Compiling FAST v8.12.00a-bjj

Bonnie Jonkman
National Renewable Energy Laboratory
October 5, 2015

Introduction

FAST v8.12.00a-bjj is distributed with several options for compiling the source code. It contains a Visual Studio project (XML) file, a Windows® batch file for Intel Fortran, and a makefile for gfortran. The FAST archive contains all of the source files necessary to compile, and the compiling tools are set up with the correct paths to those files. The tools also set up all the compiling options required to compile FAST.

The compiling tools distributed with FAST v8.12.00a-bjj were created primarily for Windows® systems. However, the makefile can be used on non-Windows® systems. Please see "Compiling and Running on Linux or Mac OS" at the end of this document for some suggestions.

We have developed these instructions with the following two compilers:

- Intel® Visual Fortran (IVF) Composer XE 2013 SP1, version 14.0.5.239
  Intel Math Kernel Library (MKL) 11.1 Update 4
  Microsoft Visual Studio Community 2013, Version 12.0.31101.00 Update 4
- gfortran/gcc version 4.6.2
  mingw32

The MKL is used only for LAPACK routines.

Please note that FAST v8.12.00a-bjj is distributed with executables of both 32-bit and 64-bit Windows applications. Unless you are compiling for a different architecture or are adding functionality to FAST, you should not have to recompile the code.

Compiling with Intel Fortran in Visual Studio

Open "FAST_Project.vfproj" in the Compiling\VisualStudio directory of the FAST v8 archive. This should open a Solution Explorer window that lists all of the source files and Registry input files for FAST v8. See Figure 1.

Choose either Debug or Release configuration from the Configuration manager (Debug is the default), and then select "Build Solution" from the "Build" menu. This will run the FAST Registry (when necessary) to produce the *_Types.f90 files needed in the project and then will compile and link the project. When it has successfully completed, it will place an executable called "FAST_dev_Win32.exe" (Release mode) or "FAST_dev_debug_Win32.exe" (Debug mode) in the <FAST v8>\bin folder. (Note that if you are using Intel Visual Fortran 2015 or newer, the executable might be placed in the

<FAST8>\Compiling\VisualStudio folder. This is a result of differences in formats between versions of Intel Visual Fortran integrations with Visual Studio).

When you compile the first time, a box will probably pop up to ask you to save the solution file. Select “ok” to save the solution file in the same directory as your project file.

Note that sometimes the Registry will create a *_Types.f90 file that does not get compiled when building the solution. If that happens, you will need to select “Build Solution” a second time, so the *_Types.f90 file will compile.

Note that this project requires that you have the Intel® Math Kernel Library (MKL) to access LAPACK routines. MKL comes with newer versions of Intel® Visual Fortran (IVF), however some old versions of IVF required a separate license for MKL. If you have the Intel compiler without access to MKL, see the “LAPACK Libraries” section of this document, and see instructions at this web site: http://icl.cs.utk.edu/lapack-for-windows/lapack/ for help in installing LAPACK and BLAS libraries for Release and Debug modes.

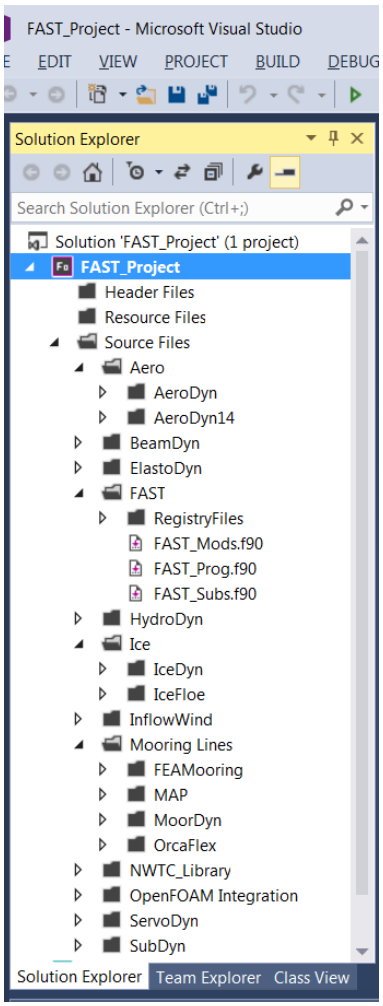

Figure 1. Source Files in the FAST v8.12.00 Visual Studio Project

Compiling with Intel Fortran from a Windows Command Prompt

FAST is distributed with a batch file called “Compile_FAST.bat” that will compile the code using Intel Visual Fortran (IVF) from the command line. However, if you are using IVF for Windows, we recommend using Visual Studio instead of the command prompt to compile.

Before using Compile_FAST.bat, either you must modify variables in the section labeled “set compiler internal variables” or you must run Compile_FAST.bat *only* from the compiler’s command prompt window.

Setting Compiler Environment Variables

In the “Set Compiler Internal Variables” section of the batch script, you make sure that the proper paths and environment variables are set for the compiler and linker. If this step is not done correctly, you will not be able to build the executable. The **blue** text shown in Figure 2 (copied from Compile_FAST.bat) must be changed to reflect your compiler.

```
REM ------------------------------------------------------------------------
REM                   set compiler internal variables
REM ------------------------------------------------------------------------
REM    You can run this bat file from the IVF compiler's command prompt (and not
REM    do anything in this section). If you choose not to run from the IVF command
REM    prompt, you must call the compiler's script to set internal variables.
REM    TIP: Right click on the IVF Compiler's Command Prompt shortcut, click
REM    properties, and copy the target (without cmd.exe and/or its switches) here:

CALL "C:\Program Files (x86)\Intel\Composer XE 2013 SP1\bin\ipsxe-comp-vars.bat" ia32 vs2013
```

Figure 2. The "Set Compiler Internal Variables section of Compile_FAST.bat

One way to find this command is to open the shortcut to the IVF command prompt (also called IVF Build Environment in some versions). You can usually find the shortcut at a location named something like **Start > All Programs > Intel** *CompilerName* **>** *CommandPromptName*. (Different versions of the compiler may have more submenus.) Note that if the shortcut says “non-compilation mode”, then you likely have a version of Visual Studio without the Intel Fortran integrations installed. You need to have the integrated product for this script to work.

Right click on the shortcut to the command prompt and click “Properties.” (See Figure 3 for an example.) A window similar to Figure 4 will open.

Copy the text from the Shortcut’s “Target” field and paste it in the Compile_FAST.bat script:

```
C:\Windows\SysWOW64\cmd.exe /E:ON /V:ON /K ""C:\Program Files
(x86)\Intel\Composer XE 2013 SP1\bin\ipsxe-comp-vars.bat" ia32 vs2013"
```

You will need to remove the call to cmd.exe and its switches, leaving you with the name of a batch file (and possibly some of its arguments):

```
"C:\Program Files (x86)\Intel\Composer XE 2013 SP1\bin\ipsxe-comp-vars.bat" ia32 vs2013
```

If you do not want to call this batch file from Compile_FAST.bat, you may remove the line from the file. However, you must then run Compile_FAST.bat *only* from the compiler’s command line window. Please refer to your compiler’s documentation about using **ifort** and calling it from the command line.

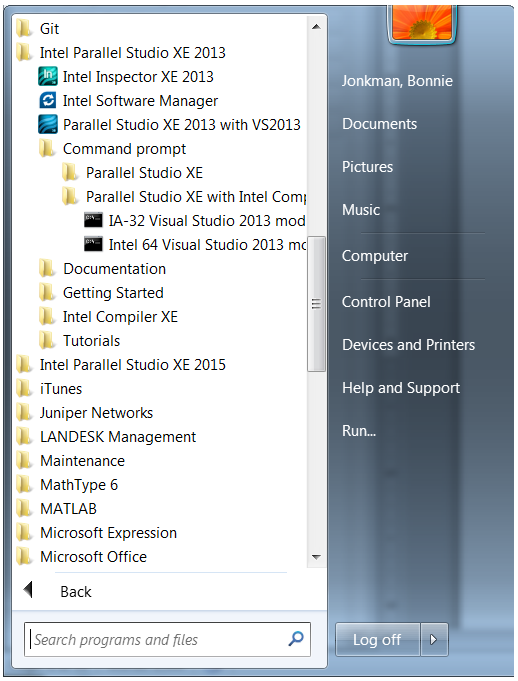

Figure 3. An example of finding the IVF command prompt shortcut

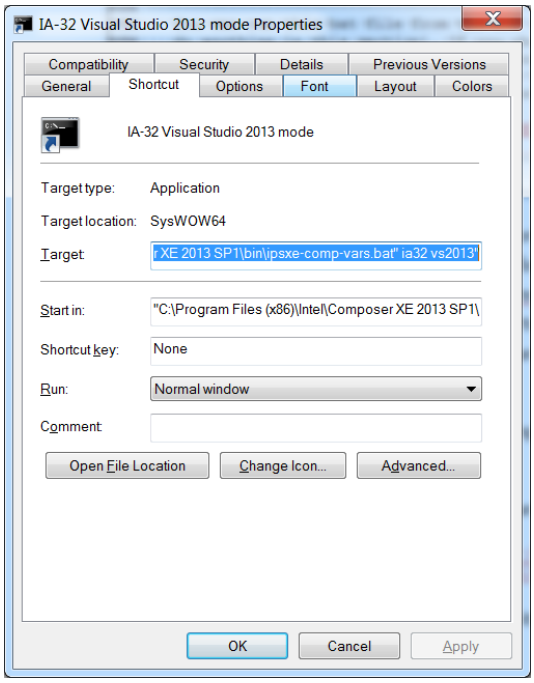

Figure 4. The properties window for an IVF command prompt shortcut

Running the Batch File

After you have modified Compile_FAST.bat, you can run it from the command line by typing

```
Compile_FAST.bat
```

in the directory where the batch file is stored. Figure 5 shows the screen output after a successful build using Intel® Composer XE 2011. Notice the title of the command window after Compile_FAST.bat has been run. The script that you call in the "Set Compiler Internal Variables" section to set the paths and environment variables for the IVF compiler also modifies the title of the window. **If the title does not say anything about the compiler and Visual Studio versions, please verify that you have modified Compile_FAST.bat correctly.**

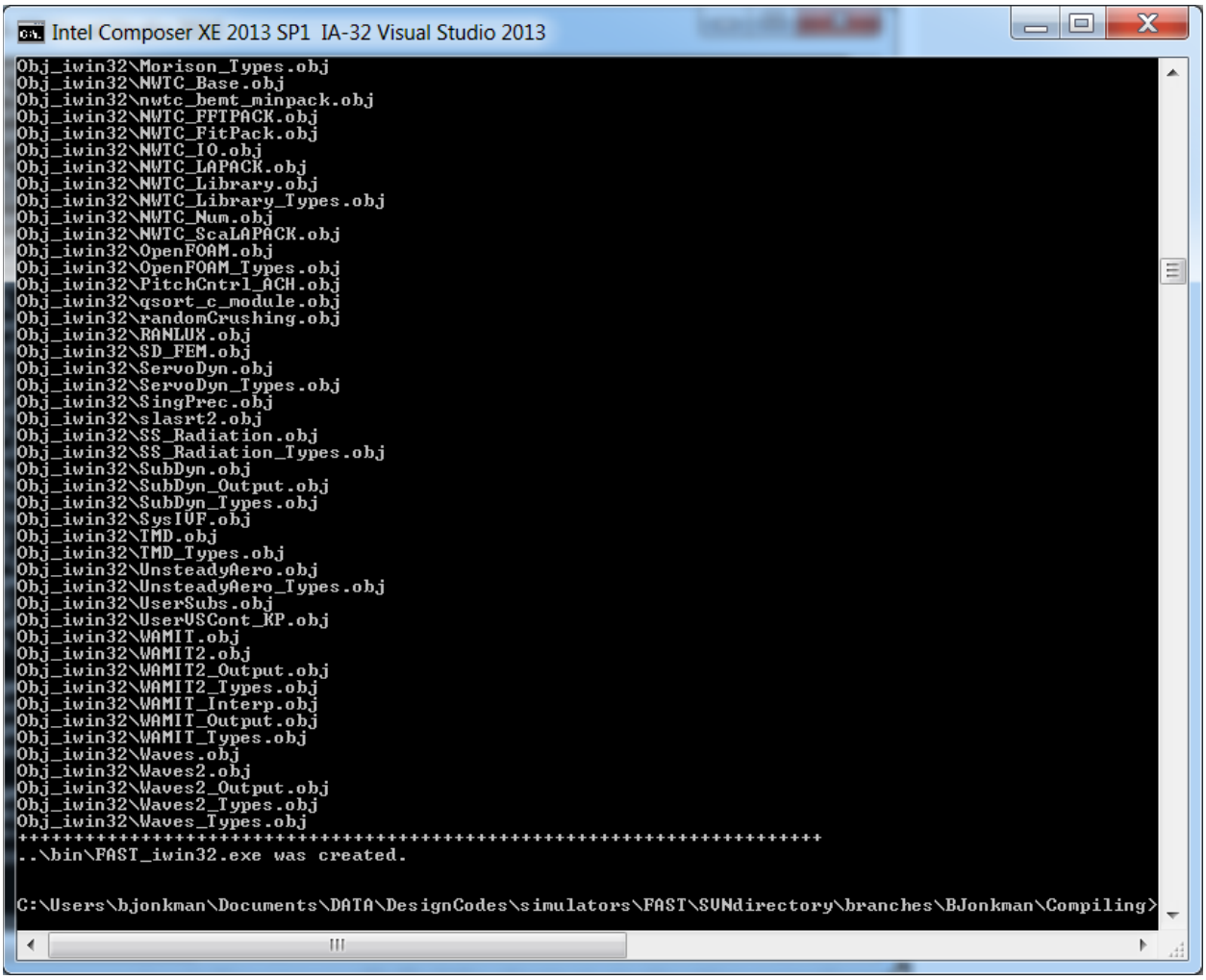

Figure 5. Screen output from Compile_FAST.bat. (Note that the title of the command window indicates that the batch script to set the compiler and linker paths was successfully run)

Compiling with gfortran for Windows

The makefile in the FAST\Compiling folder can be used to compile FAST v8 using gfortran.

gfortran (gcc) for Windows will require you to download the LAPACK and BLAS binary files, which you can get here: http://icl.cs.utk.edu/lapack-for-windows/index.html. For 32-bit Windows, you will need these files:

- http://icl.cs.utk.edu/lapack-for-windows/libraries/VisualStudio/3.5.0/Dynamic-MINGW/Win32/liblapack.dll
- http://icl.cs.utk.edu/lapack-for-windows/libraries/VisualStudio/3.5.0/Dynamic-MINGW/Win32/liblapack.lib
- http://icl.cs.utk.edu/lapack-for-windows/libraries/VisualStudio/3.5.0/Dynamic-MINGW/Win32/libblas.dll

- http://icl.cs.utk.edu/lapack-for-windows/libraries/VisualStudio/3.5.0/Dynamic-MINGW/Win32/libblas.lib

When you use gfortran for Windows, you will need to make sure these .dll files are on your Windows PATH so that the executables will run. The .lib files are required for linking.

To use the LAPACK and BLAS libraries in gfortran, use linking options –llapack and –lblas. (if you are having trouble, this web site may also provide some useful information: http://www.math.utah.edu/software/lapack.html)

Compiling and Linking Options

The options required to compile and link FAST v8.12.00a-bjj are already set in the compiling tools distributed with the code. However, it is sometimes useful to modify the options for debugging or other purposes.

Table 1 lists several useful compiling options. The first column of the table indicates which options are required to compile FAST.

We also require some additional linking options:

- You must link with the MAP library. On Windows, this means adding MAP_Win32.lib (or MAP_x64.lib) to the linking command. On non-Windows machines, this means adding libmap-{version}.so to the linking command.
- Some parts of the code place large arrays on the stack, and we have found that on Win32 systems, we sometimes need to increase the stack reserve size. The command option is:
  `/STACK:999999999` (IVF)
  `-Wl,--stack=999999999` (gfortran)
- Also on some large models (e.g., OC4 Jacket), we can exceed the 2-GB Windows memory limit for 32-bit processes. We can extend this limit when running the application on 64-bit computers by adding the option to Enable Large Addresses.
  `/LARGEADDRESSAWARE` (IVF)
  `-Wl,--large-address-aware` (gfortran)

If you compile a 64-bit version of FAST, you can omit the /STACK and /LARGEADDRESSAWARE options.

Double Precision

To compile FAST in double precision:

- Use the preprocessor definition DOUBLE_PRECISION (for the NWTC Subroutine Library). This can be done from the command line by adding –DDOUBLE_PRECISION to the compile options, or using the appropriate GUI in Visual Studio (see Figure 6).

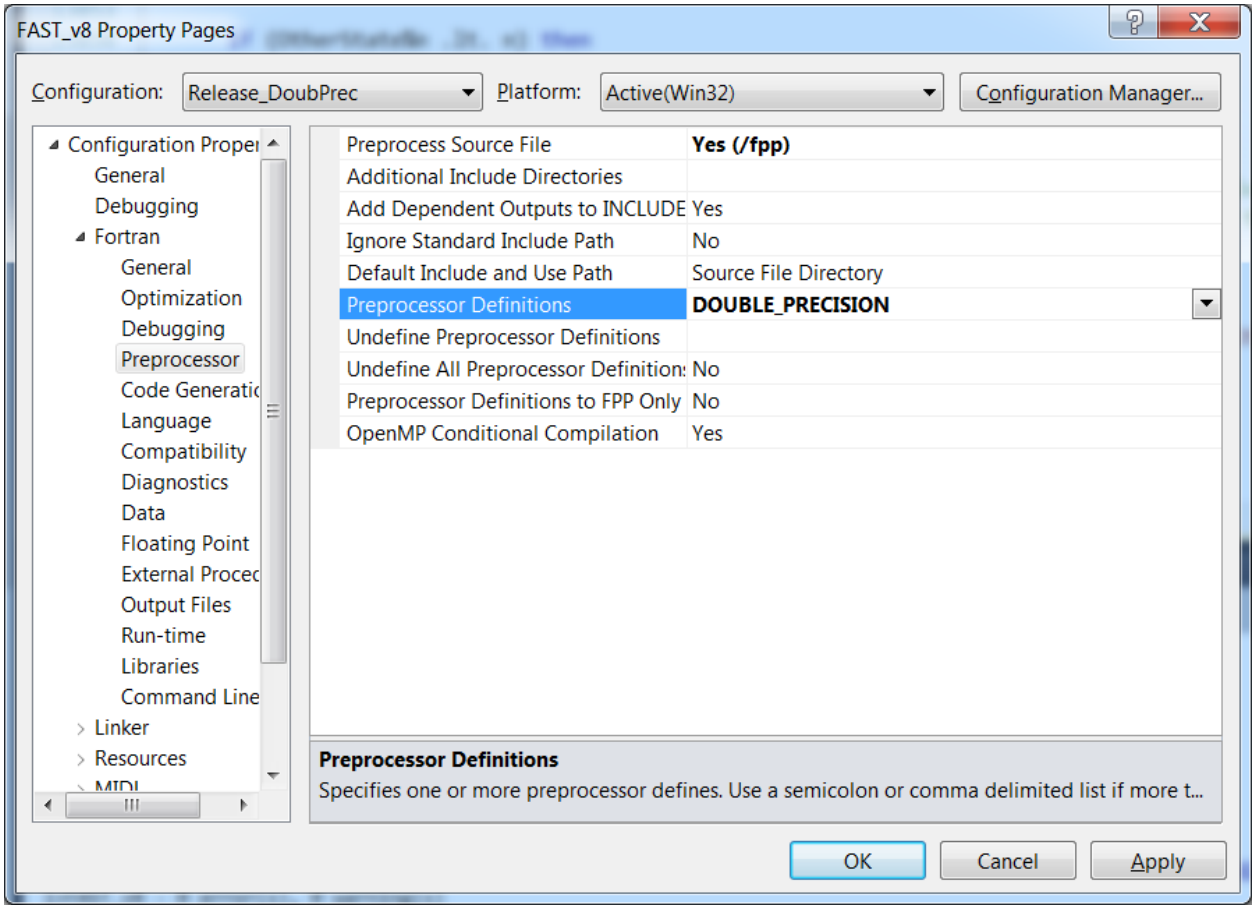

Figure 6: Adding preprocessor definitions to a Visual Studio project

- Some of the Third Party code we use also requires setting the default Real and Double KINDs to 8 and 16 bytes, respectively. (Default REAL must be the same as ReKi and Default DOUBLE must be the same as DbKi.)

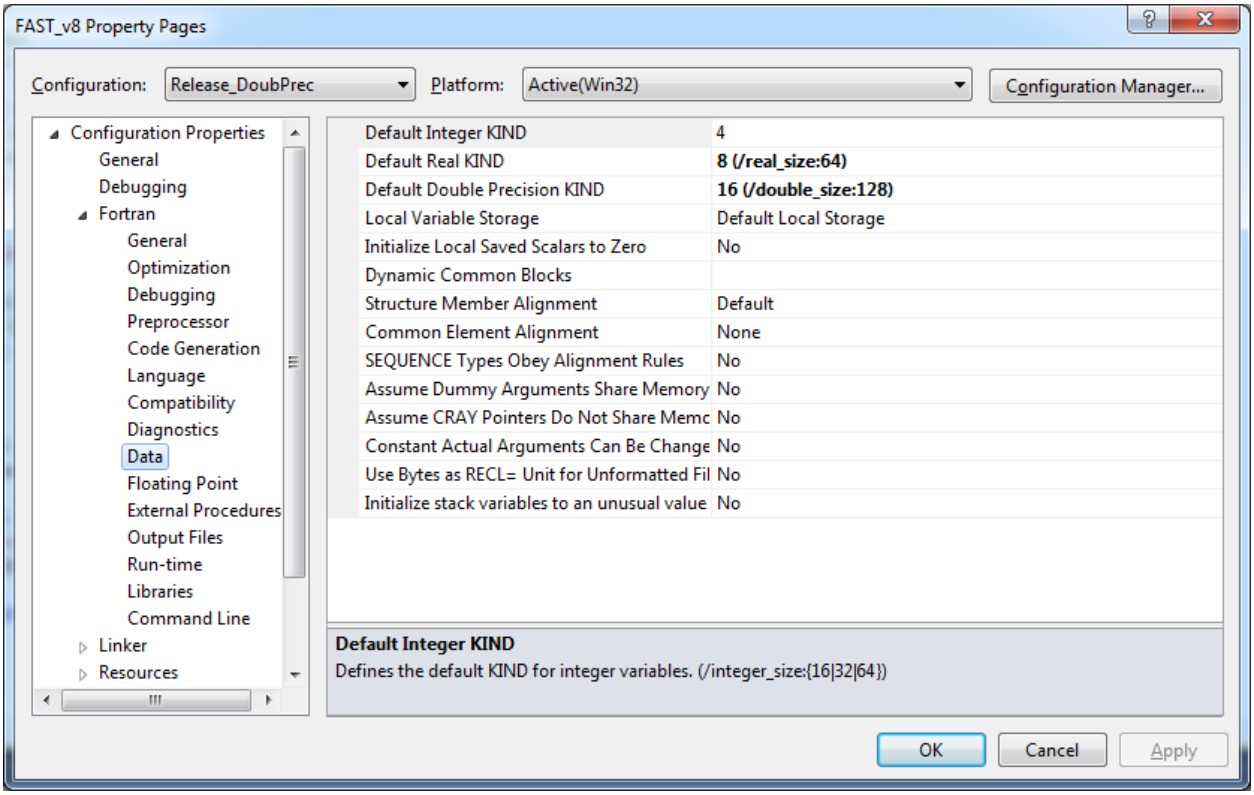

Figure 7: Changing default real and double sizes in Visual Studio

LAPACK Libraries

FAST v8 uses some LAPACK routines (http://www.netlib.org/lapack/index.html). We have made the decision to link with prebuilt libraries, which should include highly optimized versions of the Basic Linear Algebra Subprograms (BLAS).

These prebuilt libraries typically come installed on Linux and Mac operating systems. Use linking options –llapack and –lblas for gfortran.

If you are using the Intel compiler, you can use the Intel® Math Kernel Library. To activate MKL, you will need to set *Project* -> {project name} *Properties -> Configuration Properties -> Fortran -> Libraries -> Use Intel Math Kernel Library* to **Sequential (/Qmkl:sequential)**.

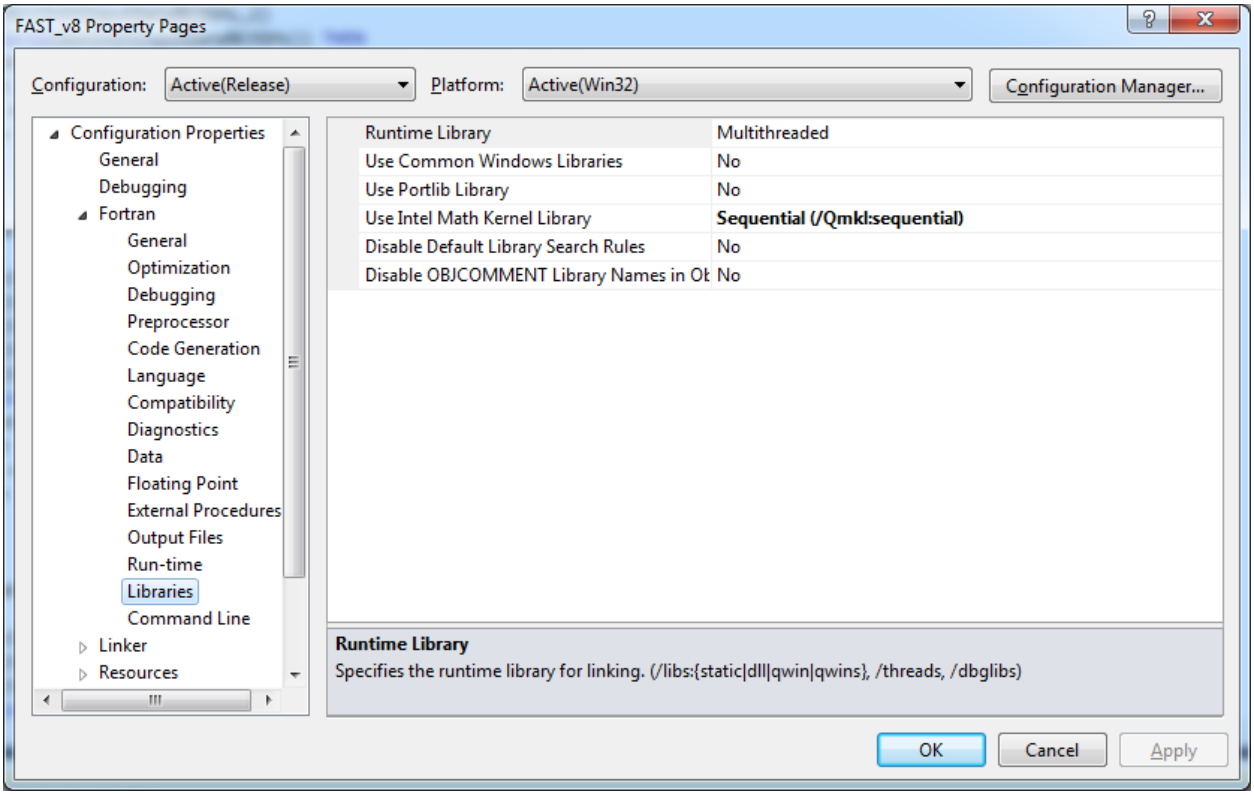

Figure 8. Setting Visual Studio to use LAPACK

With this option set, you can compile and use any routine in the LAPACK libraries.

If you have an older version of the Intel compiler, you may not have access to MKL. In that case, please see the instructions at this web site: http://icl.cs.utk.edu/lapack-for-windows/lapack/ for help in installing LAPACK and BLAS libraries for Release and Debug modes.

LAPACK in Double Precision

To facilitate compiling in double precision, we have created some wrapper routines in for the LAPACK and ScaLAPACK routines used in FAST and its modules.

The NWTC wrapper routines for LAPACK have been written assuming that they are calling prebuilt libraries. If you choose to compile with the LAPACK source files instead of using the prebuilt libraries (not recommended), you must make sure that default Real and Double KINDs for the LAPACK routines are 4 and 8 bytes, respectively (*even when compiling FAST in double precision*).

Table 1: Fortran Compiling Options

	Used in FAST?	Intel Fortran (Windows)	Intel Fortran (Linux and Mac OS)	GNU Fortran
Data options				
Define all default real (and complex) variables as 4 bytes long	Recommended *(compiler default)*	/real-size:32	-real-size 32	*(set by default)*
Define all default real (and complex) variables as 8 bytes long		/real-size:64	-real-size 64	-fdefault-real-8
Optimization options				
Disable all optimizations (debug mode)	Recommended for Debug	/Od	-O0	-O0
Enable optimizations for speed (release mode)	Recommended for Release	/O2	-O2	-O2
Enable higher optimizations (may set other options; may not be appropriate for all codes)		/O3	-O3	-O3
External libraries				
Use optimized LAPACK routines	**Required**	/Qmkl:sequential	-mkl=sequential	-llapack -lblas*
Debugging options				
Provide source file traceback information when a severe error occurs at run time		/traceback	-traceback	-fbacktrace
Check array subscripts		/check:bounds	-check bounds	-fcheck=bounds
Fortran Dialect Options				
Produce warning/error for non-standard Fortran 2003 code	Recommended	/stand:f03	-std03	-std=f2003
Allow free-format code to exceed 132 columns	**Required**	*(allowed by default)*		-ffree-line-length-none
Other				
Display compiler version information		/logo	-logo -V	-v --version
Preprocess source files before compilation	**Required**	/fpp	-fpp	-x f95-cpp-input
Create 32-bit code		*(use appropriate Visual Studio configuration or call appropriate script prior to using compiler from command line)*		-m32
Create 64-bit code				-m64

* These are actually linking options in gfortran.

Compiling and Running on Linux or Mac OS

The makefile distributed in the <FAST8>/Compiling directory has been tested with Mac OS X and some Linux systems. Before compiling the FAST v8 executable, though, you will need to compile a few components.

FAST Registry

We have included the source code for the FAST Registry in the <FAST8>/Source/dependencies/Registry folder. That folder also contains a makefile that will generate the FAST Registry executable using gcc. In the Registry folder, perform the command

```
make
```

This command will generate an executable named “registry.exe” in the <FAST8>/Source/dependencies folder.

MAP++ Library

The source files for the MAP++ library are not included in the FAST archive, but they are available on the MAP++ web site: https://nwtc.nrel.gov/MAP . The MAP++ archive contains a makefile (in its src folder) that can be used to generate the MAP++ library.

Running the “make” command should produce a .so file in the MAP++ src folder. At this writing, the file is called “libmap-1.10.01.so”. When finished, copy libmap-1.10.01.so to the <FAST8>/bin directory, where the FAST makefile will look for the file.

When running FAST, the system must be able to load libmap-1.10.01.so. This can be accomplished in several ways, including putting it in the directory where you run FAST, copying it to /usr/lib, or changing DYLD_LIBRARY_PATH on Mac OS.

FAST Executable

Before running the FAST v8 makefile, make sure you have the FAST registry executable and MAP++ library compiled for your operating system. The FAST Registry executable file will be necessary only if you modify any of the FAST Registry input files or if you perform the command

```
make superclean
```

for creating the FAST executable.

The makefile has been set up to link with the file ../bin/libmap-1.10.01.so. If this MAP++ library file has a different name, you will need to edit the makefile accordingly.

When the FAST Registry and MAP++ library have been generated, you can run the command

```
make
```

in the <FAST8>/Compiling directory. This should generate a file called FAST_glin32 in the <FAST8>/bin directory. If you have set the makefile to compile with 64-bit addresses, the file will be called FAST_glin64.

DISCON: Discrete Controllers in Dynamically Loaded Libraries

If you wish to run any FAST model that uses the DISCON dynamically loaded libraries (e.g., CertTests 18-26), you must compile the control algorithms as a dynamic library (shared object). The DISCON routine is an industry-standard interface (Gerrad Hassan's Bladed Interface). The source files for each of the controllers used in CertTests 18-26 are in the <FAST8>/CertTest/5MW_Baseline/ServoData/Source folder, and a makefile (makefile_DISCON_DLL) to compile them is in <FAST8>/Compiling. Running the command

```
make -f makefile_DISCON_DLL
```

in the Compiling folder will generate a file called DISCON_glin32.so in the <FAST8>/CertTest/5MW_Baseline/ServoData folder. This controller is used for Test 18 (and a few others). If you wish to compile a different source file, you will need to edit "makefile_DISCON_DLL" and modify variables **SOURCE_FILE** and **OUTPUT_NAME**.

When <FAST8>/CertTest/5MW_Baseline/ServoData/DISCON_glin32.so has been created, you must open the ServoDyn input file and tell it to read this library. Open the file <FAST8>/CertTest/5MW_Baseline/NRELOffshrBsline5MW_Onshore_ServoDyn.dat and change the input parameter "DLL_FileName" to "ServoData/DISCON_glin32.dll".

When that is complete, you can run the FAST CertTest 18 model on Mac OS X or Linux. For example, in the <FAST8>/CertTest folder, type:

```
../bin/FAST_glin32 Test18.fst
```

Note: if you compile a 64-bit version of FAST (i.e., FAST_glin64), you also need to compile a 64-bit version of the DISCON*.so file.

Also note that at this time, DISCON.f90 does not compile on Mac OS X with gcc version 5.2.0 (MacPorts gcc5 5.2.0_0)

Feedback

If you have questions or suggestions for improvements, please use our forums:
https://wind.nrel.gov/forum/wind/