UseLATEX.cmake: LaTeX Document Building Made Easy

Kenneth Moreland

Version 2.4.8

Contents

1 Description

Compiling LaTeX files into readable documents is actually a very involved process. Although CMake comes with FindLATEX.cmake, it does nothing for you other than find the commands associated with LaTeX. I like using CMake to build my LaTeX documents, but creating targets to do it is actually a pain. Thus, I've compiled a bunch of macros that help me create targets in CMake into a file I call "UseLATEX.cmake." Here are some of the things UseLATEX.cmake handles:

- Runs LaTeX multiple times to resolve links.
- Can run bibtex, makeindex, and makeglossaries to make bibliographies, indexes, and/or glossaries.
- Optionally runs configure on your LaTeX files to replace `@VARIABLE@` with the equivalent CMake variable.
- Automatically finds png, jpeg, eps, pdf, svg, tiff, gif, bmp, and other image files and converts them to formats latex and pdflatex understand.

2 Download

UseLATEX.cmake is currently posted to the CMake Wiki at

http://public.kitware.com/Wiki/CMakeUserUseLATEX.

3 Basic Usage

Using UseLATEX.cmake is easy. For a basic LaTeX file, simply include the file in your CMakeLists.txt and use the `add_latex_document` command to make targets to build your document. For an example document in the file MyDoc.tex, you could establish a build with the following simple CMakeLists.txt.

```
project(MyDoc NONE)

include(UseLATEX.cmake)
add_latex_document(MyDoc.tex)
```

The `add_latex_document` adds the following targets to create a readable document from MyDoc.tex:

dvi Creates MyDoc.dvi.

pdf Creates MyDoc.pdf using pdflatex. Requires the `PDFLATEX_COMPILER` CMake variable to be set.

ps Creates MyDoc.ps. Requires the `DVIPS_CONVERTER` CMake variable to be set.

safepdf Creates MyDoc.pdf from MyDoc.ps using ps2pdf. Many publishers prefer pdfs are created this way. Requires the `PS2PDF_CONVERTER` CMake variable to be set.

html Creates html pages. Requires the `HTLATEX_COMPILER` CMake variable to be set.

clean To CMake's default clean target, the numerous files that LaTeX often generates are added.

auxclean Deletes the auxiliary files from LaTeX, but not the generated input files. Sometimes LaTeX gets itself in a bad state where the auxiliary files need to be deleted to successfully build again, and this target does that without also deleting other build files (such as converted image files or files from unrelated targets in the same directory).

One caveat about using UseLATEX.cmake is that you are required to do an out-of-source build. That is, CMake must be run in a directory other than the source directory. This is necessary as latex is very picky about file locations, and the relative locations of some generated or copied files can only be maintained if everything is copied to a separate directory structure. For more details and hints on workarounds, see the "Why does UseLATEX.cmake have to copy my tex files?" frequently asked question in Section 6.5.

3.1 Using a Bibliography

For any technical document, you will probably want to maintain a BibTeX database of papers you are referencing in the paper. You can incorporate your .bib files by adding them after the `BIBFILES` argument to the `add_latex_document` command.

```
add_latex_document(MyDoc.tex BIBFILES MyDoc.bib)
```

This will automatically add targets to build your bib file and link it into your document. To use the BibTeX file in your LaTeX file, just do as you normally would with `\cite` commands and bibliography commands:

```
\bibliographystyle{plain}
\bibliography{MyDoc}
```

You can list as many bibliography files as you like.

3.2 Incoporating Images

To be honest, incorporating images into LaTeX documents can be a real pain. This is mostly because the format of the images needs to depend on the version of LaTeX you are running (latex vs. pdflatex). With these CMake macros, you only need to convert your raster graphics to png or jpeg format and your vector graphics to eps or pdf format. Place them all in a common directory (e.g. images) and then use the `IMAGE_DIRS` option to the `add_latex_document` macro to point to them. UseLATEX.cmake will take care of the rest.

```
add_latex_document(MyDoc.tex
  BIBFILES MyDoc.bib
  IMAGE_DIRS images
  )
```

If you want to break up your image files in several different directories, you can do that, too. Simply provide multiple directories after the `IMAGE_DIRS` option.

```
add_latex_document(MyDoc.tex
  BIBFILES MyDoc.bib
  IMAGE_DIRS icons figures
  )
```

Alternatively, you could list all of your image files separatly with the `IMAGES` option.

```
set(MyDocImages
  logo.eps
  icons/next.png
  icons/previous.png
  figures/flowchart.eps
  figures/team.jpeg
  )
add_latex_document(MyDoc.tex
  IMAGES ${MyDocImages}
  )
```

For every image file specified and found with the `IMAGE_DIRS` and `IMAGES` options, UseLATEX.cmake adds makefile targets to use ImageMagick's magick or convert to convert the file types to those appropriate for the build.[1] If you do not have ImageMagick, you can get it for free from http://www.imagemagick.org. CMake will also give you a `LATEX_SMALL_IMAGES` option that, when on, will downsample raster images. This can help speed up building and viewing documents. It will also make the output image sizes smaller.

UseLATEX.cmake will occasionally use a conversion program other than ImageMagick's magick. For example, ps2pdf will be used when converting eps to pdf to get around a problem where ImageMagick drops the bounding box information. When available, the pdftops from the Poppler utilities will be used to convert pdf to eps because it better preserves vector graphics and color spaces. At any rate, you do not need to worry about setting the appropriate image conversion program. UseLATEX.cmake will automatically select the best one and issue errors or warnings if there is a problem.

The `IMAGE_DIRS` option tries to identify image files by their extensions. The current list of image extensions UseLATEX.cmake checks for is: .bmp, .bmp2, .bmp3, .dcm, .dcx, .ico, .gif, .jpeg, .jpg, .eps, .pdf, .pict, .png, .ppm, .tif, and .tiff. If you are trying to use an image format that is supported by ImageMagick but is not recognized by UseLATEX.cmake, you can specify the files directly with the `IMAGES` option instead. UseLATEX.cmake will assume that any file specified with the `IMAGES` option is an image file regardless of its extension.

Both the `IMAGE_DIRS` and `IMAGES` can be used together. The combined set of image files will be processed. If you wish to provide a separate eps file and pdf or png file, that is OK, too. UseLATEX.cmake will handle that by copying over the correct file instead of converting.

Depending on what program is launched to build your LaTeX file (either latex or pdflatex, and UseLATEX.cmake supports both), a particular format for your image is required. As stated, UseLATEX.cmake handles the necessary conversions for you. However, you will not know in advance what file extension is used on the image. That is no problem. Simply leave out the file extension in the

[1] The convert program was essentially renamed magick in ImageMagick 7.0. Most, but not all, recent installations provide both. UseLATEX.cmake looks for both just in case.

file name argument to `\includegraphics` and LaTeX will find the file with the appropriate extension for you.

Note that in order to ensure that the resulting image files are placed in the appropriate directory, you are required to give *relative* paths for images and image directories. For example, `IMAGE_DIRS ${CMAKE_CURRENT_SOURCE_DIR}/images` will fail. Use `IMAGE_DIRS images` instead.

3.3 Selecting a Default Build

By default, when you use `add_latex_document` and then run make with no arguments, pdflatex is used to create a pdf file. You can of course always specify a target described at the top of Section 3 to build a different document format. However, for convenience you can change the default build.

UseLATEX.cmake defines the CMake variable `LATEX_DEFAULT_BUILD` that controls which build is performed by default. Valid values for this variable are `pdf`, `dvi`, `ps`, `safepdf`, and `html`. This variable is usually initialized to `pdf`, but you can override this behavior by setting the `LATEX_DEFAULT_BUILD` environment variable before the first configuration. Thus, if you have a preference for a particular default build, you can set your system environment to use it by default for all UseLATEX.cmake builds.

3.4 Force a Type of Build

UseLATEX.cmake does its best to make LaTeX builds as portable as possible, but there might be a number of technical reasons why a particular document can only be built using one type of system. If that is the case, it is best if the configuration only supports one type of build.

`add_latex_document` has several options to force the document generation to a particular type of build. If you give the option `FORCE_PDF`, only the pdf targets that use the pdflatex command are created.

```
add_latex_document(MyDoc.tex
  BIBFILES MyDoc.bib
  IMAGE_DIRS images
  FORCE_PDF
  )
```

Likewise, the `FORCE_DVI` option restricts `add_latex_document` to targets that use the latex command. In addition to building dvi files, `FORCE_DVI` also allows ps generation from the dvi files and "safe" pdf generation from the ps files.

```
add_latex_document(MyDoc.tex
  BIBFILES MyDoc.bib
  IMAGE_DIRS images
```

```
  FORCE_PS
  )
```

Finally, the FORCE_HTML option will restrict targets that are used for html generation.

```
add_latex_document(MyDoc.tex
  BIBFILES MyDoc.bib
  IMAGE_DIRS images
  FORCE_HTML
  )
```

The behavior is undefined if more than one force option is given.

3.5 Create Nothing by Default

Sometimes it is desirable to disable the building of your LaTeX document by default (that is, not build it with the all target). This is convenient when including LaTeX documentation with some other source to build such as when you are documenting a library. To remove all targets from the default, simply add the EXCLUDE_FROM_ALL option to add_latex_document.

```
add_latex_document(MyDoc.tex
  BIBFILES MyDoc.bib
  IMAGE_DIRS images
  EXCLUDE_FROM_ALL
  )
```

3.6 SyncTeX-Enabled Editors

Some implementations of LaTeX compilers have a feature called SyncTeX that allows an editor or viewer to link between the compiled version of the document (such as a pdf) and the original LaTeX source code. The most common way to do this is to add -synctex=1 to the pdflatex command. This will create a file named ⟨*base-name*⟩.synctex.gz where each part of the final document points to the original LaTeX files.

However, there is a problem. UseLATEX.cmake copies all of the input LaTeX source files to an out-of-source build directory (see Section 6.5 for more information on why). But the LaTeX compiler does not know that. Thus, the created ⟨*base-name*⟩.synctex.gz will point to the temporary files in the build directory rather than your original source files.

UseLATEX.cmake can add commands to the make targets that "correct" the ⟨*base-name*⟩.synctex.gz. To add these commands, simply turn on the LATEX_USE_SYNCTEX in ccmake or equivalent CMake configuring tool. When this

option is on, the -synctex=1 argument is added to the LaTeX compile commands (settable with the LATEX_SYNCTEX_FLAGS variable) and a command is added to targets that will find files in ⟨*base-name*⟩.synctex.gz and change their paths to the original files in the source directory.

4 Package Support

Modern LaTeX distributions provide a great many packages to provide additional features to the document building process. A great many more packages are available in online package distributions. The vast majority of these packages provide features that are self contained within the LaTeX call itself. That is, the build process does not have to change to accommodate these packages.

That said, there are a small number of packages that require supplementary programs to be run or to otherwise change the build process. These packages require special options to `add_latex_document`, which are documented here.

4.1 Making an Index

You can make an index in a LaTeX document by using the makeidx package. However, this package requires you to run the makeindex program. Simply add the `USE_INDEX` option anywhere in the `add_latex_document` arguments, and makeindex will automatically be added to the build.

```
add_latex_document(MyDoc.tex
  BIBFILES MyDoc.bib
  IMAGE_DIRS images
  USE_INDEX
  )
```

4.2 Making Multiple Indexes

The multind package allows you to create multiple indexes in a single LaTeX document. For example, when documenting a software library you might want to have a general index of terms and a second index of function names.

The way the multind package works is that it creates a separate index file for each of the indexes being created, and the makeindex program must be run independently on each of them. To get UseLATEX.cmake to run makeindex on all of the required index file, list all of the indexes created with the `INDEX_NAMES` option of `add_latex_document`. For example, in a LaTeX document that declares two indexes like the following

```
\usepackage{multind}
\makeindex{general}
\makeindex{functions}
```

you would name the indexes in `add_latex_document` like the following.

```
add_latex_document(MyDoc.tex
  BIBFILES MyDoc.bib
  IMAGE_DIRS images
  USE_INDEX
  INDEX_NAMES general functions
  )
```

4.3 Making a Glossary

There are multiple ways to make a glossary in a LaTeX document, but the glossaries package provides one of the most convenient ways of doing so. Like the makeidx package, glossaries requires running makeindex or xindy for building auxiliary files. However, building the glossary files can be more complicated as there can be different sets of glossary files with different extensions. UseLATEX.cmake will handle that for you. It does it in a way similar to the makeglossary command, but in a more portable way. Simply add the `USE_GLOSSARY` option anywhere in the `add_latex_document` arguments, and the glossary creating will be handled for you.

```
add_latex_document(MyDoc.tex
  BIBFILES MyDoc.bib
  IMAGE_DIRS images
  USE_GLOSSARY
  )
```

4.4 Nomenclature Support

The nomencl package provides a mechanism to collect nomenclature and print it together in a single section. The nomencl behaves very similarly to glossaries (described in Section 4.3) including running the makeindex command. However, the arguments to makeindex are a bit different (to avoid clashes with creating glossaries), and unfortunately nomencl provides no hints in the auxiliary file about it. Thus, UseLATEX.cmake provides a special `USE_NOMENCL` option to `add_latex_document` to add the necessary commands to build the nomenclature.

```
add_latex_document(MyDoc.tex
  BIBFILES MyDoc.bib
  IMAGE_DIRS images
  USE_NOMENCL
  )
```

It should be noted that this feature only works with nomencl version 4.0 or later. The nomencl package changed how makeindex is called to make it compatible with indices and glossaries. The correct version of nomencl is easily identified as the one that uses the `\makenomenclature` and `\printnomenclature` commands (as opposed to the old `\makeglossary` and `\printglossary` commands). If you are using an older version of nomencl, you are best off to update for a number of reasons.

4.5 multibib Support

The multibib package provides a mechanism to create a set of distinct bibliographies that are not necessarily associated with sections of the document. Part of the operation of this package creates multiple LaTeX auxiliary files that need to be processed independently with BibTeX. Consequently, the build needs to be modified to run BibTeX multiple times with different inputs. This can be achieved with the `MULTIBIB_NEWCITES` argument to `add_latex_document`.

As an example, consider the following usage of the multibib package, partially taken from its documentation. It creates a set of distinct citation commands named `own`, `submitted`, and `internal` with the section heads `Own Work`, `Submitted Work`, and `Master Theses and Ph.D. Theses` respectively. They collectively use the bibliography files own.bib, submitted.bib, techreports.bib, and theses.bib.

```
\newcites{own,submitted,internal}%
  {Own Work,%
   Submitted Work,%
   {Technical Reports, Master Theses and Ph.D. Theses}}
```

```
\bibliographyown{own.bib}
```

```
\bibliographysubmitted{submitted.bib}
```

```
\bibliographyinternal{techreports.bib,theses.bib}
```

The three suffixes specified to the `\newcite` command and the four bibliography files referenced must all be specified in the `add_latex_document` command with the `MULTIBIB_NEWCITES` and `BIBFILES` arguments, respectively.

```
add_latex_document(MyDoc.tex
  BIBFILES own.bib submitted.bib techreports.bib theses.bib
  MULTIBIB_NEWCITES own submitted internal
  )
```

4.6 biblatex Support

The biblatex package provides an alternate mechanism for building bibliographies that has many options not available to the standard bibliography commands. The package (typically) requires an external program named biber, which is an alternative to the standard bibtex command.

Thus, to support the biblatex package, the build system must run biber instead of bibtex. This is done simply with UseLATEX.cmake by adding the `USE_BIBLATEX` option to `add_latex_document`.

```
add_latex_document(MyDoc.tex
  BIBFILES MyDoc.bib
  USE_BIBLATEX
  )
```

5 Advanced Configurations

This document has heretofore described using UseLATEX.cmake for a single LaTeX document and associated files (bibliographies, images, indices, etc.). However there are many configurations to document building that extend beyond this simple scenario including multipart files, multiple documents, and depended builds.

5.1 Multipart LaTeX Files

Often, it is convenient to split a LaTeX document into multiple files and use the LaTeX `\input` or `\include` command to put them back together. To do this, all the files have to be located together. UseLATEX.cmake can take care of that, too. Simply add the `INPUTS` argument to `add_latex_document` to copy these files along with the target tex file. Build dependencies to these files is also established.

```
add_latex_document(MyDoc.tex
  INPUTS Chapter1.tex Chapter2.tex Chapter3.tex Chapter4.tex
  BIBFILES MyDoc.bib
  IMAGE_DIRS images
  USE_INDEX
  )
```

As far as UseLATEX.cmake is concerned, input files do not necessarily have to be tex files. For example, you might be including the contents of a text file into your document with the `\VerbatimInput` command of the fancyvrb package. In fact, you could also add graphic files as inputs, but you would not get the extra conversion features described in Section 3.2.

5.2 Configuring LaTeX Files

Sometimes it is convenient to control the build options of your tex file with CMake variables. You can achieve this by using the `CONFIGURE` argument to `add_latex_document`.

```
add_latex_document(MyDoc.tex
  INPUTS Chapter1.tex Chapter2.tex Chapter3.tex Chapter4.tex
  CONFIGURE MyDoc.tex
  BIBFILES MyDoc.bib
  IMAGE_DIRS images
  USE_INDEX
  )
```

In the above example, in addition to copying MyDoc.tex to the binary directory, UseLATEX.cmake will configure MyDoc.tex. That is, it will find all occurrences of `@`*`VARIABLE`*`@` and replace that string with the current CMake variable *`VARIABLE`*.

With the `CONFIGURE` argument you can list the target tex file (as shown above) as well as any other tex file listed in the `INPUTS` argument.

```
add_latex_document(MyDoc.tex
  INPUTS Ch1Config.tex Ch1.tex Ch2Config.tex
         Ch2.tex Ch3Config Ch3.tex
  CONFIGURE Ch1Config.tex Ch2Config.tex Ch3Config.tex
  BIBFILES MyDoc.bib
  IMAGE_DIRS images
  USE_INDEX
  )
```

Be careful when using the `CONFIGURE` option. Unfortunately, the `@` symbol is used by LaTeX in some places. For example, when establishing a tabular environment, an `@` is used to define the space between columns. If you use it more than once, then UseLATEX.cmake will erroneously replace part of the definition of your columns for a macro (which is probably an empty string). This can be particularly troublesome to debug as LaTeX will give an error in a place that, in the original document, is legal. Hence, it is best to only configure tex files that contain very little text of the actual document and instead are mostly setup and options.

5.3 Building Multiple LaTeX Documents

The most common use for UseLATEX.cmake is to build a single document, such as a paper you are working on. However, some use cases involve building several documents at one time.

Multiple LaTeX documents in the same CMake project can be created by simply calling `add_latex_document` multiple times. Each call to `add_latex_document` will create its own set of unique targets that will be added as dependencies of dvi, pdf, ps, safepdf and html.

Consider the following code.

```
add_latex_document(MyDoc1.tex)
add_latex_document(MyDoc2.tex)
```

In the example above, the first call to `add_latex_document` will create targets named MyDoc1_dvi, MyDoc1_pdf, MyDoc1_ps, etc. whereas the second call will create targets named MyDoc2_*. Calling dvi, pdf, etc. will execute the respective targets for the two documents.

The `EXCLUDE_FROM_DEFAULTS` option suppresses these links to the document's targets.

```
add_latex_document(MyDoc1.tex)
add_latex_document(MyDoc2.tex)
add_latex_document(MyDoc3.tex EXCLUDE_FROM_DEFAULTS)
```

In this augmented example, MyDoc1 and MyDoc2 are built when targets such as dvi and pdf are called, but MyDoc3 is not. Note, however, that in this example MyDoc3 is still built as part of the all target that CMake sets as the default build target. Use `EXCLUDE_FROM_ALL` to remove a document from the default all build. `EXCLUDE_FROM_ALL` and `EXCLUDE_FROM_DEFAULTS` can be used together or independently.

An issue that can come up in larger builds with multiple LaTeX documents is a name collision. If two subdirectories each have a LaTeX document with the same .tex file in it, then the respective calls to `add_latex_document` will create the same target names, which CMake does not allow. One way around this problem is to rename the files to be unique (so that `add_latex_document` will create unique target names). But a more convenient way is to use the `TARGET_NAME` option to change the target names. For example, consider the following use of `TARGET_NAME`.

```
add_latex_document(doc.tex TARGET_NAME MyDoc1)
```

This will change the behavior of `add_latex_document` to create targets named MyDoc1_dvi, MyDoc1_pdf, MyDoc1_ps, etc. instead of doc_dvi, doc_pdf, doc_ps, etc.

5.4 Identifying Dependent Files

In some circumstances, CMake's configure mechanism is not sufficient for creating input files. Input LaTeX files might be auto-generated by any number of other mechanisms.

If this is the case, simply add the appropriate CMake commands to generate the input files, and then add that file to the DEPENDS option of `add_latex_document`. To help you build the CMake commands to place the generated files in the correct place, you can use the LATEX_GET_OUTPUT_PATH convenience function to get the output path.

```
latex_get_output_path(output_dir)

add_custom_command(OUTPUT ${output_dir}/generated_file.tex
  COMMAND tex_file_generate_exe
  ARGS ${output_dir}/generated_file.tex
  )

add_latex_document(MyDoc.tex DEPENDS generated_file.tex)
```

6 Frequently Asked Questions

This section includes resolutions to common questions and issues concerning use of UseLATEX.cmake and with LaTeX in general.

6.1 How do I process LaTeX files on Windows?

I have successfully used two different ports of LaTeX for windows: the cygwin port (http://www.cygwin.com/) and the MikTeX port (http://www.miktex.org/).

If you use the cygwin port of LaTeX, you must also use the cygwin port of CMake, make, and ImageMagick. If you use the MikTeX port of LaTeX, you must use the CMake from http://www.cmake.org/HTML/Download.html, the ImageMagick port from http://www.imagemagick.org/script/binary-releases.php#windows, and a native build tool like MSVC or the GNU make port at http://unxutils.sourceforge.net/. *Do not use the "native" CMake program with any cygwin programs or the cygwin CMake program with any non-cygwin programs.* This issue at hand is that the cygwin ports create and treat filenames differently then other windows programs.[2]

Also be aware that if you have images in your document, there are numerous problems that can occur on Windows with the ImageMagick convert program. See Section 6.7 for more information.

[2] If you are careful, you can use the cygwin version of make with the windows ports of CMake, LaTeX, and ImageMagick. It is an easy way around the problems described in Section 6.7.

6.2 How do I process LaTeX files on Mac OS X?

Using LaTeX on Mac OS X is fairly straightforward because this OS is built on top of Unix. By using the Terminal program or X11 host, you can run LaTeX much like any other Unix variant. The only real issue is that LaTeX and some of the supporting programs like CMake and ImageMagick are not typically installed (whereas on Linux they often are).

Most applications port fairly easily to Mac OS so long as you are willing to use them as typical Unix or X11 programs. To make things even easier, I recommend taking advantage of a Mac porting project to make this process even easier. MacPorts (http://www.macports.org) is a good tool providing a comprehensive set of tool ports including LaTeX, CMake, and ImageMagick. The fink project and FinkCommander (http://finkcommander.sourceforge.net/) is a similar although less active project.

6.3 How do I process with XeLaTeX?

UseLATEX.cmake was not designed with XeLaTeX in mind, but the interface of that program is similar enough to LaTeX that you should be able to use it. Simply change the `PDFLATEX_COMPILER` CMake variable to the xelatex program and build the pdf target.

6.4 How do I process with LuaLaTeX?

UseLATEX.cmake was not designed with LuaLaTeX in mind, but the interface of that program is similar enough to LaTeX that you should be able to use it. Simply change the `PDFLATEX_COMPILER` CMake variable to the lualatex program and build the pdf target.

6.5 Why does UseLATEX.cmake have to copy my tex files?

UseLATEX.cmake cannot process your tex file without copying it. As explained in Section 3, LaTeX is very picky about file locations. The relative locations of files that your input files point to, and all but the most simple LaTeX files point to other files, must remain consistent.

UseLATEX.cmake will often have to modify at least one file either through configurations or image format and size conversions. When creating new files, UseLATEX.cmake will have to copy either all of the files or none of the files. Since configuring and writing over an original file is unacceptable, UseLATEX.cmake forces you to configure it such that LaTeX builds in a different directory than where you have placed the original. If you do not specify a seperate directory, you get an error like the following.

```
CMake Error at UseLATEX.cmake:377 (MESSAGE):
  LaTeX files must be built out of source or you must set
  LATEX_OUTPUT_PATH.
```

The best way around this problem is do an "out of source" build, which is really the preferred method of using CMake in general. To do an out of source build, create a new build directory, go to that directory, and run cmake from there, pointing to the source directory.

If for some reason an out of source build is not feasable or desireable, you can set the `LATEX_OUTPUT_PATH` variable to a directory other than . (the local directory). If you are building a LaTeX document in the context of a larger project for which you wish to support in source builds, consider pragmatically setting the `LATEX_OUTPUT_PATH` CMake cache variable from within your CMakeLists.txt.

6.6 How can LaTeX find a file not a tex, image, or bibliography?

The most common files included from a LaTeX tex file are other tex files, images, and bibliographies, all of which are handled with options to `add_latex_document`.

But what happens if the LaTeX build includes some other type of file? For example, the fancyvrb package can import a text file with the `\VerbatimInput` command to be formatted in a teletype font. Other examples occur, such as program formatting packages that can read in source code files.

As far as UseLATEX.cmake is concerned, these types of files are simply other inputs to LaTeX and handled in the same way as included tex files. They can be included by adding them to the `INPUTS` argument as described in Section 5.1.

If an inputted file does not immediately exist but is generated by some other process, then the file should be generated in the output directory and added to the `DEPENDS` of the build as described in Section 5.4.

6.7 Why is convert failing on Windows?

Assuming that you have correctly downloaded and installed an appropriate version of ImageMagick (as specified in Section 6.1), there are several other problems that users can run into the created build files attempt to run the magick or convert program.

A common error is that magick or convert not finding a file that is clearly there.

```
convert: unable to open image 'filename'
```

If you notice that the drive letter is stripped off of the filename (i.e. C:), then you are probably mixing the Cygwin version of convert with the non-cygwin CMake. The cygwin version of convert uses the colon (:), as a directory separator for inputs. Thus, it assumes the output file name is really two input files separated by the colon. Switch to the non-cygwin port of ImageMagick to fix this.

If you are using nmake, you may also see the following error:

```
convert.exe: unable to open image 'C:': Permission denied.
```

I don't know what causes this error, but it appears to have something to do with some strange behavior of nmake when quoting the convert executable. The easiest solution is to use a different build program (such as make or MSVC's IDE or a unix port of make). If anyone finds away around this problem, please contribute back.

Another possible error seen is

```
Invalid Parameter - filename
```

This is probably because CMake has found the wrong convert program. Windows is installed with a program named convert in %SYSTEMROOT%\system32. This convert program is used to change the filesystem type on a hard drive. Since the windows convert is in a system binary directory, it is usually found in the path before the installed ImageMagick convert program. (Don't get me started about the logic behind this.) Make sure that the `IMAGEMAGICK_CONVERT` CMake variable is pointing to the correct convert program. Or better yet, make sure you have ImageMagick 7.0 or higher and use the magick program instead of convert. Recent versions of UseLATEX.cmake should give a specific warning about this with instructions on how to fix it.

6.8 How do I automate plot generation with command line programs?

LaTeX is often used in conjunction with plotting programs that run on the command line like gri or gnuplot. Although there is no direct support for these programs in UseLATEX.cmake, it is straightforward to use these programs.

One way to use a plotting program is simply to run it yourself to generate the plot and then incorporate the image file into your LaTeX document as you would any other image file (see Section 3.2). This the easiest way to incorporate these plots since it does not require additional CMake code. It also ensures consistency of how the plot looks (often the plots will look different if created on different platforms), and it provides the opportunity to edit the image to make it look better for publication.

Another way to use these plotting programs is to automatically run them when building the LaTeX document. This is convenient if you frequently change the data you are plotting or if you are creating many plots. To automate running the plotting program build one or more targets to generate these files and then add these targets as LaTeX dependencies (see Section 5.4 for information on adding dependencies). Here is an example of creating the targets for converting a directory of gri files and incorporating the resulting files in a LaTeX document.

```
# Set GRI executable
set(GRI_COMPILE "/usr/bin/gri")
# Set the location of data files
set(DATA_DIR data)
# Set the location of the directory for image files
set(IMAGE_DIR graphics)

# Get a list of gri files
file(GLOB_RECURSE GRI_FILES "*.gri")

foreach(file ${GRI_FILES})
  get_filename_component(basename "${file}" NAME_WE)
  # Replace stings in gri file so data files can be found
  file(READ
    ${CMAKE_CURRENT_SOURCE_DIR}/${IMAGE_DIR}/${basename}.gri
    file_contents
    )
  string(REPLACE "${DATA_DIR}" "${IMAGE_DIR}/${DATA_DIR}"
    changed_file_contents ${file_contents}
    )
  file(WRITE
    ${CMAKE_CURRENT_BINARY_DIR}/${IMAGE_DIR}/${basename}.gri
    ${changed_file_contents}
    )

  # Command to run gri
  if(GRI_COMPILE)
    add_custom_command(
      OUTPUT
        ${CMAKE_CURRENT_BINARY_DIR}/${IMAGE_DIR}/${basename}.eps
      DEPENDS
        ${CMAKE_CURRENT_BINARY_DIR}/${IMAGE_DIR}/${basename}.gri
        ${CMAKE_CURRENT_BINARY_DIR}/${IMAGE_DIR}/${DATA_DIR}
      COMMAND
        ${GRI_COMPILE}
      ARGS
        -output
        ${CMAKE_CURRENT_BINARY_DIR}/${IMAGE_DIR}/${basename}.eps
        ${CMAKE_CURRENT_BINARY_DIR}/${IMAGE_DIR}/${basename}.gri
      )
  endif()
  # Make a list of all gri files (for ADD_LATEX_DOCUMENT depend)
  set(ALL_GRI_FILES ${ALL_GRI_FILES}
    ${CMAKE_CURRENT_BINARY_DIR}/${IMAGE_DIR}/${basename}.eps
```

```
    )
endforeach(file)

# Copy over all data files needed to generate gri graphs
add_custom_command(
  OUTPUT  ${CMAKE_CURRENT_BINARY_DIR}/${IMAGE_DIR}/${DATA_DIR}
  DEPENDS ${CMAKE_CURRENT_SOURCE_DIR}/${IMAGE_DIR}/${DATA_DIR}
  COMMAND ${CMAKE_COMMAND} -E copy_directory
          ${CMAKE_CURRENT_SOURCE_DIR}/${IMAGE_DIR}/${DATA_DIR}
          ${CMAKE_CURRENT_BINARY_DIR}/${IMAGE_DIR}/${DATA_DIR}
  )

add_latex_document(MyDoc.tex
  IMAGE_DIRS ${IMAGE_DIR}
  DEPENDS ${ALL_GRI_FILES}
  )
```

6.9 Why does make stop after each image conversion?

There is a bug in the ImageMagick convert version 6.1.8 that inappropriatly returns a failure condition even when the image convert was successful. The problem might also occur in other ImageMagick versions. Try updating your installation of ImageMagick.

6.10 How do I resolve \write 18 errors with pstricks or pdftricks?

A `\write18` command is LaTeX's obtuse syntax for running a command on your system. Commands in the pstricks and pdftricks packages may need to run commands on your system to, for example, convert graphics from one format to another.

Unfortunately, allowing LaTeX to run commands on your system can be considered a security issue. Some versions of LaTeX such as MikTeX disable the feature by default. Thus, in order to use packages that rely on `\write18`, you may have to enable the feature, typically with a command line option. For MikTeX, this command line option is `--enable-write18`.

You can instruct UseLATEX.cmake to pass any flag to LaTeX by adding it to the `LATEX_COMPILER_FLAGS` CMake variable. One way to do this is through the CMake GUI. Simply go to the advanced variables, find `LATEX_COMPILER_FLAGS`, and add `--enable-write18` (or equivalent flag) to the list of arguments.

You can also automatically add this flag by setting the flag in your CMakeLists.txt file. For example:

```
set(LATEX_COMPILER_FLAGS
  "-interaction=nonstopmode --enable-write18"
```

```
  CACHE STRING "Flags passed to latex."
  )
include(UseLATEX.cmake)
```

The disadvantage of this latter approach is the reduction of portability. This addition could cause a failure for any LaTeX implementation that does not support the `--enable-write18` flag (for which there are many).

6.11 Why is UseLATEX.cmake complaining about image file names?

If you have an image file with a filename that contains multiple periods, for example my.image.pdf, UseLATEX.cmake will issue a warning like the following.

```
Some LaTeX distributions have problems with image file names
with multiple extensions or spaces.  Consider changing
my.image.pdf to something like my-image.pdf.
```

This is because, just as the warning reports, some versions of LaTeX have problems with including image filenames with multiple extensions. For example, if you tried to include my.image.pdf with a command like

```
\includegraphics{my.image}
```

then some versions of LaTeX will respond that the image extension .image is not recognized or that the file my.image is not found because it fails to look for files with valid extensions.

Although it is inadvisable (per Section 3.2), you might try to get around the problem by specifying the extension like this.

```
\includegraphics{my.image.pdf}
```

This might work, or it might just make LaTeX complain that it does not recognize images with extension .image.pdf.

In the end, the best solution is to simply use filenames that will not trouble LaTeX. Even though some LaTeX distributions (like MacTeX) seem to handle this extension ambiguity correctly, others clearly do not. Thus, even if your LaTeX distribution handles these image filenames correctly, it is still a bad idea in case you need to change distributions or build on other computers. Your best course of action is to simply heed the warning and rename your files.

6.12 Why are there no FORCE_PS or FORCE_SAFEPDF options?

Because you should just use the FORCE_DVI option instead.

Both the ps and safepdf targets are built by first creating a .dvi file using the standard latex program. The .dvi file is then converted to .ps and subsequently to a .pdf file. Thus, you can just enable the FORCE_DVI option to force UseLATEX.cmake on this build path.

The force options are really disabling compile paths that do not work for your document. For example, pdflatex does not support all postscript packages, so that program can fail for some documents. The FORCE_DVI ensures that the document can only be built in ways that support the postscript features.

6.13 Why is my image file not being automatically converted?

UseLATEX.cmake has the ability to find image files and automatically convert them to a format LaTeX understands. Usually this conversion happens with the ImageMagick magick program.

Users occasionally report that image formats that should be supported because ImageMagick can convert them are ignored by UseLATEX.cmake. This can happen even when the IMAGE_DIRS option points to the directory containing the image files.

The problem here is that UseLATEX.cmake only considers files in IMAGE_DIRS directories that it identifies as a bona fide image. This prevents UseLATEX.cmake from picking up another type of file, such as a README text file, and erroneously trying to do image conversion on it.

UseLATEX.cmake checks for image files by looking for a known set of image extensions. This extension list is maintained separately from ImageMagick's extension list and is generally a small subset of all the potential formats ImageMagick supports. Consequently, it is possible for UseLATEX.cmake to ignore an image file that could be converted.

You can work around this problem by specifying images independently with the IMAGES option. UseLATEX.cmake will assume any image specified under the IMAGES option is in fact an image that can be converted with ImageMagick regardless of the extension. See Section 3.2 for more details.

If there is a file extension that you think should be added to the list of image extensions to check, send a note to the UseLATEX.cmake maintainers.

6.14 Why is the MANGLE_TARGET_NAMES option deprecated?

The original concept for UseLATEX.cmake was part of a build system for a single document. As such, add_latex_document created generically named targets (like dvi and pdf). This became problematic when UseLATEX.cmake was used in larger projects that built multiple targets. The multiple documents would each try to create their own dvi, pdf, etc. targets, and this would create CMake errors when they conflicted with each other.

To solve this problem, in 2006 the `MANGLE_TARGET_NAMES` was added to `add_latex_document`. When this option was given, `add_latex_document` would create "mangled" targets that are unique to the name of the document so that they would not conflict with each other.

This option solved the problem for projects building multiple documents, but a couple of undesirable elements were later discovered. The first was that LaTeX documents built with the `MANGLE_TARGET_NAMES` option were never built by default. To build the document, the user had to specifically request the target, which had an unwieldy name, to be built or to explicitly set up dependencies to those targets. The second and more serious issue was that if a project incorporated one or more sub-projects (not uncommon) and more than one of these projects used UseLATEX.cmake, you were likely to get conflicting targets again.

Consequently, in 2015 a change was made to `add_latex_document` to mangle all targets. The UseLATEX.cmake package establishes a single set of default target names (dvi, pdf, etc.), and `add_latex_document` sets up dependencies from these default targets to the mangled target names. Thus, when UseLATEX.cmake is used for a single document, the same simple targets work fine. When multiple documents are added, the default targets are automatically set up for all documents without conflicts. See Section 5.3 for more details on building multiple LaTeX documents in a project.

So, `MANGLE_TARGET_NAMES` is deprecated because it is redundant. All targets are mangled. The only difference is that `add_latex_document` establishes dependencies to the default target names. If these dependency targets are not desired, use the `EXCLUDE_FROM_DEFAULTS` option. (Once again, see Section 5.3 for more details.)

6.15 What is the point of the default LaTeX arguments?

The LaTeX commands (e.g. latex and pdflatex) were originally designed to be run interactively. The tex file is fed to the interpreter and verbose responses are generated. When an error is encountered, LaTeX stops and provides a prompt to type commands to resolve the problem. This interactive mode of building a LaTeX file is problematic when attempting to automate it in a batch or build system. Thus, the `LATEX_COMPILER_FLAGS` and `PDFLATEX_COMPILER_FLAGS`, which contain the command line flags passed to the LaTeX program, are initialized to modify the behavior to work better in a build system.

The first flag added is `-interaction=batchmode`. This flag does two major things. The first thing this flag does is hide most of the LaTeX output. A typical LaTeX build contains extremely verbose status messages that provide all sorts of useless information. Any important information (like a syntax error) is easily lost. Instead, you have to consult the .log file to see the full output. Because important warnings can be hidden along with the unimportant, UseLATEX.cmake performs several greps of the log file after the build to look for the most important warnings encountered with LaTeX.

The second thing the `-interaction=batchmode` flag does is to change the

behavior of LaTeX when an error occurs. Rather than enter an interactive prompt, the LaTeX program simply quits. This is how pretty much every build system expects a compiler to behave.

The second flag added is `-file-line-error`. For some odd reason the default behavior of LaTeX is to simply print out a message and leave it you to trace the location of the error. Instead, this flag instructs LaTeX to prepend the filename and line number to every error to simplify finding the error.

6.16 Why do the ps2pdf arguments have the # character in them?

When calling the ps2pdf program, it is typical to use several arguments that are passed to the underlying ghostscript system. These arguments often take the form of an option followed by an equal (=) character and then the value for that option. For example, arguments like `-dCompatibilityLevel=1.3`, `-dEmbedAllFonts=true`, and `-dColorImageFilter=/FlateEncode` are common. This is a standard convention for command line arguments in systems using Unix-like shells.

In truth, the ps2pdf program and its variants are actually shell scripts that provide a simplified interface for calling the gs ghostscript program. On Unix-like systems they are naturally enough implemented as shell scripts. However, the standard Windows port instead uses bat scripts, which are native to that system. Unfortunately, the interpreter for bat scripts treats the = character as special. Ultimately it will split the arguments on the = character, and that will lead to strange errors from ps2pdf. For example, on Windows the `-dCompatibilityLevel=1.3` argument will be split into the arguments `-dCompatibilityLevel` and `1.3`. ps2pdf will think `1.3` is referring to the input file name and give an obtuse error about the file not being found.

The workaround is that gs (and therefore all its derived scripts like ps2pdf) support using the # character in lieu of =. Thus, on Windows machines, UseLATEX.cmake defaults to an alternate set of arguments for ps2pdf that use # in them.

An issue you might encounter is that the # character is also frequently treated as special by script and build interpreter. It is most often used to define a comment. For this reason the # variant is only used on Windows where it is most likely to be needed. The build systems I have tried seem pretty resilient to using # in commands. If you have issues running ps2pdf with either character, you can attempt to resolve the problem by switching back and forth. If you do notice a problem, please let us know so that we can fix it for other users.

7 Acknowledgments

Thanks to all of the following contributors.

Matthias Bach Instructions for using LuaLaTeX.

Martin Baute Check for Windows version of convert being used instead of ImageMagick's version.

Izaak Beekman Help in fixing the order of arguments for `LATEX_SMALL_IMAGES` with Imagemagick 7.0.

Arnout Boelens Example of using gri in conjunction with LaTeX.

Mark de Wever Fixes for interactions between the makeglossaries and BibTeX commands.

Alin Elena Suggestions on removing dependence on makeglossaries command.

Myles English Support for the nomencl package.

Tomasz Grzegurzko Support for htlatex.

Øystein S. Haaland Support for making glossaries.

Sven Klomp Help with SyncTeX support.

Nikos Koukis Suggestions for default latex options.

Thimo Langbehn Support for pstricks with the `--enable-write18` option.

Antonio LaTorre Support for the multibib package.

Edwin van Leeuwen Fix for a bug when copying BibTeX files.

Dan Lipsa Support for the multind package.

Lukasz Lis Workaround for problem with ImageMagick dropping the BoundingBox of eps files by using the ps2pdf program instead.

Eric Noulard Support for any file extension on LaTeX input files.

Theodore Papadopoulo `DEPENDS` parameter for `add_latex_document` and help in identifying some dependency issues.

Jorge Gerardo Peña Pastor Support for SVG files.

Julien Schueller Check for existence of Imagemagick convert only when used.

David Tracey Support for using biber command with the `USE_BIBLATEX` option.

Raymod Wan `DEFAULT_SAFEPDF` option.

This work was primarily done at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

This document is released as technical report SAND 2008-2743P.

A Sample CMakeLists.txt

Following is a sample listing of CMakeLists.txt. In fact, it is the CMakeLists.txt that is used to build this document.

```
cmake_minimum_required(VERSION 2.8.4)

project(UseLATEX_DOC NONE)

include(UseLATEX.cmake)

# Note that normally CMakeLists.txt would not be considered an
# input to the document, but in this special case of documenting
# UseLATEX.cmake the contents of this file is actually included
# in the document.
add_latex_document(UseLATEX.tex
  INPUTS CMakeLists.txt
  )
```