

CUDA-LIB

(Version 1.1)

(Documentation and Installation guide)

Thejaswi. N. S

(snanditale@nvidia.com)

Contents:

1. Introduction
2. Installation
3. Deliverables
 - a. cudaInstall
 - b. cu.sh
 - c. libGen.sh
 - d. Project folder for CUDA library
 - e. Project folder for CUDA utility
4. Limitations
5. Contact

1. Introduction

In order to run CUDA on a CUDA compatible machine, one needs to download the CUDA drivers, toolkit and SDK, and install them in that order. Then, after installation, if one wants to create his/her own project, then either he/she will copy the project template present in the SDK into the projects folder of SDK itself and work on it or create a folder at anywhere, but, be sure to include all the paths in the project IDE. These are some things which can easily anyone (if they forget about the path dependencies of CUDA). Thus, in this regard, the ‘**cuda-lib**’ utility is a step taken to solve the issues mentioned above. The contents of this utility and their usages will be described in detail in section 3. The installation details for this utility will be in section 2.

2. Installation

This utility is open-source and free to use and distribute. (For non-commercial purposes only) Since the entire source inside this utility are put under bash-scripts, there’s nothing really to install! However, one just needs to set his/her environment properly before starting to use this script. Hence, a fool-proof way of pre-cursors to using this utility will be as follows:

- a. Download the compressed file `cuda-lib-1.1.zip`.
- b. Extract it into your local directories.
- c. Go inside the directory ‘`cuda-lib-1.1/`’.
- d. Give execution permission to all the scripts inside this directory. (i.e. `chmod +x *`)
- e. Add the location of these scripts into your ‘`PATH`’ environment variable. (Better if you can add it in a start-up script so that you don’t have to do it again)
- f. Put the location of your ‘`C-compiler`’ in the line numbered 1044 in the script ‘`libGen.sh`’ (must be inside double quotes). If you already see another location being present in that line, please over-write on it.
- g. Repeat the step ‘f’ for the script ‘`cu.sh`’ as well, but at line numbered 269.
- h. Use the command ‘`updatedb`’ (probably as root user), in order for a faster and accurate search for CUDA files by these scripts.

A typical sequence of installation Linux commands will look like:

```
$ unzip cuda-lib.1.1.zip
$ cd cuda-lib-1.1
$ chmod +x *
$ echo export PATH=$PATH`pwd` >> ~/.bashrc  # if you are using ‘bash’ as your
                                           # default login shell.
                                           # Note that the pwd command is
                                           # encapsulated by back-quotes.

# put the location of C-compiler at the lines mentioned in steps ‘f’ and ‘g’.
$ updatedb                                # This might take SEVERAL minutes, so please be patient!
                                           # You might have to run this command as a root-user.
```

A word of caution!

If you are using cygwin, then the location of the compiler should be mentioned similar to the Windows absolute paths but with the Windows backward slashes (for directories) replaced by the Unix forward slashes. For eg., if the location of C-compiler is “C:\Program Files\Microsoft Visual Studio 8\VC\bin” in Windows, then put the following line “C:/Program Files/Microsoft

Visual Studio 8/VC/bin” while following the steps ‘f’ and ‘g’ above. If you are in Linux, then you just need to put the absolute path of the C-compiler for Linux in those lines mentioned.

3. Deliverables

‘cuda-lib’ comes with three simple yet very useful bash-scripts which, when used properly can put an end to the sufferings mentioned in the introduction section, above ☺. The three scripts and their purpose are described in short in the table 1 below and in detail in the sub-sections which follow.

Sl. No.	Scripts	Usage
1	cudaInstall	Like a One-Command-Install (OCI) script for CUDA.
2	cu.sh	Resolves the Path-Dependency (PD) issues with CUDA compiler.
3	libGen.sh	Generates well organized project folders for CUDA coding.

Table1. Deliverables in this utility with their brief usage.

What follows is a detailed description of each of the scripts mentioned in the table above, along with their usage.

3a. cudaInstall

As mentioned above, this is an OCI script for CUDA version 2.0. Instead of manually visiting the CUDA webpage and downloading the installers and then installing them, this script removes the manual intervention in this process. Additionally, this script also provides an option of downloading the appropriate cuda visual profiler into your local directory.

But, however, unlike CUDA web-page, this script lacks the ability to determine the type of OS in which it is executing and for this purpose, it relies on the user to input these commands. It displays a menu of all the supported OS types and asks for the user input and takes the necessary download and installation actions.

Usage: The usage of this script is as follows:

cudaInstall [-h, --help, -v, -V, -p, -P, -d, -r]

- h, --help Print this help message and exit.
- v Print the version number of this script and exit.
- V Exit before printing the version of CUDA which it downloads and installs.
- p Proxy option must be enabled.
- P Download and install the profiler for CUDA as well.
- d Only download the required files, don't install them.
- r Remove the downloaded files.

OS types Supported: The OS types supported by this script currently are as follows.

1. Windows XP 32 bits
2. Windows XP 64 bits
3. Windows Vista 32 bits
4. Windows Vista 64 bits
5. Linux 32 bits

- a) Redhat Enterprise Linux 4.x
 - b) Redhat Enterprise Linux 5.x
 - c) SUSE Linux Enterprise Desktop 10
 - d) OpenSUSE 10.2
 - e) OpenSUSE 10.3
 - f) Fedora 8
 - g) Ubuntu 7.04
 - h) Ubuntu 7.10
6. Linux 64 bits
- a) Redhat Enterprise Linux 4.x
 - b) Redhat Enterprise Linux 5.x
 - c) SUSE Linux Enterprise Desktop 10
 - d) OpenSUSE 10.2
 - e) OpenSUSE 10.3
 - f) Fedora 8
 - g) Ubuntu 7.04
 - h) Ubuntu 7.10

Dependencies: External utilities on which this script depends for a successful run are as follows.

1. An active internet connection. ☺
2. wget: for downloading the installation files from the internet.
3. unzip: for unzipping the '.zip' files.
4. tar: for unzipping the '.tar.gz' files.
5. http_proxy: if the proxy option is enabled, then one has to declare this environment variable to the proxy server of interest.

For more details please go through the preamble of this script.

3b. cu.sh

This script is a solution to the path dependency problem faced by the CUDA developers. After the CUDA installation, the user must declare the paths related to the CUDA libraries and header files and also the CUDA compiler. Even though this seems to be easier at the first look, this can sometimes produce errors during compilation (if they are set incorrectly) which can make people go crazy! So, this script finds the location of the CUDA headers and libraries and the compiler by itself and generates an installer script (roughly equivalent to 'make' in functionality) which can then be used to compile the whole project.

Usage: The usage of this script is as follows:

- ```
cu.sh [-h, --help, -r, -q, -l, -v] [-o <targetName>] [<nvcc_compiler_options>]
```
- h, --help    Print help message and exit.
  - r            Create the installer script and run it.
  - q            Run in silent mode.
  - l            Target is a library instead of an (default) executable.
  - v            Print the version number of this script and exit.
  - o            Specify custom target name, over-riding the defaults.

<targetName> Specify the name of the target, by over-riding the default value.  
By default, the name of the target will be the name of the main folder  
(if the target is an executable) and the name of the target will be  
the name of the main folder followed by '.a' (for linux) and '.lib'  
(for cygwin and windows) (if the target is a library).  
<nvcc\_compiler\_options> These are some of the compiler options provided by 'nvcc', keep all  
of these options only at the end. (inside double quotes)

Dependencies: External utilities on which this script depends are as follows:

1. locate: in order to locate for the specific files.

Rules: In order for the correct functionality of this script one MUST have to adhere to certain way of his/her project folder organization. The rules for such an organization are as follows:

1. This script will only compile for the projects which contain a collection of .c files or .cu files or both, but not for any other files. So, please make sure that you have only .c and .cu (along with .h files, of course) files as your source.
2. All source files (.cu, .c) must be put inside a folder named 'src'.
3. All header files (.h) must be put inside a folder named 'include'.
4. These 2 folders must in-turn reside in a folder which is named after the 'project' which you are building. This folder is called as main folder.
5. This script must be executed only from inside the main folder.
6. The executable created from the final compilation will have the same name as that of the main folder, and will be found in the main folder itself.
7. Do not pass -ccbin and -c as the nvcc compiler options, as these are used (as required) by this script itself.

For more details please go through the preamble of this script.

### 3c. libGen.sh

After creating the script 'cu.sh', I realized that the rules put-forward by this script were rather too much and very clumsy to follow! Hence, I decide to write another script which acts as a layer of abstraction between these rules and the user, thereby preventing the user from the conscious effort of following these rules! ☺ The solution is the script 'libGen.sh'. In simple words, this is an automated project folder generator i.e. it generates the required folders following the protocol of 'cu.sh' where the user can develop his/her code, compile, execute and debug it. Also, as a final part, can be distributed for others to use! This script currently supports 2 kinds of project folders to be created. One is a user-created CUDA library whose output will be a library file, and the other is a user-created CUDA utility whose output will be an executable. Both kinds have a slightly different folder organization which will be described in detail in sections '3d' and '3e'.

Thus, after executing this script, you can see a folder (with the name as specified by you for the project name) created in the current working directory. Now, you just need to keep on adding the source files to the 'src' directory and header files to the 'include' directory and all your documentations to the 'docs' directory. That's all folks! ☺

Usage: The usage of this script is as follows:

```
libGen.sh [-h, --help, -q, -v, -l, -w] [-o <projectName>] [-p <purpose>]
```

- h, --help    Print help message and exit.
- q            Run in silent mode.
- v            Print the version number of this script and exit.
- l            Project for the creation of a library.
- o            Specify custom project name, over-riding the defaults.  
By default, the name of the project will be 'project'. (If such a project already exists, then, it'll be named as 'project1' and so on until such a name doesn't exist)
- p            Specify the purpose of this project. This should be under double quotes.
- w            Over-write the project folder, if it exists. Except the source and header files in the directories 'src' and 'include', all others will be over-written. The default projects (ones which are created when the user doesn't specify a project name) are never overwritten by this option.

But the project name should NOT contain any spaces!

Dependencies: -None-

### **3d. Project folder for CUDA library**

This section describes the project folder organization created by 'libGen.sh' for a CUDA library. CUDA library means you want to develop a project whose output will be a library file so that others just need to add the header files provided by you and link to your particular library file and can then use your functions easily. The project folder organization will be as given in the figure1 below. Table2 will describe the details of each entity in the organization chart below. For more details regarding the organization the user is encouraged to go inside the folder and have a tour.

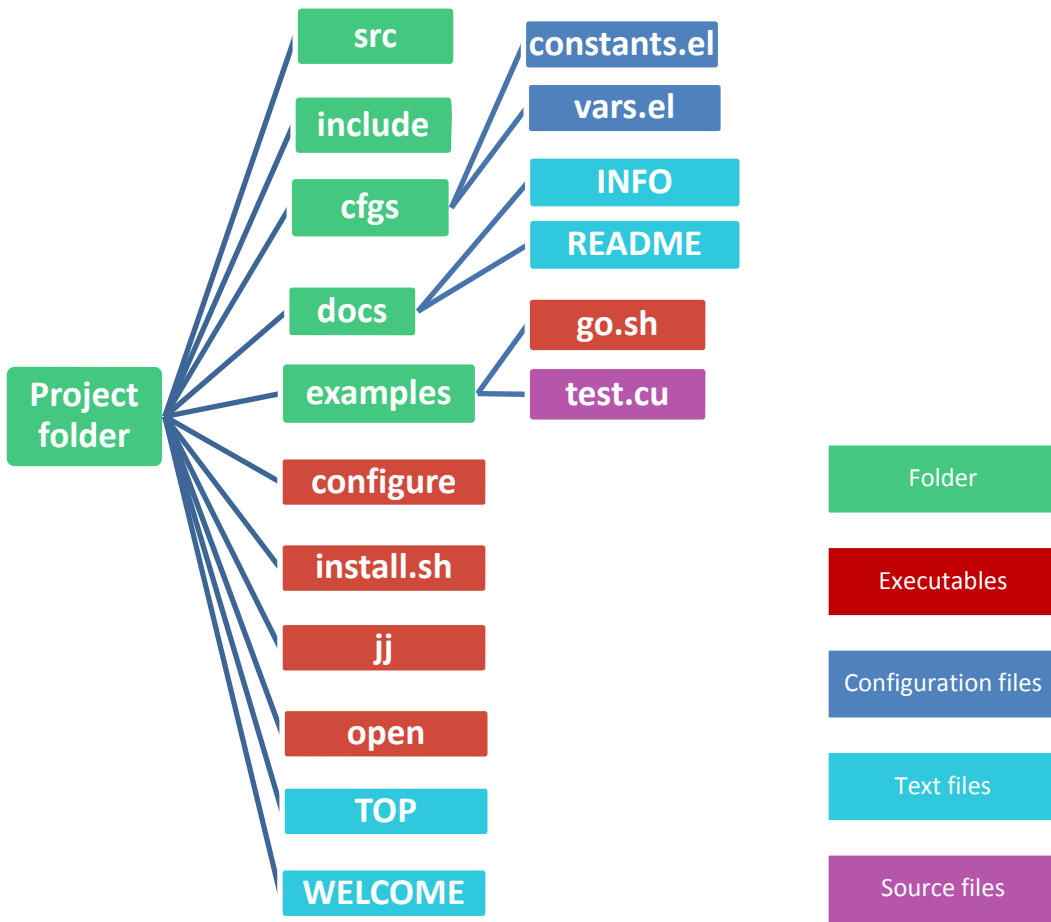


Figure1. Project organization chart for CUDA library.

| Sl. No. | Entity         | Description                                                                                                                                                                   |
|---------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1       | Project folder | The main folder created by 'libGen.sh' which contains all the project related files.                                                                                          |
| 2       | src            | Contains all the source files related to this project.                                                                                                                        |
| 3       | include        | Contains all the header files related to this project.                                                                                                                        |
| 4       | cfgs           | Contains some configuration files for 'emacs'. These help create an 'emacs' environment which is highly configured for syntax highlighting, source compilation and execution. |
| 5       | docs           | Contains the project documentation.                                                                                                                                           |
| 6       | examples       | Contains a source file written in order to test the functions which are being added to the library.                                                                           |
| 7       | configure      | First time executed script for the project in order to set the project environment and also in order to update any changes performed inside the project folder.               |
| 8       | install.sh     | Installer script for the project.                                                                                                                                             |

|    |              |                                                                                                                                                                                                                                                     |
|----|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 9  | jj           | A wrapper over ‘open’ which can dynamically update the changes done in the project.                                                                                                                                                                 |
| 10 | open         | Used to launch the customized ‘emacs’ environment for source development.                                                                                                                                                                           |
| 11 | TOP          | A dummy file created in order to make sure that the current folder (where it resides) is the main folder for the project. Hence, never create any files named ‘TOP’ in the project folder!                                                          |
| 12 | WELCOME      | This is a sort of welcome-text (and a pointer for the subsequent documentation) for the users who enter into this project folder for the first time.                                                                                                |
| 13 | constants.el | Contains some basic configurations which remain the same for all projects created using ‘libGen.sh’.                                                                                                                                                |
| 14 | vars.el      | Contains project specific configurations for the current project.                                                                                                                                                                                   |
| 15 | INFO         | Contains very useful information regarding some of the commands to be used and their ‘emacs’ shortcuts.                                                                                                                                             |
| 16 | README       | Contains general information regarding the project organization and release dates and versions.                                                                                                                                                     |
| 17 | go.sh        | This is the script which compiles the example source file ‘test.cu’ by linking the library file from the current project, in order to generate an executable which can then be run to make sure the correctness of the particular library function. |
| 18 | test.cu      | The example source file which is used to call the library functions in order to verify their correctness.                                                                                                                                           |

Table2. CUDA library project folder organization.

### 3e. Project folder for CUDA utility

This section describes the project folder organization created by ‘libGen.sh’ for a CUDA utility. CUDA utility means you want to develop a project whose output will be an executable so that others just need to launch this executable for a particular functionality. The project folder organization will be as given in the figure2 below. Table3 will describe the details of each entity in the organization chart below. For more details regarding the organization the user is encouraged to go inside the folder and have a tour.



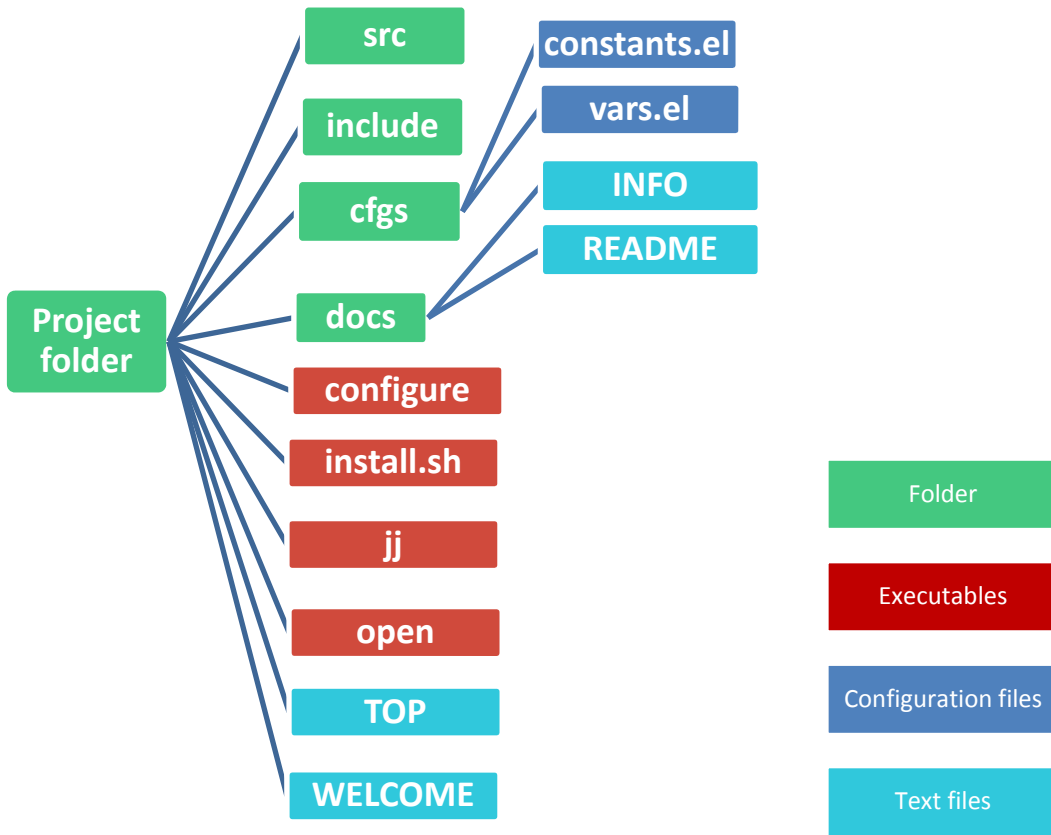


Figure2. Project organization chart for CUDA utility.

| Sl. No. | Entity         | Description                                                                                                                                                                   |
|---------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1       | Project folder | The main folder created by 'libGen.sh' which contains all the project related files.                                                                                          |
| 2       | src            | Contains all the source files related to this project.                                                                                                                        |
| 3       | include        | Contains all the header files related to this project.                                                                                                                        |
| 4       | cfgs           | Contains some configuration files for 'emacs'. These help create an 'emacs' environment which is highly configured for syntax highlighting, source compilation and execution. |
| 5       | docs           | Contains the project documentation.                                                                                                                                           |
| 6       | configure      | First time executed script for the project in order to set the project environment and also in order to update any changes performed inside the project folder.               |
| 7       | install.sh     | Installer script for the project.                                                                                                                                             |
| 8       | jj             | A wrapper over 'open' which can dynamically update the changes done in the project.                                                                                           |

|    |              |                                                                                                                                                                                            |
|----|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 9  | open         | Used to launch the customized ‘emacs’ environment for source development.                                                                                                                  |
| 10 | TOP          | A dummy file created in order to make sure that the current folder (where it resides) is the main folder for the project. Hence, never create any files named ‘TOP’ in the project folder! |
| 11 | WELCOME      | This is a sort of welcome-text (and a pointer for the subsequent documentation) for the users who enter into this project folder for the first time.                                       |
| 12 | constants.el | Contains some basic configurations which remain the same for all projects created using ‘libGen.sh’.                                                                                       |
| 13 | vars.el      | Contains project specific configurations for the current project.                                                                                                                          |
| 14 | INFO         | Contains very useful information regarding some of the commands to be used and their ‘emacs’ shortcuts.                                                                                    |
| 15 | README       | Contains general information regarding the project organization and release dates and versions.                                                                                            |

Table3. CUDA utility project folder organization.

#### 4. Limitations (TODO list)

Some of the limitations with the current version of this utility are as follows. These will, by default, sit in my TODO list.

1. ‘cu.sh’ and ‘libGen.sh’ depend on the user to type-in the location of the C-compilers into their respective lines mentioned before.
2. These scripts currently don't check apriori, for any previous CUDA installations.
3. You need to work on either the linux platform or on the linux emulators (viz. cygwin) to use these scripts.
4. ‘cudaInstall’ is unable to check for the OS type currently.
5. An initial configure script which performs all the tasks given in the section 2.

#### 6. Contact:

For any queries, bugs and suggestions feel free to contact me at [snanditale@nvidia.com](mailto:snanditale@nvidia.com). Your feedbacks and suggestions are very critical in development of this utility and will be highly appreciated in this regard.

Thejaswi. N. S