opencl_12.1	OpenCL library.
visual_profiler_12.1	Visual Profiler.
sanitizer_12.1	Compute Sanitizer API.
thrust_12.1	CUDA Thrust.
cublas_12.1 cublas_dev_12.1	cuBLAS runtime libraries.
cufft_12.1 cufft_dev_12.1	cuFFT runtime libraries.
curand_12.1	cuRAND runtime libraries.

Subpackage Name	Subpackage Description
curand_dev_12.1	
cusolver_12.1 cusolver_dev_12.1	cuSOLVER runtime libraries.
cusparse_12.1 cusparse_dev_12.1	cuSPARSE runtime libraries.
npp_12.1 npp_dev_12.1	NPP runtime libraries.
nvjpeg_12.1 nvjpeg_dev_12.1	nvJPEG libraries.
nsight_compute_12.1	Nsight Compute.
nsight_systems_12.1	Nsight Systems.
nsight_vse_12.1	Installs the Nsight Visual Studio Edition plugin in all VS.
visual_studio_integration_12.1	Installs CUDA project wizard and builds customization files in VS.
occupancy_calculator_12.1	Installs the `CUDA_Occupancy_Calculator.xls` tool.
Driver Subpackages	
Display.Driver	The NVIDIA Display Driver. Required to run CUDA applications.

For example, to install only the compiler and driver components:

```
<PackageName>.exe -s nvcc_12.1 Display.Driver
```

Use the `-n` option if you do not want to reboot automatically after install or uninstall, even if reboot is required.

Extracting and Inspecting the Files Manually

Sometimes it may be desirable to extract or inspect the installable files directly, such as in enterprise deployment, or to browse the files before installation. The full installation package can be extracted using a decompression tool which supports the LZMA compression method, such as 7-zip or WinZip.

Once extracted, the CUDA Toolkit files will be in the `CUDAToolkit` folder, and similarly for CUDA Visual Studio Integration. Within each directory is a .dll and .nvi file that can be ignored as they are not part of the installable files.

Note: Accessing the files in this manner does not set up any environment settings, such as variables or Visual Studio integration. This is intended for enterprise-level deployment.

2.3.1. Uninstalling the CUDA Software

All subpackages can be uninstalled through the Windows Control Panel by using the Programs and Features widget.

2.4. Using Conda to Install the CUDA Software

This section describes the installation and configuration of CUDA when using the Conda installer. The Conda packages are available at https://anaconda.org/nvidia.

2.4.1. Conda Overview

The Conda installation installs the CUDA Toolkit. The installation steps are listed below.

2.4.2. Installation

To perform a basic install of all CUDA Toolkit components using Conda, run the following command:

```
conda install cuda -c nvidia
```

2.4.3. Uninstallation

To uninstall the CUDA Toolkit using Conda, run the following command:

```
conda remove cuda
```

2.4.4. Installing Previous CUDA Releases

All Conda packages released under a specific CUDA version are labeled with that release version. To install a previous version, include that label in the `install` command such as:

```
conda install cuda -c nvidia/label/cuda-11.3.0
```

Note:

Some CUDA releases do not move to new versions of all installable components. When this is the case these components will be moved to the new label, and you may need to modify the install command to include both labels such as:

```
conda install cuda -c nvidia/label/cuda-11.3.0 -c nvidia/label/cuda-11.3.1
```

This example will install all packages released as part of CUDA 11.3.0.

2.5. Use a Suitable Driver Model

On Windows 10 and later, the operating system provides two *driver models* under which the NVIDIA Driver may operate:

- The *WDDM* driver model is used for display devices.
- The *Tesla Compute Cluster (TCC)* mode of the NVIDIA Driver is available for non-display devices such as NVIDIA Tesla GPUs and the GeForce GTX Titan GPUs; it uses the Windows *WDM* driver model.

TCC is enabled by default on most recent NVIDIA Tesla GPUs. To check which driver mode is in use and/or to switch driver modes, use the `nvidia-smi` tool that is included with the NVIDIA Driver installation (see `nvidia-smi -h` for details).

Note: Keep in mind that when TCC mode is enabled for a particular GPU, that GPU *cannot* be used as a display device.

Note: NVIDIA GeForce GPUs (excluding GeForce GTX Titan GPUs) do not support TCC mode.

2.6. Verify the Installation

Before continuing, it is important to verify that the CUDA toolkit can find and communicate correctly with the CUDA-capable hardware. To do this, you need to compile and run some of the included sample programs.

2.6.1. Running the Compiled Examples

The version of the CUDA Toolkit can be checked by running `nvcc -V` in a **Command Prompt** window. You can display a **Command Prompt** window by going to:

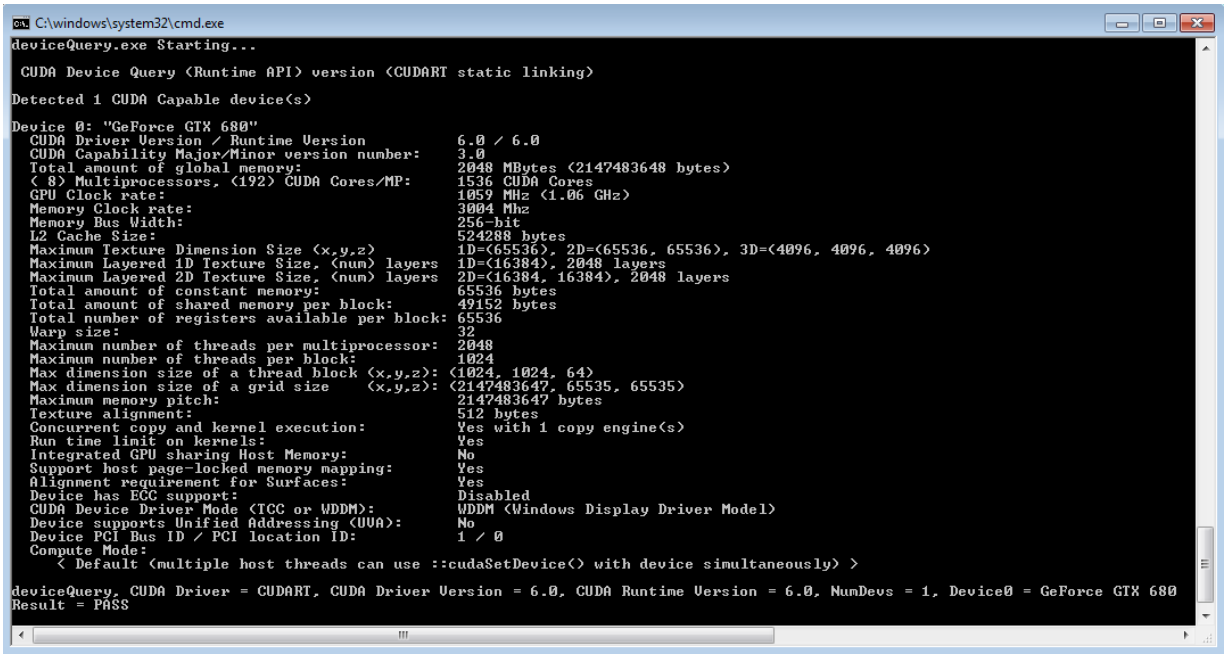
Start > All Programs > Accessories > Command Prompt

CUDA Samples are located in https://github.com/nvidia/cuda-samples. To use the samples, clone the project, build the samples, and run them using the instructions on the Github page.

To verify a correct configuration of the hardware and software, it is highly recommended that you build and run the `deviceQuery` sample program. The sample can be built using the provided VS solution files in the `deviceQuery` folder.

This assumes that you used the default installation directory structure. If CUDA is installed and configured correctly, the output should look similar to Figure 1.

Figure 1. Valid Results from deviceQuery CUDA Sample

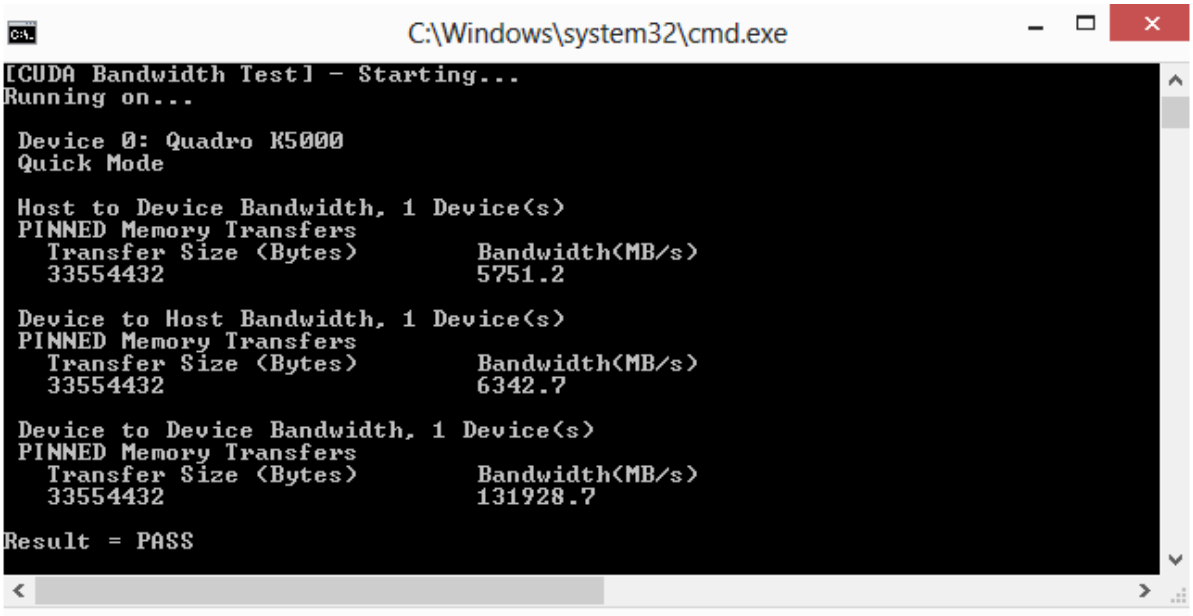


The exact appearance and the output lines might be different on your system. The important outcomes are that a device was found, that the device(s) match what is installed in your system, and that the test passed.

If a CUDA-capable device and the CUDA Driver are installed but `deviceQuery` reports that no CUDA-capable devices are present, ensure the deivce and driver are properly installed.

Running the `bandwidthTest` program, located in the same directory as `deviceQuery` above, ensures that the system and the CUDA-capable device are able to communicate correctly. The output should resemble Figure 2.

Figure 2. Valid Results from bandwidthTest CUDA Sample



The device name (second line) and the bandwidth numbers vary from system to system. The important items are the second line, which confirms a CUDA device was found, and the second-to-last line, which confirms that all necessary tests passed.

If the tests do not pass, make sure you do have a CUDA-capable NVIDIA GPU on your system and make sure it is properly installed.

To see a graphical representation of what CUDA can do, run the `particles` sample at

```
https://github.com/NVIDIA/cuda-samples/tree/master/Samples/particles
```

Chapter 3. Pip Wheels

NVIDIA provides Python Wheels for installing CUDA through pip, primarily for using CUDA with Python. These packages are intended for runtime use and do not currently include developer tools (these can be installed separately).

Please note that with this installation method, CUDA installation environment is managed via pip and additional care must be taken to set up your host environment to use CUDA outside the pip environment.

Prerequisites

To install Wheels, you must first install the `nvidia-pyindex` package, which is required in order to set up your pip installation to fetch additional Python modules from the NVIDIA NGC PyPI repo. If your pip and setuptools Python modules are not up-to-date, then use the following command to upgrade these Python modules. If these Python modules are out-of-date then the commands which follow later in this section may fail.

```
py -m pip install --upgrade setuptools pip wheel
```

You should now be able to install the `nvidia-pyindex` module.

```
py -m pip install nvidia-pyindex
```

If your project is using a `requirements.txt` file, then you can add the following line to your `requirements.txt` file as an alternative to installing the `nvidia-pyindex` package:

```
--extra-index-url https://pypi.ngc.nvidia.com
```

Procedure

Install the CUDA runtime package:

```
py -m pip install nvidia-cuda-runtime-cu12
```

Optionally, install additional packages as listed below using the following command:

```
py -m pip install nvidia-<library>
```

Metapackages

The following metapackages will install the latest version of the named component on Windows for the indicated CUDA version. "cu12" should be read as "cuda12".

- nvidia-cuda-runtime-cu12

- nvidia-cuda-cupti-cu12
- nvidia-cuda-nvcc-cu12
- nvidia-nvml-dev-cu12
- nvidia-cuda-nvrtc-cu12
- nvidia-nvtx-cu12
- nvidia-cuda-sanitizer-api-cu12
- nvidia-cublas-cu12
- nvidia-cufft-cu12
- nvidia-curand-cu12
- nvidia-cusolver-cu12
- nvidia-cusparse-cu12
- nvidia-npp-cu12
- nvidia-nvjitlink-cu12
- nvidia-nvjpeg-cu12
- nvidia-nvvm-samples-cu12
- nvidia-opencl-cu12

These metapackages install the following packages:

- nvidia-nvml-dev-cu120
- nvidia-cuda-nvcc-cu120
- nvidia-cuda-runtime-cu120
- nvidia-cuda-cupti-cu120
- nvidia-cublas-cu120
- nvidia-cuda-sanitizer-api-cu120
- nvidia-nvtx-cu120
- nvidia-cuda-nvrtc-cu120
- nvidia-npp-cu120
- nvidia-cusparse-cu120
- nvidia-cusolver-cu120

- nvidia-curand-cu120
- nvidia-cufft-cu120
- nvidia-nvjitlink-cu120
- nvidia-nvjpeg-cu120
- nvidia-nvvm-samples-cu120
- nvidia-opencl-cu120

Chapter 4. Compiling CUDA Programs

The project files in the CUDA Samples have been designed to provide simple, one-click builds of the programs that include all source code. To build the Windows projects (for release or debug mode), use the provided `*.sln` solution files for Microsoft Visual Studio 2015 (deprecated in CUDA 11.1), 2017, 2019, or 2022. You can use either the solution files located in each of the examples directories in

```
https://github.com/nvidia/cuda-samples
```

4.1. Compiling Sample Projects

The `bandwidthTest` project is a good sample project to build and run. It is located in https://github.com/NVIDIA/cuda-samples/tree/master/Samples/bandwidthTest.

If you elected to use the default installation location, the output is placed in `CUDA Samples\v12.1\bin\win64\Release`. Build the program using the appropriate solution file and run the executable. If all works correctly, the output should be similar to Figure 2.

4.2. Sample Projects

The sample projects come in two configurations: debug and release (where release contains no debugging information) and different Visual Studio projects.

A few of the example projects require some additional setup.

These sample projects also make use of the `$CUDA_PATH` environment variable to locate where the CUDA Toolkit and the associated `.props` files are.

The environment variable is set automatically using the Build Customization `CUDA 12.1.props` file, and is installed automatically as part of the CUDA Toolkit installation process.

Table 4. CUDA Visual Studio .props locations

Visual Studio	CUDA 12.1 .props file Install Directory
Visual Studio 2015 (deprecated)	C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\V140\BuildCustomizations

Visual Studio	CUDA 12.1 .props file Install Directory
Visual Studio 2017	<Visual Studio Install Dir>\Common7\IDE\VC\VCTargets\BuildCustomizations
Visual Studio 2019	C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional\MSBuild\Microsoft\VC\v160\BuildCustomizations
Visual Studio 2022	C:\Program Files\Microsoft Visual Studio\2022\Professional\MSBuild\Microsoft\VC\v170\BuildCustomizations

You can reference this `CUDA 12.1.props` file when building your own CUDA applications.

4.3. Build Customizations for New Projects

When creating a new CUDA application, the Visual Studio project file must be configured to include CUDA build customizations. To accomplish this, click File-> New | Project... NVIDIA-> CUDA->, then select a template for your CUDA Toolkit version. For example, selecting the "CUDA 12.1 Runtime" template will configure your project for use with the CUDA 12.1 Toolkit. The new project is technically a C++ project (.vcxproj) that is preconfigured to use NVIDIA's Build Customizations. All standard capabilities of Visual Studio C++ projects will be available.

To specify a custom CUDA Toolkit location, under **CUDA C/C++**, select **Common**, and set the **CUDA Toolkit Custom Dir** field as desired. Note that the selected toolkit must match the version of the Build Customizations.

Note: A supported version of MSVC must be installed to use this feature.

4.4. Build Customizations for Existing Projects

When adding CUDA acceleration to existing applications, the relevant Visual Studio project files must be updated to include CUDA build customizations. This can be done using one of the following two methods:

1. Open the Visual Studio project, right click on the project name, and select **Build Dependencies**->**Build Customizations...**, then select the CUDA Toolkit version you would like to target.
2. Alternatively, you can configure your project always to build with the most recently installed version of the CUDA Toolkit. First add a CUDA build customization to your project as above. Then, right click on the project name and select **Properties**. Under **CUDA C/C++**, select **Common**, and set the **CUDA Toolkit Custom Dir** field to `$(CUDA_PATH)` . Note that the `$(CUDA_PATH)` environment variable is set by the installer.

While Option 2 will allow your project to automatically use any new CUDA Toolkit version you may install in the future, selecting the toolkit version explicitly as in Option 1 is often

better in practice, because if there are new CUDA configuration options added to the build customization rules accompanying the newer toolkit, you would not see those new options using Option 2.

If you use the `$(CUDA_PATH)` environment variable to target a version of the CUDA Toolkit for building, and you perform an installation or uninstallation of any version of the CUDA Toolkit, you should validate that the `$(CUDA_PATH)` environment variable points to the correct installation directory of the CUDA Toolkit for your purposes. You can access the value of the `$(CUDA_PATH)` environment variable via the following steps:

1. Open a run window from the Start Menu
2. Run:

   ```
   control sysdm.cpl
   ```

3. Select the "Advanced" tab at the top of the window
4. Click "Environment Variables" at the bottom of the window

Files which contain CUDA code must be marked as a `CUDA C/C++` file. This can done when adding the file by right clicking the project you wish to add the file to, selecting **Add\New Item**, selecting **NVIDIA CUDA 12.1\Code\CUDA C/C++ File**, and then selecting the file you wish to add.

Note for advanced users: If you wish to try building your project against a newer CUDA Toolkit without making changes to any of your project files, go to the Visual Studio command prompt, change the current directory to the location of your project, and execute a command such as the following:

```
msbuild <projectname.extension> /t:Rebuild /p:CudaToolkitDir="drive:/path/to/new/
toolkit/"
```

Chapter 5. Additional Considerations

Now that you have CUDA-capable hardware and the NVIDIA CUDA Toolkit installed, you can examine and enjoy the numerous included programs. To begin using CUDA to accelerate the performance of your own applications, consult the *CUDA C++ Programming Guide*.

A number of helpful development tools are included in the CUDA Toolkit or are available for download from the NVIDIA Developer Zone to assist you as you develop your CUDA programs, such as NVIDIA® Nsight™ Visual Studio Edition, and NVIDIA Visual Profiler.

For technical support on programming questions, consult and participate in the developer forums at https://developer.nvidia.com/cuda/.

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
https://www.nvidia.com