How LAPACK library enables Microsoft Visual Studio support with CMake and LAPACKE

Julie Langou[1], Bill Hoffman[2], Brad King[2]

1. *University of Tennessee Knoxville, USA*
2. *Kitware Inc., USA*

*This article is an extended version of the blog **"Fortran for C/C++ developers made easier with CMake"** posted on the Kitware blog at http://www.kitware.com/blog/home/post/231. This extended version is intended especially for Fortran library developers and includes many useful examples available for download.*

LAPACK is written in Fortran and offers a native Fortran interface simple enough to inter-operate with other languages. Recently, INTEL Inc and the LAPACK team developed a C-language API for LAPACK called *LAPACKE* [1] to guarantee LAPACK interoperability with C. However, many developers are either unfamiliar with the non-trivial process of mixing C and Fortran together or develop with Microsoft Visual Studio which offers no builtin Fortran support. Starting with LAPACK 3.3.0, the LAPACK team decided to join forces with Kitware Inc. to develop a new build system for LAPACK using CMake [2]. This fruitful collaboration gave LAPACK the ability to easily compile and run on the popular Linux, Windows, and Mac OS/X platforms in addition to most UNIX-like platforms. In addition, LAPACK now offers Windows users the ability to code in C using Microsoft Visual Studio and link to LAPACK Fortran libraries without the need of a vendor-supplied Fortran compiler add-on.

CMake Fortran Features

The CMake build system contains many powerful features that make combining Fortran with C and/or C++ easy and cross-platform.

- Automatic detection of **Fortran runtime libraries** required to link to a Fortran library into a C/C++ application.
- Automatic detection of **Fortran name mangling** required to call Fortran routines from C/C++.
- Automatic detection of **Fortran 90 module dependencies** required to compile sources in the **correct order**. (This is not needed by LAPACK.)
- Windows: Automatic conversion of Fortran libraries compiled with **MinGW gfortran** to **.lib files** compatible with Microsoft Visual Studio.
- Windows: Seamless **execution of MinGW gfortran from within Visual Studio** C/C++ projects to build Fortran libraries.

The following describes how to apply these features to any CMake project requiring mixed C/C++ and Fortran code.

[1] LAPACKE: http://www.netlib.org/lapack/#_standard_c_language_apis_for_lapack
LAPACKE User guide: http://www.netlib.org/lapack/lapacke.html
[2] CMake: http://www.cmake.org/

Introduction by Example

In order to demonstrate the above features we will work through a series of C examples calling Fortran routines from BLAS and LAPACK (See [3]).

We will use **two C programs**:

- **timer_dgemm.c:** Program to time a Dense Matrix Multiplications in double precision (BLAS routine DGEMM).
- **timer_dgesv.c:** Program to time a Linear Solve in double precision (LAPACK routine DGESV) and checks that the solution is correct.

and two Fortran libraries:

- refblas: the Complete Reference BLAS.
- dgesv: a LAPACK subset that just contains the DGESV routine and its dependencies. This library depends on refblas.

To correctly call Fortran from C source code, the following issues must be addressed in the two C programs:

- In **Fortran**, all routine arguments are **passed by address** and not by value.
- In C global functions can be called directly by the name used in the C code (routine "foo" is called by the name "foo"). However, when calling **Fortran** from C the external name is modified by a vendor-specific mapping. Some Fortran compilers append an **underscore** (routine "foo" is symbol "foo_") , some **two underscores**, some leave it **unchanged** and some convert the name to **uppercase**. This is known as **Fortran Name Mangling**. Since the C compiler has no knowledge of the Fortran compiler's naming scheme, the functions must be correctly mangled in the C code.
- **Fortran** stores arrays in **row order** while **C and C++** store arrays in **column order**.

For more complete information on how to call Fortran from C see [4].

Examples 1 to 4 demonstrate the approach to mixing C and Fortran in the CMake build system:

- **Example 1:** Linking Fortran libraries from C/C++ code.
- **Example 2:** Resolving Fortran mangling
- **Example 3:** Introducing Microsoft Visual Studio
- **Example 4:** Complete example based on LAPACKE (linking C and Fortran libraries directly under Visual Studio)

The complete set of examples is available for download at http://icl.cs.utk.edu/lapack-for-windows/Example_C_Fortran_CMake.tgz

---

[3] NETLIB LAPACK Library: http://www.netlib.org/netlib
LAPACK technical articles: http://www.netlib.org/netlib/lawns
LAPACK for windows http://icl.cs.utk.edu/lapack-for-windows/lapack

[4] Mixing Fortran and C: http://www.math.utah.edu/software/c-with-fortran.html

Automatic detection of Fortran runtime libraries to link

The first challenge to mix Fortran and C/C++ is to link all the language runtime libraries. The Fortran, C, and C++ compiler front-ends automatically pass their corresponding language runtime libraries when invoking the linker. When linking to a Fortran library from a C/C++ project the C/C++ compiler front end will not automatically add the Fortran language runtime libraries. Instead one must pass them explicitly when linking. Typically one must run the Fortran compiler in a verbose mode, look for the –L and –l flags, and manually add them to the C/C++ application link line. This tedious process must be repeated for each platform supported!

CMake handles this automatically for projects that mix C/C++ with Fortran. It detects the language runtime library link options used by the compiler front-end for each language. Whenever a library written in one language (say Fortran) is linked by a binary written in another language (say C++) CMake will explicitly pass the necessary link options for the extra languages.

CMake module name: none

This feature is builtin to CMake and automatically applied to projects that use both Fortran and C or C++.

Example 1: Linking Fortran from C
Do not pay too much attention to the C code yet, just keep in mind that it is calling a Fortran BLAS and/or LAPACK routine. The further we go in those examples, the simpler the C code will get.

Example 1: CMakeLists.txt **Linking Fortran from C**

```
CMake_minimum_required(VERSION 2.8.8)
# Fortran and C are specified so the Fortran runtime library will be
# automatically detected and used for linking
project (TIMER_Example1 Fortran C)

# Set Mangling by hand
# Assume _name mangling see C code for how ADD_ will be used in
# Fortran mangling
add_definitions (-DADD_)
# Build the refblas library
add_subdirectory (refblas)

# Timer for Matrix Multiplication
# Calling Fortran DGEMM routine from C
add_executable(timer_dgemm timer_dgemm.c)
target_link_libraries(timer_dgemm refblas)
# Create the dgesv library
```

```
add_subdirectory (dgesv)

# Timer for Linear Solve
# Calling Fortran DGESV routine from C also requires linking
# the BLAS routines.
add_executable(timer_dgesv timer_dgesv.c)
target_link_libraries(timer_dgesv dgesv refblas)
```

Example 1: timer_dgemm.c **Calling Fortran from C with manual mangling**

```
[…]

# Mangling is done "by hand"
# User needs to know which define to set at compile time.
#if defined(ADD_)
   #define dgemm dgemm_
#elif defined(UPPER)
   #define dgemm DGEMM
#elif defined(NOCHANGE)
   #define dgemm dgemm
#endif

[…]

# Declaration of Fortran DGESV routine
extern void dgemm (char *transa, char *transb, int *m, int *n, int *k, double
*alpha, double *a, int *lda, double *b, int *ldb, double *beta, double *c,
int *ldc );

[…]

# Call to Fortran DGEMM routine
dgemm (&char01, &char01, &m, &n, &k, &dble01, A, &m, B, &k, &dble01, C, &m);

[…]
```

Download Example 1:
http://icl.cs.utk.edu/lapack-for-windows/Example_C_Fortran_CMake/Example1.tgz

Automatic Detection of Fortran Name Mangling

The second challenge in mixing Fortran and C/C++ is to **know how the Fortran compiler transforms symbol names**. Different compilers append or prepend "_", or use upper or lower case for the function names. For more information on name mangling in Fortran, see [5]. CMake

[5] Fortran MANGLING:
http://en.wikipedia.org/wiki/Name_mangling#Name_mangling_in_Fortran

contains a module that can be used to determine the mangling scheme used by the Fortran compiler. It can be used to create C/C++ header files that contain preprocessor definitions that map C/C++ symbol names to their Fortran-mangled counterparts.

CMake module name: **`FortranCInterface`**

CMake function name: **`FortranCInterface_HEADER`**

CMake help: `cmake --help-module FortranCInterface`

Syntax: FortranCInterface_HEADER from include(FortranCInterface)	

`FortranCInterface_HEADER(`
`<file>`
	`# name of the header file to be created`

`[MACRO_NAMESPACE <macro-ns>]`	`# The` **`MACRO_NAMESPACE`** `option replaces the default "FortranCInterface_" prefix with a given namespace "<macro-ns>".`
`[SYMBOL_NAMESPACE <ns>]`	`# The` **`SYMBOL_NAMESPACE`** `option prefixes all preprocessor definitions generated by the SYMBOLS option with a given namespace "<ns>".`

`[SYMBOLS [<module>:]<function> …])`	`# The` **`SYMBOLS`** `option lists symbols to mangle automatically with C preprocessor definitions:`
	`<function>  #define <ns><function> ...`
	`<module>:<function> `
	`#define <ns><module>_<function> ...`
	
	`# If the mangling for some symbol is not known then no preprocessor definition is created, and a warning is displayed.`

Example 2: Resolving Fortran Mangling

In Example 1 we assumed the mangling for the routine to be “add underscore” but this is not necessarily the case. Let’s use the CMake FortranCInterface module described above to create the netlib.h header file that will take care of the mangling automatically. It will generate a header mapping dgemm, dgesv, dcopy and dnrm1 to the correct routine name. The header file is by default generated in the current binary directory so we must also tell CMake to add that as an include directory.

Example 2: CMakeLists.txt **to Detect Fortran Name Mangling**

```
cmake_minimum_required(VERSION 2.8.8)
project (TIMER_Example2 Fortran C) # enable Fortran and C

# Create a header file netlib.h with the correct mangling
# for the Fortran routines called in my C programs.
include(FortranCInterface)
FortranCInterface_HEADER(netlib.h
     MACRO_NAMESPACE "NETLIB_"
     SYMBOLS dgemm dgesv dcopy dnrm2)


# To find the newly generated header file netlib.h at compile time
include_directories (${PROJECT_BINARY_DIR})

# Build the refblas library
add_subdirectory (refblas)

# Timer for Matrix Multiplication
# Calling Fortran DGEMM routine from C
add_executable(timer_dgemm timer_dgemm.c)
target_link_libraries(timer_dgemm refblas)

# Create the dgesv library
add_subdirectory(dgesv)


# Timer for Linear Solve
# Calling Fortran DGESV routine from C requires also to
# have the BLAS routines.
add_executable(timer_dgesv timer_dgesv.c)
target_link_libraries(timer_dgesv dgesv refblas)
```

Example 2: sample netlib.h **Header generated by CMake FortranCInterface module**

```
#ifndef NETLIB_HEADER_INCLUDED
#define NETLIB_HEADER_INCLUDED

/* Mangling for Fortran global symbols without underscores. */
#define NETLIB_GLOBAL(name,NAME) name##_

/* Mangling for Fortran global symbols with underscores. */
#define NETLIB_GLOBAL_(name,NAME) name##_

/* Mangling for Fortran module symbols without underscores. */
#define NETLIB_MODULE(mod_name,name, mod_NAME,NAME) \
  __##mod_name##_MOD_##name

/* Mangling for Fortran module symbols with underscores. */
#define NETLIB_MODULE_(mod_name,name, mod_NAME,NAME) \
  __##mod_name##_MOD_##name
```

```
/*------------------------------------------------------------------------*/
/* Mangle some symbols automatically.                                     */
#define dgemm NETLIB_GLOBAL(dgemm, DGEMM)
#define dgesv NETLIB_GLOBAL(dgesv, DGESV)
#define dcopy NETLIB_GLOBAL(dcopy, DCOPY)
#define dnrm2 NETLIB_GLOBAL(dnrm2, DNRM2)

#endif
```

Example 2: timer_dgemm.c **Calling Fortran from C with automatic mangling**

```
[…]

/* Use CMake generated header file with auto-detected Mangling */
#include <netlib.h>

[…]

/* Declaration of Fortran DGESV routine */
extern void sdgemm (char *transa, char *transb, int *m, int *n, int *k,
double *alpha, double *a, int *lda, double *b, int *ldb, double *beta, double
*c, int *ldc );

[…]

/* Call to Fortran DGEMM routine */
dgemm (&char01, &char01, &m, &n, &k, &dble01, A, &m, B, &k, &dble01, C, &m);

[…]
```

Download Example2:
http://icl.cs.utk.edu/lapack-for-windows/Example_C_Fortran_CMake/Example2.tgz

Using MinGW6 gfortran with Visual Studio

The third challenge to mix Fortran and C/C++ is to make it work on Windows under **Microsoft Visual Studio** without the need of installing a commercial Fortran compiler. Fortunately MinGW provides a Windows port of **gfortran**, the free GNU Fortran compiler. However, unlike commercial Fortran compilers such as Intel, PGI, Silverforst, etc. MinGW gfortran does not provide direct integration into Visual Studio.

[6] MINGW: http://www.mingw.org/
http://mingw-w64.sourceforge.net/ for both x64 & x86 Windows

CMake offers two features to make using MinGW gfortran with Microsoft Visual Studio easy. First, it can create a MS-format import library (.lib) for a shared library (.dll) compiled with MinGW gfortran. Second, it comes with a new CMake module called CMakeAddFortranSubdirectory. The module brings a subdirectory containing a Fortran-only subproject into an otherwise C/C++-only project. The module will **automatically enable Fortran support under Visual Studio**. It can use native Fortran language support when available (e.g. a VS plugin from Intel or PGI) or generate rules to compile the subproject by invoking MinGW gfortran inside VS. Of course, this will work just fine if you do have a commercial Fortran compiler.

Creating Visual Studio .lib files from MinGW gfortran

When creating a shared library (.dll) the MinGW toolchain produces a GNU-format import library (.dll.a) needed to link to the shared library. Microsoft (MS) tools do not recognize this format. In order to use MS tools and link to a shared library created by MinGW tools one must generate a MS-format import library (.lib) using the MS lib.exe tool installed with Visual Studio.

When building with MinGW tools on a machine that also has Visual Studio installed CMake 2.8.7 and higher offers a “CMAKE_GNUtoMS” option. When enabled CMake automatically generates additional build rules to run the MS lib.exe tool and create a MS-format import library (.lib) for each shared library (.dll) created by a project in addition to its normal GNU-format import library (.dll.a). One may enable the option by setting CMAKE_GNUtoMS inside the project CMake code or by passing it on the CMake command line when generating the build tree:

CMake FLAGS
`cmake … -DCMake_GNUtoMS=ON …`

Note that the feature currently works only for SHARED libraries because they reference their dependence on the GNU Fortran runtime libraries internally. STATIC libraries are more challenging because they requires explicit linking to the runtime libraries whose format would also need conversion. A project may specify that a library be shared by using the SHARED option of the add_library command:

Inside the project CMake code
`add_library(… SHARED …)`

If a project does not specify the library type when calling add_library one may tell CMake to build shared libraies on the command line:

CMake FLAGS

```
cmake … -DBUILD_SHARED_LIBS=ON …
```

Side note: DLLs and Intel Fortran

If a project needs to work with the Intel Fortran compiler as well then one needs to make sure symbols are exported from each shared library. MinGW gfortran can export all the symbols from a shared library automatically but Intel Fortran requires explicit specification of symbols to export. One may manually create a DLL module definition (.def) file that lists the symbols to export and add it to the add_library call as a source file in CMake. Alternatively, one may mark up the Fortran code with special "DEC$" comments recognized by the Intel Fortran compiler:

```
hello.f

!DEC$ ATTRIBUTES DLLEXPORT :: HELLO
        SUBROUTINE HELLO
        PRINT *, 'Hello'
```

The CMakeAddFortranSubdirectory Module

CMake 2.8.8 offers a new experimental module called CMakeAddFortranSubdirectory. It provides a "cmake_add_fortran_subdirectory" function to make a Fortran subproject appear as if it were included in a C/C++ project using a normal add_subdirectory command. **The subproject may contain only Fortran code and may not contain any C/C++ code**.

The function first checks if a Fortran compiler is available along with the C/C++ compiler toolchain and if so simply calls add_subdirectory. This is the normal case on most platforms. The "magic" happens when **building for Visual Studio without an integrated Fortran compiler on a machine with MinGW gfortran**. The cmake_add_fortran_subdirectory function searches for MinGW gfortran installation and uses the CMake ExternalProject module to add a custom target that builds the subdirectory with the MinGW tools. It enables the BUILD_SHARED_LIBS and CMAKE_GNUtoMS options in the external project to build the Fortran code into shared libraries with MS-format import libraries.

In the latter case the function creates CMake "imported" targets to make the Fortran libraries available to the main project as if they had been built by a direct add_subdirectory call. This requires one to pass additional arguments to the function to tell it what libraries will be built by the subproject.

CMake module name: `CMakeAddFortranSubdirectory`

CMake function name: `cmake_add_fortran_subdirectory`

CMake help: `cmake --help-module` **`CMakeAddFortranSubdirectory`**

Syntax: cmake_add_fortran_subdirectory from include(CMakeAddFortranSubdirectory)

```
cmake_add_fortran_subdirectory (
   <subdir>                            # name of subdirectory
   PROJECT <project_name>              # project name in subdir
   ARCHIVE_DIR <dir>                   # dir where project places .lib files
   RUNTIME_DIR <dir>                   # dir where project places .dll files
   LIBRARIES <lib>...                  # names of library targets to import
   LINK_LIBRARIES                      # link dependencies for LIBRARIES
   [LINK_LIBS <lib> <dep>...]...
   CMake_COMMAND_LINE ...              # extra command line to pass to CMake
   NO_EXTERNAL_INSTALL                 # skip installation of external project
         )
```

Note: The relative paths in ARCHIVE_DIR and RUNTIME_DIR are interpreted with respect to the build directory corresponding to the source directory in which the function is invoked.

Example 3: Introducing Microsoft Visual Studio

Here is our second example modified to use the CMakeAddFortranSubdirectory feature. Only the CMake configuration has been changed; the C and Fortran source files are the same as in the previous example. The main difference is that now we work under **Windows** with **Microsoft Visual Studio**. On our machine, no Fortran compiler is integrated with VS (i.e no Intel Fortran Compiler for Windows installed) but we **have MinGW** installed on our machine. The MinGW installation must include gfortran and gcc in order to determine the Fortran mangling scheme used.

- We move our Fortran subfolders to the new fortran folder that will be an actual CMake Fortran project. We are now able to use the CMakeAddFortranSubdirectory module.
- Our Fortran Project will still need C as we want still to use the CMake **FortranCInterface** module to create our netlib.h header. The header will now be generated in the binary dir of the dgesv project.
- Because now we are going to work under Visual Studio and with dll's, we need to add extra commands to make sure the dll's generated will be available in our binary directory.

Example 3: CMakeLists.txt **Using MinGW gfortran from Visual Studio**

```
CMake_minimum_required(VERSION 2.8.8)
# Fortran is not mentioned as we accept no Native Fortran Support
project (TIMER_Example3)

# Organize output so that all generated lib go to the same lib directory
# and all dll and executable go to the same bin directory
set(CMake_RUNTIME_OUTPUT_DIRECTORY "${PROJECT_BINARY_DIR}/bin")
set(CMake_ARCHIVE_OUTPUT_DIRECTORY "${PROJECT_BINARY_DIR}/lib")
set(CMake_LIBRARY_OUTPUT_DIRECTORY "${PROJECT_BINARY_DIR}/lib")

# Get the module CMakeAddFortranSubdirectory
include(CMakeAddFortranSubdirectory)
```

```
# Add the fortran subdirectory as a fortran project
# the subdir is fortran, the project is DGESV
CMake_add_fortran_subdirectory(fortran
PROJECT DGESV
ARCHIVE_DIR ${CMake_ARCHIVE_OUTPUT_DIRECTORY}
RUNTIME_DIR ${CMake_RUNTIME_OUTPUT_DIRECTORY}
LIBRARIES refblas dgesv # target libraries created
LINK_LIBRARIES  # link interface libraries
LINK_LIBS dgesv refblas   # dgesv needs refblas to link
CMake_COMMAND_LINE
-DCMake_ARCHIVE_OUTPUT_DIRECTORY=${CMake_ARCHIVE_OUTPUT_DIRECTORY}
-DCMake_RUNTIME_OUTPUT_DIRECTORY=${CMake_RUNTIME_OUTPUT_DIRECTORY}
NO_EXTERNAL_INSTALL
)

# To find the newly generated header file
include_directories (${PROJECT_BINARY_DIR}/fortran)

# The autofortran library linking does not work unless Fortran is enabled.
include(CheckLanguage)
check_language(Fortran)
if(CMAKE_Fortran_COMPILER)
   enable_language(Fortran)
else()
   message(STATUS "No native Fortran support ")
endif()

# Timer for Matrix Multiplication Calling Fortran DGEMM routine from C
add_executable(timer_dgemm timer_dgemm.c)
target_link_libraries(timer_dgemm refblas)

# Timer for Linear Solve Calling Fortran DGESV routine from C
add_executable(timer_dgesv timer_dgesv.c)
target_link_libraries(timer_dgesv dgesv)

# Add extra command to copy dll's next to the binaries
# so that the PATH does not have to be altered to run
# the executables
IF(WIN32)
   ADD_CUSTOM_COMMAND(
       TARGET timer_dgemm
       POST_BUILD
       COMMAND ${CMake_COMMAND} -E copy
       ${PROJECT_BINARY_DIR}/bin/librefblas.dll
       ${PROJECT_BINARY_DIR}/bin/${CMake_CFG_INTDIR}
       )

  ADD_CUSTOM_COMMAND(
       TARGET timer_dgesv
       POST_BUILD
       COMMAND ${CMake_COMMAND} -E copy
       ${PROJECT_BINARY_DIR}/bin/libdgesv.dll
       ${PROJECT_BINARY_DIR}/bin/${CMake_CFG_INTDIR}
       )
ENDIF(WIN32)
```

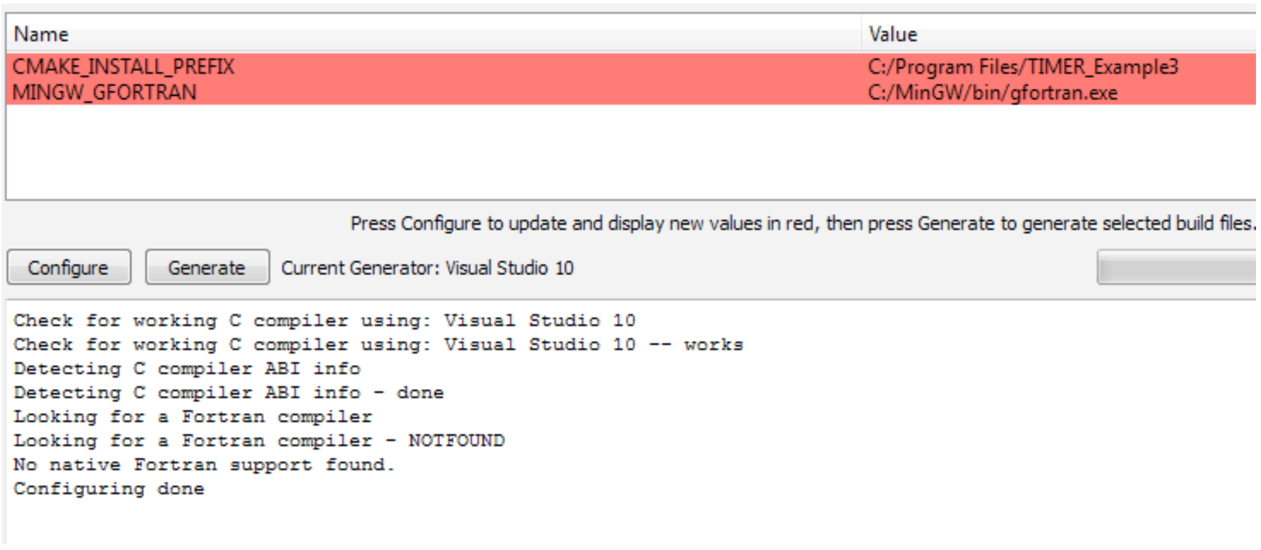
Example 3: fortran/CMakeLists.txt **DGESV Fortran Project**

```
CMake_minimum_required(VERSION 2.8.7)
project(DGESV Fortran C)

# Create a header file netlib.h for the routines called in my C programs
include(FortranCInterface)
FortranCInterface_HEADER(netlib.h
                         SYMBOLS dgemm dgesv dcopy dnrm2)

# Creating a library with the Fortran routine DGESV and its dependencies
add_subdirectory(refblas)
add_subdirectory(dgesv)
```

Example 3: Screenshot of CMake output under Windows



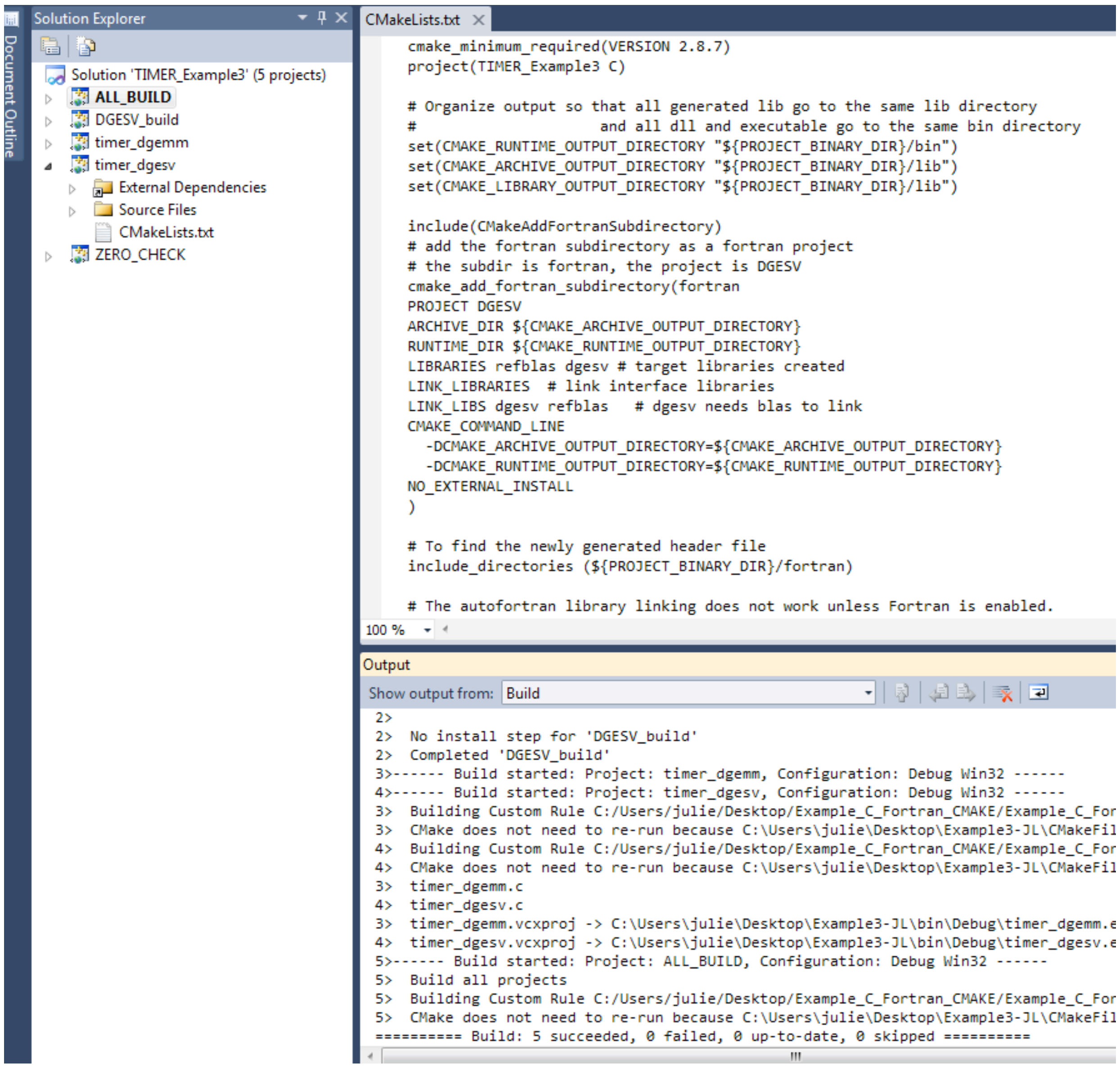
Name	Value
CMAKE_INSTALL_PREFIX	C:/Program Files/TIMER_Example3
MINGW_GFORTRAN	C:/MinGW/bin/gfortran.exe

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 10

```
Check for working C compiler using: Visual Studio 10
Check for working C compiler using: Visual Studio 10 -- works
Detecting C compiler ABI info
Detecting C compiler ABI info - done
Looking for a Fortran compiler
Looking for a Fortran compiler - NOTFOUND
No native Fortran support found.
Configuring done
```

Example 3: Screenshot of Visual Studio Build



```
cmake_minimum_required(VERSION 2.8.7)
project(TIMER_Example3 C)

# Organize output so that all generated lib go to the same lib directory
#                      and all dll and executable go to the same bin directory
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY "${PROJECT_BINARY_DIR}/bin")
set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY "${PROJECT_BINARY_DIR}/lib")
set(CMAKE_LIBRARY_OUTPUT_DIRECTORY "${PROJECT_BINARY_DIR}/lib")

include(CMakeAddFortranSubdirectory)
# add the fortran subdirectory as a fortran project
# the subdir is fortran, the project is DGESV
cmake_add_fortran_subdirectory(fortran
PROJECT DGESV
ARCHIVE_DIR ${CMAKE_ARCHIVE_OUTPUT_DIRECTORY}
RUNTIME_DIR ${CMAKE_RUNTIME_OUTPUT_DIRECTORY}
LIBRARIES refblas dgesv # target libraries created
LINK_LIBRARIES  # link interface libraries
LINK_LIBS dgesv refblas   # dgesv needs blas to link
CMAKE_COMMAND_LINE
  -DCMAKE_ARCHIVE_OUTPUT_DIRECTORY=${CMAKE_ARCHIVE_OUTPUT_DIRECTORY}
  -DCMAKE_RUNTIME_OUTPUT_DIRECTORY=${CMAKE_RUNTIME_OUTPUT_DIRECTORY}
NO_EXTERNAL_INSTALL
)

# To find the newly generated header file
include_directories (${PROJECT_BINARY_DIR}/fortran)

# The autofortran library linking does not work unless Fortran is enabled.
```

```
2>
2>  No install step for 'DGESV_build'
2>  Completed 'DGESV_build'
3>------ Build started: Project: timer_dgemm, Configuration: Debug Win32 ------
4>------ Build started: Project: timer_dgesv, Configuration: Debug Win32 ------
3>  Building Custom Rule C:/Users/julie/Desktop/Example_C_Fortran_CMAKE/Example_C_For
3>  CMake does not need to re-run because C:\Users\julie\Desktop\Example3-JL\CMakeFil
4>  Building Custom Rule C:/Users/julie/Desktop/Example_C_Fortran_CMAKE/Example_C_For
4>  CMake does not need to re-run because C:\Users\julie\Desktop\Example3-JL\CMakeFil
3>  timer_dgemm.c
4>  timer_dgesv.c
3>  timer_dgemm.vcxproj -> C:\Users\julie\Desktop\Example3-JL\bin\Debug\timer_dgemm.e
4>  timer_dgesv.vcxproj -> C:\Users\julie\Desktop\Example3-JL\bin\Debug\timer_dgesv.e
5>------ Build started: Project: ALL_BUILD, Configuration: Debug Win32 ------
5>  Build all projects
5>  Building Custom Rule C:/Users/julie/Desktop/Example_C_Fortran_CMAKE/Example_C_For
5>  CMake does not need to re-run because C:\Users\julie\Desktop\Example3-JL\CMakeFil
========== Build: 5 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Download Example3:
http://icl.cs.utk.edu/lapack-for-windows/Example_C_Fortran_CMake/Example3.tgz

Plugging a C Interface with the FORTRAN Library

Now that your project of mixing Fortran and C with CMake is ready, the only burden that remains for the user is the coding part. Some of you will decide that a C Interface could to be useful to your project to handle for example column-major and row-major format, complex data types , input arguments passed by value, .. In this section, we will show that plugging your C Interface in your project is straight-forward now that you understood the powerful CMake features shown in the previous examples.

Example 4: Introducing LAPACKE

Since CMake allow us to link Fortran directly from Microsoft Visual Studio, we are going to add a C Interface layer, namely LAPACKE, to make calls to LAPACK easier from C from a user point-of view. Please refer to the LAPACKE User guide for further help [i] . Example 4 demonstrates how LAPACKE is included into the LAPACK CMake build. As LAPACKE subset is build directly from Microsoft Visual Studio, we can choose to build static or dynamic library of LAPACKE (default being static). We are going to use the official LAPACKE DGESV examples written by INTEL corporation.

This example represents a **subset of LAPACK and LAPACKE** as only DGESV and its dependencies are used. To get complete prebuilt of Reference BLAS, LAPACK, LAPACKE libraries, please refer to http://icl.cs.utk.edu/lapack-for-windows/lapack/index.html#libraries.

Example 4: CMakeLists.txt **Using MinGW gfortran and including LAPACKE from Visual Studio**

```
CMake_minimum_required(VERSION 2.8.8)
# Fortran is not mentioned as we accept no Native Fortran Support
project (LAPACKE_Example C)

# Organize output so that all generated lib go to the same lib directory
# and all dll and executable go to the same bin directory
set(CMake_RUNTIME_OUTPUT_DIRECTORY "${PROJECT_BINARY_DIR}/bin")
set(CMake_ARCHIVE_OUTPUT_DIRECTORY "${PROJECT_BINARY_DIR}/lib")
set(CMake_LIBRARY_OUTPUT_DIRECTORY "${PROJECT_BINARY_DIR}/lib")

# Looking for an optimized BLAS library already installed on the machine

# If none available, we will use the Reference BLAS

# result will be stored in the BLAS_LIBRARIES variable
include(${PROJECT_SOURCE_DIR}/setBLAS.CMake)
setBLAS()

# Get the module CMakeAddFortranSubdirectory
include(CMakeAddFortranSubdirectory)

# Add the fortran subdirectory as a fortran project
# the subdir is fortran, the project is DGESV
CMake_add_fortran_subdirectory(fortran
PROJECT DGESV
ARCHIVE_DIR ${CMake_ARCHIVE_OUTPUT_DIRECTORY}
RUNTIME_DIR ${CMake_RUNTIME_OUTPUT_DIRECTORY}
LIBRARIES refblas dgesv # target libraries created
LINK_LIBRARIES  # link interface libraries
LINK_LIBS dgesv refblas   # dgesv needs refblas to link
CMake_COMMAND_LINE
-DCMake_ARCHIVE_OUTPUT_DIRECTORY=${CMake_ARCHIVE_OUTPUT_DIRECTORY}
-DCMake_RUNTIME_OUTPUT_DIRECTORY=${CMake_RUNTIME_OUTPUT_DIRECTORY}
NO_EXTERNAL_INSTALL
)
# To find the newly generated header file
include_directories (${PROJECT_BINARY_DIR}/include)
# All C subdirectory to build LAPACKE
add_subdirectory(c)

# The autofortran library linking does not work unless Fortran is enabled.
include(CheckLanguage)
check_language(Fortran)
if(CMake_Fortran_COMPILER)
   enable_language(Fortran)
else()
   message(STATUS "No native Fortran support ")
endif()
# LAPACKE Examples: Calling LAPACKE DGESV routine from C
add_executable(example_DGESV_rowmajor example_DGESV_rowmajor.c)
target_link_libraries(example_DGESV_rowmajor lapacke dgesv ${BLAS_LIBRARIES}
)
add_executable(example_DGESV_colmajor example_DGESV_colmajor.c)
target_link_libraries(example_DGESV_colmajor lapacke dgesv ${BLAS_LIBRARIES}
)
# Add extra command to copy dll's next to the binaries
IF(WIN32)
   ADD_CUSTOM_COMMAND(
        TARGET example_DGESV_rowmajor
        POST_BUILD
        COMMAND ${CMake_COMMAND} -E copy
        ${PROJECT_BINARY_DIR}/bin/librefblas.dll
        ${PROJECT_BINARY_DIR}/bin/${CMake_CFG_INTDIR}
        )

  ADD_CUSTOM_COMMAND(
        TARGET example_DGESV_rowmajor
        POST_BUILD
        COMMAND ${CMake_COMMAND} -E copy
        ${PROJECT_BINARY_DIR}/bin/libdgesv.dll
        ${PROJECT_BINARY_DIR}/bin/${CMake_CFG_INTDIR}
        )
ENDIF(WIN32)
```

LAPACKE requires the creation of lapacke_mangling.h. We are just going to directly use CMAKE capabilities to create the mangling macro LAPACK_GLOBAL and save it in the lapacke_mangling.h file. The lapacke.h include file will include the lapacke_mangling.h file and define the new routine name with calls to the LAPACK_GLOBAL macro.

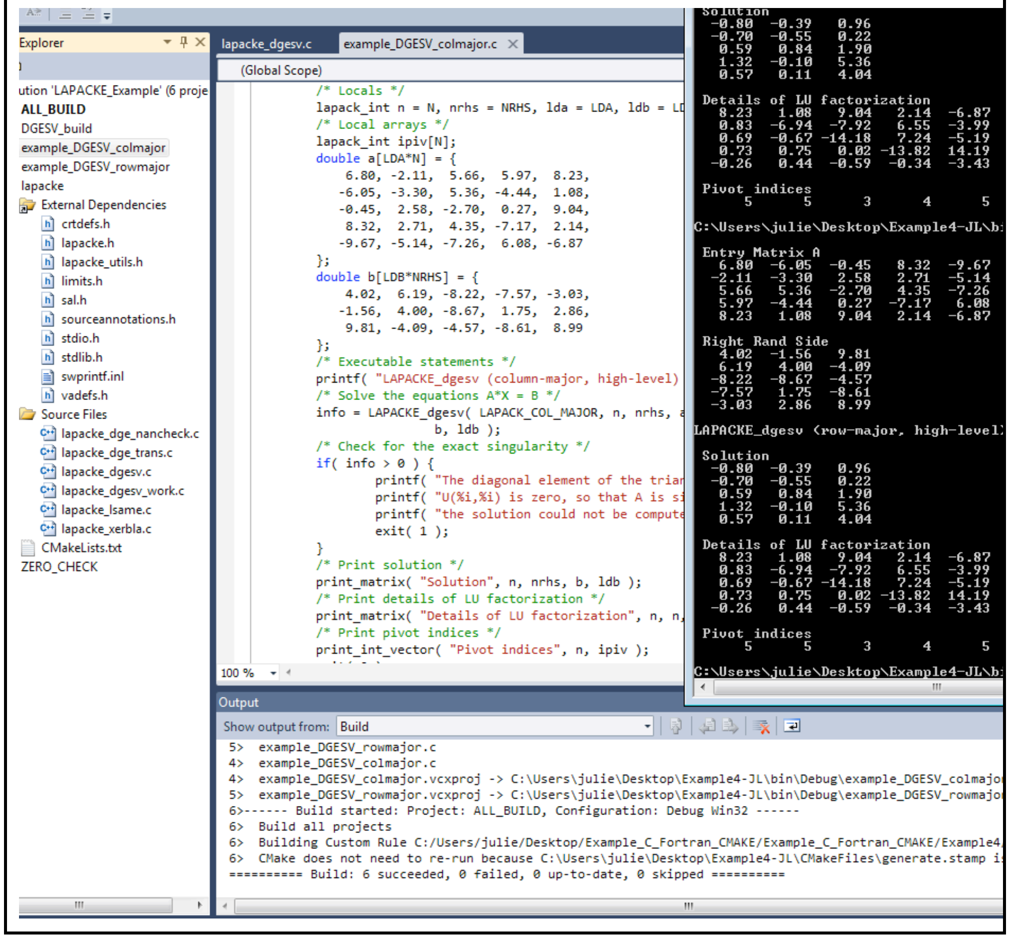
Example 4: fortran/CMakeLists.txt **DGESV Fortran Project**

```
CMake_minimum_required(VERSION 2.8.8)
project(DGESV Fortran C)


# Create a header file lapacke_mangling.h for the routines called in my C
# programs
include(FortranCInterface)
FortranCInterface_HEADER(../include/lapacke_mangling.h
                         MACRO_NAMESPACE "LAPACK_"
        -                 SYMBOL_NAMESPACE "LAPACK_")


# Creating a library with the Fortran routine DGESV and its dependencies
add_subdirectory(refblas)
add_subdirectory(dgesv)
```

Example 4: Screenshot of Visual Studio Build and executable output



```
    /* Locals */
    lapack_int n = N, nrhs = NRHS, lda = LDA, ldb = LD
    /* Local arrays */
    lapack_int ipiv[N];
    double a[LDA*N] = {
         6.80, -2.11,  5.66,  5.97,  8.23,
        -6.05, -3.30,  5.36, -4.44,  1.08,
        -0.45,  2.58, -2.70,  0.27,  9.04,
         8.32,  2.71,  4.35, -7.17,  2.14,
        -9.67, -5.14, -7.26,  6.08, -6.87
    };
    double b[LDB*NRHS] = {
         4.02,  6.19, -8.22, -7.57, -3.03,
        -1.56,  4.00, -8.67,  1.75,  2.86,
         9.81, -4.09, -4.57, -8.61,  8.99
    };
    /* Executable statements */
    printf( "LAPACKE_dgesv (column-major, high-level)
    /* Solve the equations A*X = B */
    info = LAPACKE_dgesv( LAPACK_COL_MAJOR, n, nrhs, a
                    b, ldb );
    /* Check for the exact singularity */
    if( info > 0 ) {
            printf( "The diagonal element of the triar
            printf( "U(%i,%i) is zero, so that A is si
            printf( "the solution could not be compute
            exit( 1 );
    }
    /* Print solution */
    print_matrix( "Solution", n, nrhs, b, ldb );
    /* Print details of LU factorization */
    print_matrix( "Details of LU factorization", n, n,
    /* Print pivot indices */
    print_int_vector( "Pivot indices", n, ipiv );
```

```
 Solution
 -0.80  -0.39   0.96
 -0.70  -0.55   0.22
  0.59   0.84   1.90
  1.32  -0.10   5.36
  0.57   0.11   4.04

 Details of LU factorization
  8.23   1.08   9.04   2.14  -6.87
  0.83  -6.94  -7.92   6.55  -3.99
  0.69  -0.67 -14.18   7.24  -5.19
  0.73   0.75   0.02 -13.82  14.19
 -0.26   0.44  -0.59  -0.34  -3.43

 Pivot indices
      5      5      3      4      5

C:\Users\julie\Desktop\Example4-JL\bi

 Entry Matrix A
  6.80  -6.05  -0.45   8.32  -9.67
 -2.11  -3.30   2.58   2.71  -5.14
  5.66   5.36  -2.70   4.35  -7.26
  5.97  -4.44   0.27  -7.17   6.08
  8.23   1.08   9.04   2.14  -6.87

 Right Rand Side
  4.02  -1.56   9.81
  6.19   4.00  -4.09
 -8.22  -8.67  -4.57
 -7.57   1.75  -8.61
 -3.03   2.86   8.99

LAPACKE_dgesv (row-major, high-level)

 Solution
 -0.80  -0.39   0.96
 -0.70  -0.55   0.22
  0.59   0.84   1.90
  1.32  -0.10   5.36
  0.57   0.11   4.04

 Details of LU factorization
  8.23   1.08   9.04   2.14  -6.87
  0.83  -6.94  -7.92   6.55  -3.99
  0.69  -0.67 -14.18   7.24  -5.19
  0.73   0.75   0.02 -13.82  14.19
 -0.26   0.44  -0.59  -0.34  -3.43

 Pivot indices
      5      5      3      4      5

C:\Users\julie\Desktop\Example4-JL\bi
```

```
5>  example_DGESV_rowmajor.c
4>  example_DGESV_colmajor.c
4>  example_DGESV_colmajor.vcxproj -> C:\Users\julie\Desktop\Example4-JL\bin\Debug\example_DGESV_colmajo
5>  example_DGESV_rowmajor.vcxproj -> C:\Users\julie\Desktop\Example4-JL\bin\Debug\example_DGESV_rowmajo
6>------ Build started: Project: ALL_BUILD, Configuration: Debug Win32 ------
6>  Build all projects
6>  Building Custom Rule C:/Users/julie/Desktop/Example_C_Fortran_CMAKE/Example_C_Fortran_CMAKE/Example4
6>  CMake does not need to re-run because C:\Users\julie\Desktop\Example4-JL\CMakeFiles\generate.stamp i
========== Build: 6 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Download Example4:
http://icl.cs.utk.edu/lapack-for-windows/Example_C_Fortran_CMAKE/Example4.tgz

Summary

CMake has a rich set of tools to enable the cross platform development of C/C++ code that compiles and links to Fortran source code. CMake can determine compiler symbol mangling as well as the correct run time libraries required to use the Fortran code from C/C++. The new CMake_add_fortran_subdirectory function currently under development will give VS studio users a free Fortran compiler available for use from the Visual Studio IDE using the MinGW gfortran compiler.

Thanks to all those great CMake features and to an optional development of a C Interface, the Fortran library can now be called from C directly from Microsoft Visual Studio without putting undue burden on the user.

Thanks

The CMake team would like to thank the Trilinos (http://trilinos.sandia.gov/) and DAKOTA (http://dakota.sandia.gov/) teams for their input and collaboration on the CMake Fortran support.