FAST v8.12.00a-bjj

Bonnie Jonkman and Jason Jonkman

National Renewable Energy Laboratory

October 5, 2015

Table of Contents

Table of Contents 2
Introduction 3
Major changes in FAST 8
v8.12.00a-bjj 8
v8.10.00a-bjj 10
v8.09.00a-bjj 11
v8.08.00c-bjj 11
v8.03.02b-bjj 13
FAST v8 Input and Output Files 13
File Naming Conventions 13
Variables Specified in the FAST Primary Input File 16
Checkpoint Files (Restart Capability) 22
Converting to FAST v8.12.x 22
Summary of Changes to Inputs 23
MATLAB Conversion Scripts 26
Running FAST 28
Normal Simulation: Starting FAST from an input file 29
Restart: Starting FAST from a checkpoint file 29
Modeling Tips 29
Certification Tests 30
Compiling FAST 31
FAST v8 Interface to Simulink 32
Major Changes Between the FAST v7 and v8 Interfaces to Simulink 32
Definition of the FAST v8 Interface to Simulink 32
Converting FAST v7 Simulink Models to FAST v8 35
Running FAST in Simulink 36
Compiling FAST for Simulink 38
Future Work 41
Feedback 41
Appendix A: Example FAST v8.12.* Input File 42

Introduction

This document is designed to guide you through some of the changes that the FAST wind turbine multi-physics engineering tool is undergoing, until complete documentation is made available. FAST v8.12.00a-bjj is the latest public release of FAST under the new modularization framework developed at NREL. The architecture of FAST v8 is entirely different from FAST v7.02.00d-bjj. These differences are highlighted in Figure 1.

The modules of FAST (AeroDyn, HydroDyn, etc.) correspond to different physical domains of the coupled aero-hydro-servo-elastic solution, most of which are separated by spatial boundaries. Figure 2 shows the control volumes associated with each module for fixed-bottom offshore wind turbines. Though not shown, finite-element blade structural dynamics is optionally available through the BeamDyn module and loading from surface ice on fixed-bottom offshore wind turbines is optionally available through the IceFloe or IceDyn modules. For land-based wind turbines, the HydroDyn hydrodynamics module and ice modules would not be used and the SubDyn multi-member substructure structural-dynamics module is optional. Figure 3 shows the control volumes associated with each module for floating offshore wind turbines. Though not shown, finite-element blade structural dynamics is optionally available through the BeamDyn module and mooring and hydrodynamics are optionally available through the OrcaFlexInterface module.

While FAST v8 has many features not found in FAST v7, some features of FAST v7.02.00d-bjj have not yet been added to FAST v8, so we will continue to support both versions of the software (FAST v7 and FAST v8) until FAST v8 completely replaces FAST v7. Table 1 summarizes the different features available in each version.

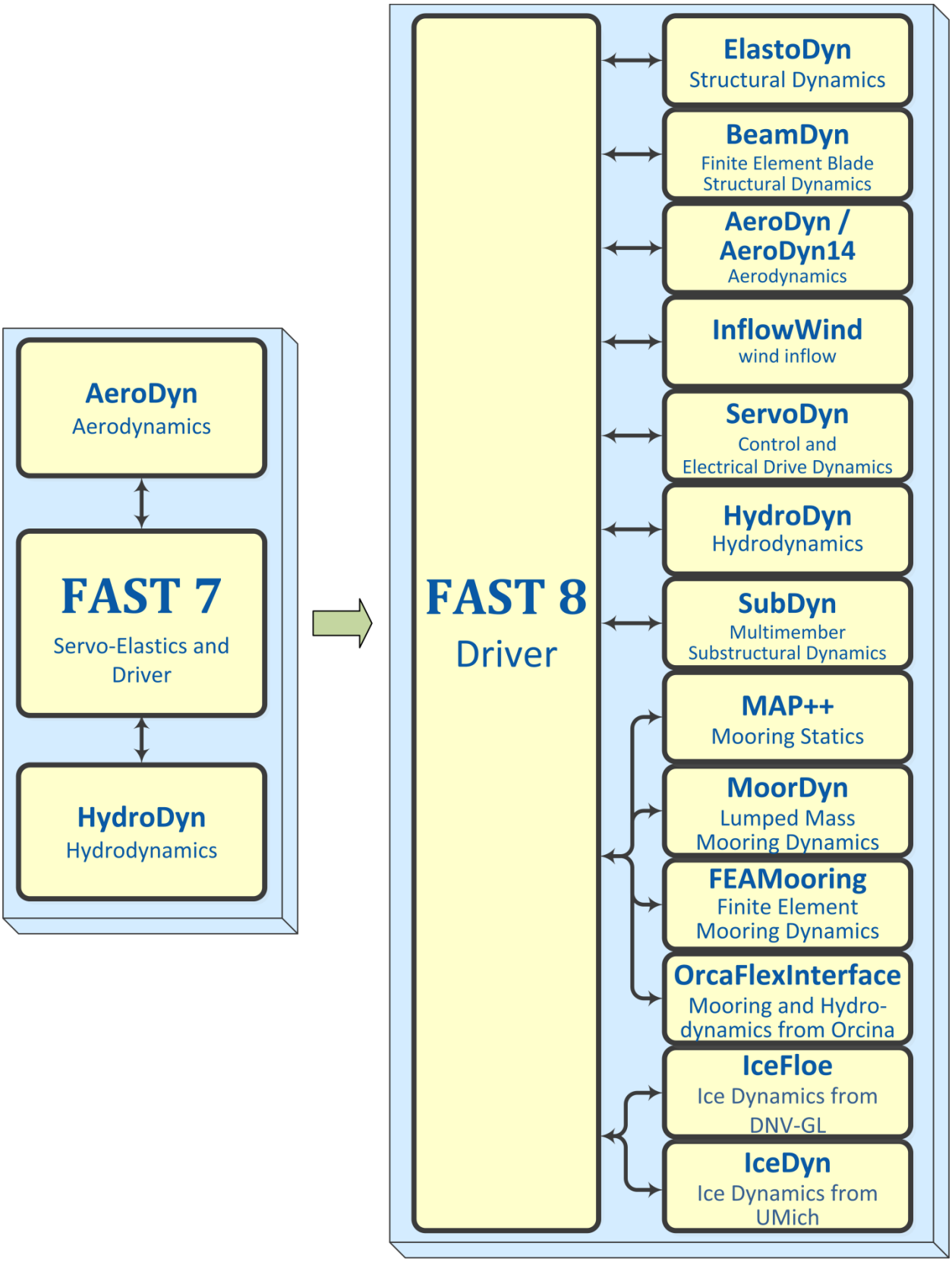

Figure 1: Architectural comparison of FAST 7 and FAST 8

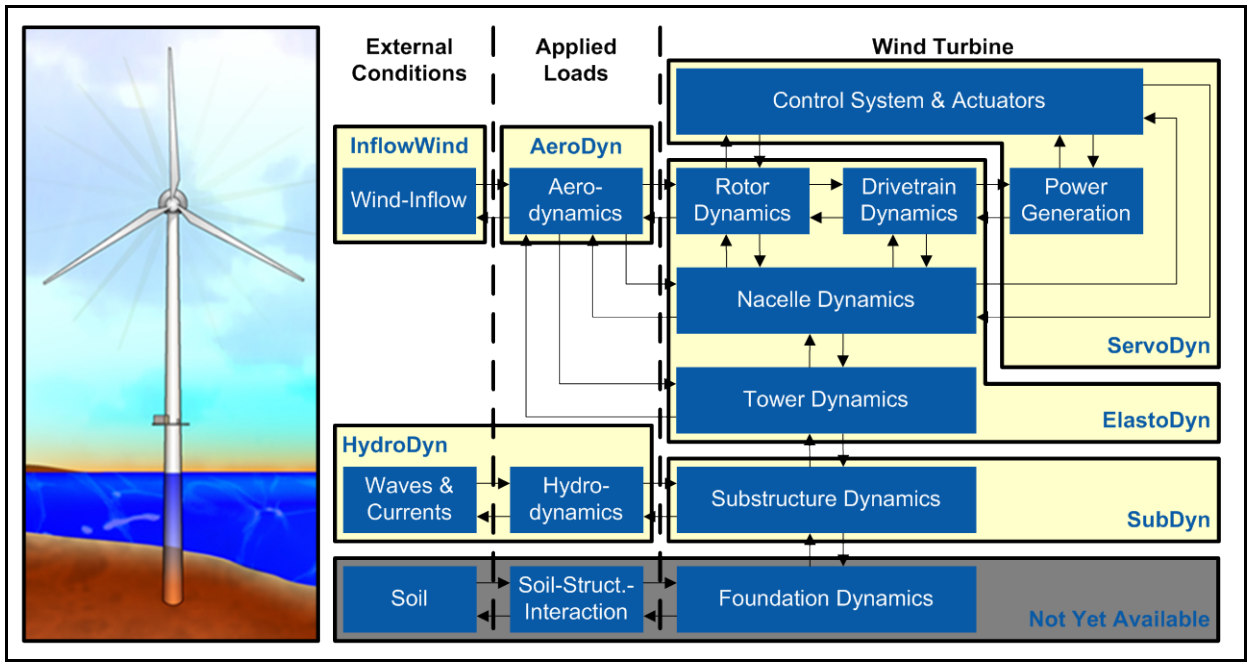

Figure 2: FAST control volumes for fixed-bottom systems (BeamDyn, IceFloe, and IceDyn not shown)

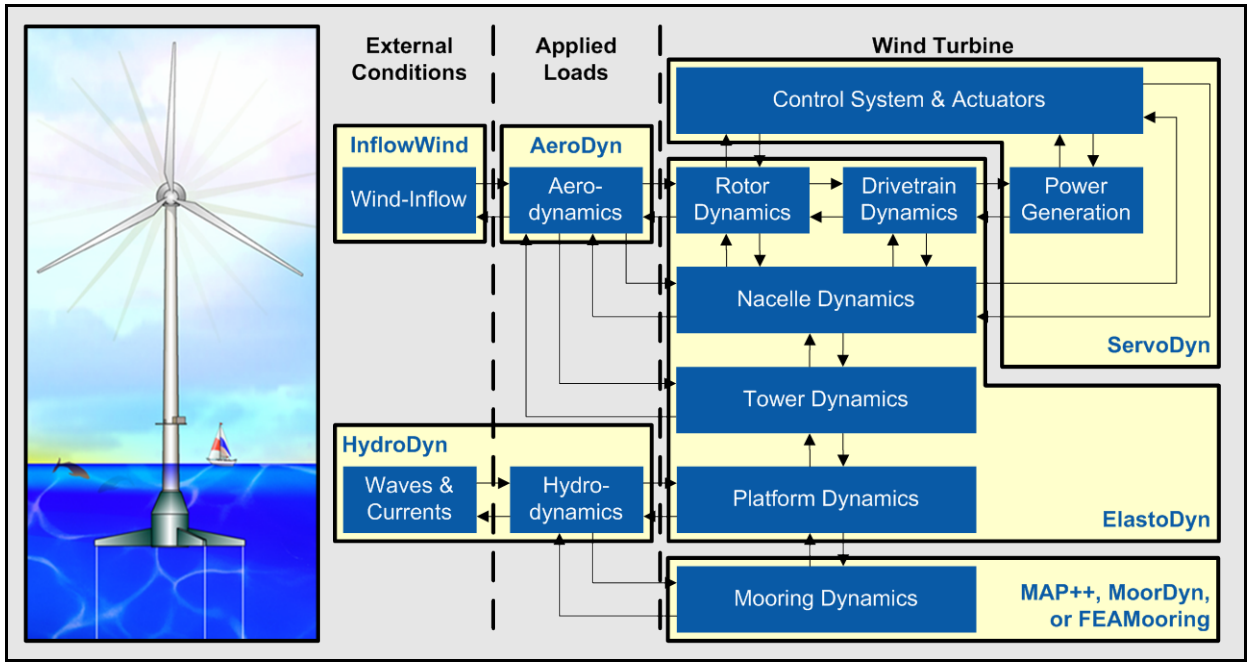

Figure 3: FAST control volumes for floating systems (BeamDyn and OrcaFlexInterface not shown)

Table 1: Comparison of features between FAST v7 and v8

Aerodynamics (AeroDyn and InflowWind)

FAST Features	v7.02	v8.12
• Quasi-steady or dynamic wake	✓	✓
• Steady or unsteady airfoil aerodynamics	✓	✓
• Support for non-straight and highly flexible blades		✓
• Tower shadow for downwind rotors	✓	✓
• Tower influence for upwind rotors	✓	✓
• Tower drag loading		✓
• Tail-fin aerodynamic loading	✓	
• "Hub-height", TurbSim, and GH Bladed wind file formats	✓	✓
• HAWC wind file format		✓
• Superimposed coherent turbulent structures from TurbSim	✓	
• Wind propagates in arbitrary directions		✓
• Aeroacoustics (noise)	✓	

Hydrodynamics (HydroDyn, OrcaFlexInterface, IceFloe, and IceDyn)

FAST Features	v7.02	v8.12
• First-order regular or irregular waves	✓	✓
• Second-order difference- and sum-frequency wave terms		✓
• White-noise waves		✓
• Wave directional spreading		✓
• Wave stretching	✓	
• Externally generated wave data	✓	✓
• Sea current	✓	✓
• Strip theory for central member	✓	✓
• Strip theory for multiple intersecting members		✓
• Distributed static buoyancy		✓
• Concentrated loads on member ends		✓
• Support for inclined and tapered members		✓
• Support for flooded and ballasted members		✓
• Support for marine growth		✓
• First-order potential flow (via WAMIT)	✓	✓
• Second-order diff.- and sum-frequency potential-flow terms (via WAMIT)		✓
• Radiation "memory effect" captured through time-domain convolution	✓	✓
• Radiation "memory effect" captured through linear state-space form		✓
• OrcaFlex interface*	✓	✓
• Quasi-steady and dynamic surface-ice loading		✓

* This option is a custom feature in FAST v7, requiring a separate executable. In FAST v8, it is part of the standard distribution.

Control and Electrical System (Servo) Dynamics (ServoDyn)

FAST Features	v7.02	v8.12
• Blade-pitch control	✓	✓
• Override pitch maneuvers	✓	✓
• Generator models	✓	✓
• Torque control	✓	✓
• High-speed shaft brake	✓	✓
• Nacelle-yaw control	✓	✓
• Override yaw maneuvers	✓	✓
• Blade-tip brakes	✓	
• Nacelle-based tuned-mass dampers		✓
• GH Bladed DLL interface*	✓	✓
• Simulink interface	✓	✓
• LabVIEW interface	✓	

Structural Dynamics (ElastoDyn, BeamDyn, SubDyn, MAP++, MoorDyn, FEAMooring, and OrcaFlexInterface)

FAST Features	v7.02	v8.12
• Blade-bending degrees-of-freedom (DOFs)	✓	✓
• Blade-shear, -extensional, and -torsion DOFs		✓
• Support for non-straight, composite, and highly flexible blades		✓
• Rotor-teeter DOF	✓	✓
• Generator azimuth and drivetrain torsion DOFs	✓	✓
• Nacelle-yaw DOF	✓	✓
• Tower-bending DOFs	✓	✓
• Rigid-body platform DOFs	✓	✓
• Furling DOFs	✓	
• Fixed-bottom multi-member substructure DOFs		✓
• Gravitational loading	✓	✓
• Gearbox friction	✓	✓
• System of independent mooring lines solved quasi-statically	✓	✓
• System of multi-segmented mooring lines solved quasi-statically		✓
• Mooring dynamics		✓
• OrcaFlex interface*	✓	✓
• Earthquake excitation	✓	

General

FAST Features	v7.02	v8.12
• Time marching	✓	✓
• Operating-point determination	✓	
• Linearization	✓	
• FAST-to-ADAMS preprocessor	✓	
• Follows the new FAST modularization framework		✓
• Structural and control routines separated from driver code		✓
• Independent time steps between modules	✓[†]	✓[‡]
• Independent spatial discretization between modules		✓
• Multiple integration options		✓
• Loose coupling with predictor-corrector across modules	✓[§]	✓
• Checkpoint-restart capability		✓
• Both 32-bit and 64-bit applications available	✓	✓
• Supports both Windows and Linux operating systems	✓	✓
• Optimized for efficiency	✓	
• Supports mixed Fortran/C		✓
• Compiles with gfortran	✓	✓

Major changes in FAST

v8.12.00a-bjj

- We introduced new features to support the analysis of advanced aero-elastically tailored blades:
  - A new structural-dynamics module (BeamDyn) for blades has been introduced. BeamDyn is based on geometrically exact beam theory (GEBT) and implemented using Legendre spectral finite elements. The model includes full geometric nonlinearity supporting large deflection, with bending, torsion, shear, and extensional DOFs; anisotropic composite material couplings (using full 6x6 mass and stiffness matrices, including bend-twist coupling); and a reference axis that permits blades that are not straight (supporting built-in curve, sweep, and sectional offsets).When these advanced features are needed, BeamDyn replaces the more simplified blade structural model of ElastoDyn that is still available as an option, but is only applicable to straight isotropic blades dominated by bending. See the new documentation provided with BeamDyn for more information.
  - An overhauled version of the aerodynamics module (AeroDyn v15) has been introduced, including new capability for the modeling of highly flexible and non-straight blades.

[†] These steps must be integer multiples of the structural time step.
[‡] These steps must be integer divisors of the glue-code time step. Future versions will allow integer multiples of the glue-code time step as well.
[§] FAST v7 is limited to one correction step and this correction step only applies to some modules.

With AeroDyn v15, blade and tower discretizations are independent of discretizations in the ElastoDyn or BeamDyn modules. See the new documentation provided with AeroDyn v15 for more information. We will remove AeroDyn v14 after AeroDyn v15 is deemed a suitable replacement for it.

- As part of the AeroDyn overhaul, the InflowWind wind module has been separated from AeroDyn and made a core module of FAST, called from the FAST glue code, with its own input file. The updated InflowWind module supports HAWC wind file formats and arbitrary wind directions for all wind file types. See the new documentation provided with InflowWind for more information.
- We added integrations for the FAST v8-SOWFA/OpenFOAM interface. More details will be available at a later date when a release of SOWFA utilizes FAST v8.
- In ServoDyn, we added a time step for the Bladed-style DLL controllers independent from the ServoDyn time step, as well as a linear ramp and first-order low-pass filter applied to the blade-pitch command from the Bladed-style DLL.
- We updated ServoDyn to use version 4.0 of the Bladed-style DLL interface instead of version 3.6. In practice, this means ServoDyn passes a longer array to the DLL and sends it the value of the shaft torque.
- We added the ability to use externally generated wave elevations or full externally generated wave kinematics within the HydroDyn hydrodynamics module. See the updated documentation provided with HydroDyn for more information.
- We added an interface between FAST and the OrcaFlex commercial software package developed by Orcina for advanced hydrodynamic and mooring analysis and design. See the documentation provided with the OrcaFlex interface for more information.
- The lumped-mass mooring-dynamics module MoorDyn has been completed; see the new documentation provided with MoorDyn for more information.
- We eliminated a memory leak in the FEAMooring module.
- Documentation for the quasi-static mooring module, MAP++, has been updated.
- The surface ice-dynamics module IceDyn from the University of Michigan has been completed; see the new documentation provided with IceDyn for more information.
- We added checkpoint-restart capability. See sections, “Checkpoint Files (Restart Capability)” and “Restart: Starting FAST from a checkpoint file” below for more information.
- We fixed a bug in SubDyn whereby the structural damping was not set properly when the Craig-Bampton reduction was disabled.
- The issue with compiling/running offshore cases with gfortran has been fixed. All certification tests now run with the gfortran executable (though it will produce different random waves).
- FAST v8 has now officially been released under the Apache 2.0 open-source license.
- FAST v8.10.00a-bjj is compiled with the components listed in Table 2.

Table 2: Components in FAST v8.12.00a-bjj

Component	Version	
Modules		ModName (for output files)
ElastoDyn	v1.03.00a-bjj	ED
BeamDyn	v1.00.00	BD
AeroDyn14	v14.04.00a-bjj	AD
AeroDyn	v15.00.00a-bjj	AD
InflowWind	v3.01.00a-adp	IfW
ServoDyn	v1.03.01a-bjj	SrvD
TMD (part of ServoDyn)	v1.00.01-wgl	TMD
HydroDyn	v2.03.00c-adp	HD
SubDyn	v1.02.00a-rrd	SD
MAP++	1.10.01	MAP
FEAMooring	v1.01.02-yhb	FEAM
MoorDyn	v1.00.00F-mth	MD
OrcaFlexInterface	v1.00.00a-adp	Orca
IceFloe	v1.00.01	IceF
IceDyn	v1.01.01-by	IceD
Other Components		
NWTC Subroutine Library	v2.06.05a-bjj	
FAST Registry**	v2.08.03	
Third Party Content		
LAPACK	v3.4.1 as part of Intel® Math Kernel Library; (v3.5.0 compiled with gfortran)	
ScaLAPACK	2.0.2	
FFTPACK	v4.1	
FitPack (Dierckx)	~1993?	

v8.10.00a-bjj

- We replaced the MAP module with MAP++ by external contributor Marco Masciola, an updated DLL that does not depend on the external libraries that previously made MAP difficult to recompile. The source code for MAP++ is available and able to be compiled with standard C compilers.
- The 64-bit version of the MAP DLL is now functioning. This means *all* modeling functionality in FAST_Win32.exe is available in FAST_x64.exe.
- We developed a new interface between FAST v8 and Simulink. This feature is described in detail in section "FAST v8 Interface to Simulink" later in this document.
- We added high-speed shaft braking to ServoDyn. This feature is compatible only with ElastoDyn's ABM4 or AB4 integrators (i.e., it is not available with RK4).

** The FAST Registry is a separate executable that reads input files from each module to auto-generate the *_Types.f90 files required for the FAST framework.

- We added gearbox friction to ElastoDyn, allowing the user to specify **GBoxEff** less than 100%. This feature is not currently available when the RK4 integrator is used.
- We fixed some bugs in HydroDyn that could cause the code to stop working with still water or with certain 3D and 4D datasets used in second-order WAMIT calculations.
- We added another module for computing mooring line dynamics: MoorDyn by external contributor Matthew Hall. The version of MoorDyn released with FAST v8.10.00a-bjj is still considered incomplete.
- We added nacelle-based tuned mass dampers to ServoDyn using the new TMD submodule. TMD is a submodule developed by the University of Massachusetts; it simulates two independent, one-DOF, linear mass-spring-damping elements that act in the for-aft (x) and side-to-side (y) directions of the nacelle. Documentation is available at https://nwtc.nrel.gov/TMD.
- We added rotational interpolation to the mesh mapping routines.
- Documentation for the IceFloe module has been released at https://nwtc.nrel.gov/IceFloe.
- Documentation and sample input files for the FEAMooring module have been released at https://nwtc.nrel.gov/FEAMooring.
- Documentation for the DWM routines and the DWM wind farm driver has been released at https://nwtc.nrel.gov/DWM.

v8.09.00a-bjj

- We added second-order wave kinematics and second-order diffraction loading to HydroDyn. Before this update, HydroDyn was previously based solely on first-order hydrodynamics theory. The second-order hydrodynamic implementations include time-domain calculations of difference- (mean- and slow-drift-) and sum-frequency terms. Second-order wave kinematics are applicable to the hydrodynamic loading of thin structural members via strip theory and second-order diffraction loads are applicable to large structural members.
- We fixed a problem where the eigenfrequencies listed in the SubDyn summary file were incorrect for some models.
- We fixed a problem with the tower distributed with the OC3-tripod model in the FAST certification test (Test20).
- We added source code for the DWM sub-module in AeroDyn. This feature was developed and will be supported by researchers at the University of Massachusetts; if you wish to use the DWM feature, you must obtain the DWM driver code, distributed separately: https://nwtc.nrel.gov/DWM.

v8.08.00c-bjj

- Coupling between ElastoDyn, SubDyn, and HydroDyn was added, allowing FAST to model fixed-bottom offshore turbines, including multi-member substructures (e.g., tripods and jackets).
- The module input-output solves have been enhanced; see the following paper for theoretical details: http://www.nrel.gov/docs/fy14osti/60742.pdf.

- The mesh mapping algorithms have been enhanced; see the following paper for theoretical details: http://www.nrel.gov/docs/fy14osti/60742.pdf.
- We now use LAPACK routines for solving linear systems, which has increased the speed of the simulations.
- The glue code allows the option for time-step subcycling. Modules can now choose to use *smaller* time steps than the glue code, as long as the module time step is an integer divisor of the glue-code time step. Note that we have found no cases where this option would be necessary.
- New modules for ice loading were added: IceFloe and IceDyn[††].
- Another module for mooring lines was added: FEAMooring[††].
- The names of output files generated by both FAST and its modules have been standardized, see Figure 4 and section "FAST v8 Input and Output Files."
- The "Time Ratio" displayed at the end of a simulation now includes only the CPU time *after* initialization. This ratio was changed to help users better predict the amount of time longer simulations will take (e.g., extrapolating the time a 1-hr simulation will take based on running a 1-min simulation).
- Information about the Jacobian and time steps was added to the FAST summary file.
- Bugs in handling errors were fixed. (In FAST v8.03.02b-bjj, InflowWind did not return all of its errors to the glue code, so it was using zero wind velocity when it went outside the turbulence grid.)
- FAST no longer allows the tower influence model, "NEWTOWER," to be used in AeroDyn on floating offshore turbines. This tower influence model assumes the tower does not move, which is a poor assumption for floating turbines.
- FAST will now abort if ElastoDyn's **TowerBsHt** value is negative for floating offshore systems.
- We fixed a bug in ElastoDyn (which is also present in FAST v7.02.00d) where the linear teeter-damper moment did not use **TeetDmpP.**
- We fixed a problem where the ElastoDyn Azimuth channel would be negative in rare cases.
- We fixed a problem with ElastoDyn's implementation of AM4, which incorrectly initialized the state history if corrections steps were taken.
- We no longer allow extrapolation orders of 0 in FAST v8.08.00c-bjj.
- We updated the DISCON*.DLL files used in the 5MW model certification tests. Previously, they did not work if users did not have Intel Visual Fortran installed on the computers they ran the simulations on.
- We fixed some bugs in the AeroDyn input files of the NREL 5-MW land-based turbine.
- We set the time steps of the floating offshore certification tests to be the same as they were in FAST v7.02.00d
- We added certification tests for the OC3 Monopile, OC3 Tripod, OC4 Jacket, and OC4-DeepCwind Semi-Submersible models.
- We have added some more utility files to the FAST archive, including:

[††] IceDyn and FEAMooring have been added to FAST v8.09.00a-bjj, but they are not complete and have not been tested well.

    - PlotFASToutput.m, a MATLAB function that can plot some or all channels of one or more FAST time-series output files.
    - ReadSubDynSummary.m, a MATLAB function that can read the SubDyn summary file and put the data into a MATLAB data-structure.
- We have updated the MAP_win32.dll file distributed with FAST so that it no longer depends on python being installed on the computer running FAST.
- We have added a 64-bit FAST executable to the archive, as well as a 64-bit version of DISCON_win64.DLL, and a "dummy" 64-bit version of MAP. This executable may be useful for running long simulations of large fixed-bottom offshore models (e.g., the OC4 Jacket); it *cannot* run any models that want to call the MAP module.

v8.03.02b-bjj

Tasks completed to develop FAST v8 included:

- Converted FAST and its various modules (including AeroDyn and HydroDyn) into the new modularization framework (splitting out the controls and electrical-drive dynamics into a new ServoDyn module and structural dynamics into a new ElastoDyn module),
- Implemented a new driver program (glue code) supporting loose coupling of the modules,
- Developed mesh-to-mesh mapping schemes between module-independent discretizations of the spatial boundaries between modules,
- Coupled in the recently developed SubDyn module for multi-member substructure structural dynamics and MAP module for multi-segmented mooring quasi-statics, and
- Included a series of models using the NREL 5-MW Baseline wind turbine in the CertTest, including offshore configurations.

The driver program (glue code) couples the modules together; it controls the overall simulation progress and maps module outputs to inputs. We use the name "FAST" both for the driver program (glue code) and overall coupled code.

FAST v8 Input and Output Files

FAST and each of its modules have their own input files; see Figure 4.

File Naming Conventions

Input files do not need particular extensions in FAST, though some modules may make their own assumptions on naming conventions (e.g. OrcaFlex interface).

Files generated by FAST are named

<RootName>.<ext>

and files generated by FAST modules are named

<RootName>.<ModName>.<ext>

where <RootName> is the root name of the primary FAST input file (the filename, including path,

without the extension), <ModName> is an abbreviation for the module generating the file (see Table 2), and <ext> is the file extension. File extensions currently are:

Output file extension	File type
sum	Summary file
out	Time-marching tabular text output
outb	Time-marching tabular binary output
ech	Echo of input file (primarily for debugging)
chkp	Checkpoint files for restart capability

If FAST generates checkpoint files, these checkpoint filenames are in the form

<RootName>.<timeStep>.chkp

where <timeStep> is an integer indicating at which time step the results in the file were generated. If the simulation was running a Bladed-style dll in ServoDyn, there will be a second checkpoint file named

<RootName>.<timeStep>.dll.chkp

When FAST is called as a library from Simulink, the output files are named

<RootName>.SFunc.<ext>

and

<RootName>.SFunc.<ModName>.<ext>

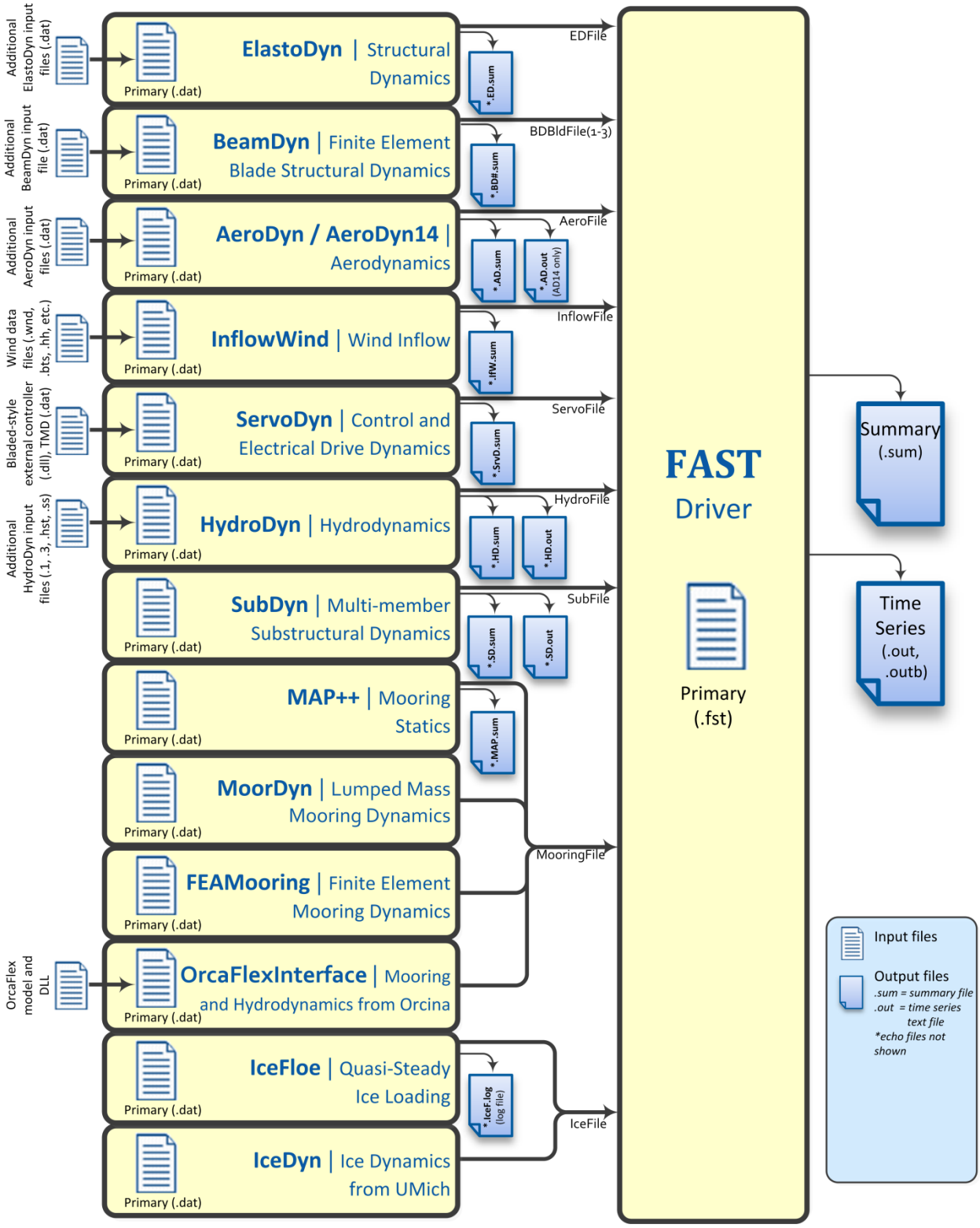

Figure 4: Summary of Input and Output Files for FAST v8.12.00a-bjj

Variables Specified in the FAST Primary Input File

FAST expects to find variables on specific lines in the text input file, so, do not add or remove lines in the file. The inputs listed in the file are described below, and an example file is provided at the end of this document, in Appendix A: Example FAST v8.12.* Input File.

Simulation Control

This section of the input file contains options for controlling the simulation.

Echo: Echo input data to <RootName>.ech [T/F]

Setting this flag to "True" will result in the FAST primary input file being echoed to a file named "<RootName>.ech" where <RootName> is the name of the FAST primary input file, excluding its file extension. This feature is useful for debugging an input file.

AbortLevel: Error level when simulation should abort ["WARNING", "SEVERE", or "FATAL"]

This string tells FAST what error level should cause an abort. Typically we set this to abort on fatal errors, but there may be instances when a user wishes to abort on severe errors or warnings.

Fatal errors are those from which the program cannot recover. For example:

- Running out of memory when trying to allocate space for variables.
- Trying to read a number from a line of an input file that does not contain numeric values.
- Reaching the end of an input file before reading all the necessary information.
- Trying to open a file for writing, but the file is locked from another process.

Some examples of severe errors include the following:

- A format specifier for real numbers is too narrow to print, so output files will almost certainly contain "***" instead of actual numbers.
- When trying to read a numeric value, logical "True" or "False" values were found instead. Fortran interprets them as 0 or 1, but that may not be what the user intended.
- A routine is using math based on the assumption that the angles are small, but the angles the routine found were larger than what it considers "small."

Warnings are typically generated when the simulation can continue—perhaps by the program adjusting inputs—but the results may not be what the user expected. Things that may generate warnings include

- Cases when user inputs are modified:
    - If the user asked for output on more tower strain gages than there are tower nodes, ElastoDyn will set the number of strain gages equal to the number of nodes.
    - If air density is set to zero, AeroDyn v14 will turn off the dynamic-inflow model.
- Cases where non-physical conditions could be modeled:
    - If the user enables ElastoDyn's second flap mode but does not enable the first flap mode.
    - If the user has disabled wake calculations in AeroDyn.

TMax: Total run time [s]

This is the total length of the simulation to be run, in seconds. The first output is calculated at *t* = 0; the last output is calculated at *t* = **TMax** seconds.

DT: Recommended module time step [s]

This is the global, or glue-code, time step; **DT** is the value FAST will suggest modules use, although some modules may choose to use a time step that is an integer multiple smaller than **DT**. Module input-output relationships used to couple the modules together are calculated every **DT** seconds. It is essential that a small enough time step is used to ensure solution accuracy (by providing a sufficient sampling rate to characterize all key frequencies of the system), to ensure numerical stability of the selected time-integrators, and to ensure that the coupling between modules of FAST is numerically stable.

Our rule of thumb is to set **DT** = 1 / (10 * highest natural frequency in Hz of coupling between modules). This natural frequency is hard to estimate before the full-system linearization of the coupled FAST v8 model is realized. For coupled FAST models that don't use BeamDyn or SubDyn, the frequency can be estimated via a linearization analysis of FAST v7. For coupled FAST models that do use SubDyn, guidance for choosing the time step is found in the SubDyn ReadMe file.

InterpOrder: Interpolation/Extrapolation order for input/output time history [1 or 2]

This is the order of the interpolation or extrapolation used for module inputs in the FAST glue code. Valid entries are "1" for linear interpolation/extrapolation or "2" for quadratic interpolation/extrapolation. Previous module inputs are extrapolated at the beginning of each step in the time-advancement loop to provide a guess for the actual module inputs at future times for those modules that rely on an implicit time-integrator. Module inputs are typically interpolated in a module's UpdateStates routine.

We have found that quadratic extrapolation typically works well. However, there are times when linear extrapolation provides a stable solution while quadratic does not. We have found this to be true for cases where the model has poor initial values or cases where the simulation may have errors building up.

NumCrctn: Number of correction iterations [-]

This is the number of corrections to be taken on each step of the predictor-corrector scheme implemented in FAST. The value of **NumCrctn** must not be negative. Most models can achieve stable solutions by using explicit calculations (i.e., no corrections: **NumCrctn** = 0), particularly if using **InterpOrder** = 2 and the recommended **DT**—see above. However, corrections may be needed if you wish to achieve a given convergence rate of an underlying time integrator (e.g., if you are using a 4^{th}-order accurate integration scheme, you may only get a 2^{nd}-order accurate solution with no corrections. If you desire a 4^{th}-order accurate solution, you may need one or more corrections).

DT_UJac: Time between calls to get Jacobians [s]

We use a Jacobian matrix to solve the module input-output relationship between accelerations and loads in the ElastoDyn-BeamDyn, ElastoDyn-HydroDyn-SubDyn, and ElastoDyn- OrcaFlexInterface

couplings. This Jacobian is computed with finite differences and can be time consuming. However, it rarely needs to be calculated frequently.

DT_UJac determines how often the Jacobian needs to be updated. For most models, **DT_UJac** can be set to a value larger than **TMax**. **DT_UJac** is not currently used for models that don't use the BeamDyn, HydroDyn, SubDyn, or OrcaFlexInterface modules. For floating systems where the platform may rotate more than several degrees in roll, pitch, and/or yaw, it is recommend to set **DT_UJac** = 1/(10*natural frequency in Hz of the roll, pitch, or yaw mode with excessive motion).

UJacSclFact: Scaling factor used in Jacobians [-]

This factor is used to divide the magnitude of the load terms in the Jacobian (see **DT_UJac)** so that they are approximately the same order of magnitude as the acceleration terms. We recommend setting **UJacSclFact** equal to a value roughly the same order of magnitude as the total system mass in kg. For the NREL 5-MW turbine models in the Certification Test, we've set it to 1E+06 and have not found any cases where that value did not work. **UJacSclFact** may need to be set larger or smaller when modeling wind turbines much larger or smaller turbines than the NREL 5-MW baseline.

Feature Switches and Flags

This section of the input file contains switches and flags that tell FAST which modules should be used in the simulation.

CompElast: Compute structural dynamics [1 or 2]

1: Use ElastoDyn for the structural dynamics of the rotor, drivetrain, nacelle, tower, and platform
2: Use BeamDyn for the structural dynamics on the blades and ElastoDyn for the drivetrain, nacelle, tower, and platform

If **CompElast** is set to 2, the blade-related inputs and outputs from the ElastoDyn module are unused, replaced with those available in the BeamDyn module.

Please note that ElastoDyn must always be used when running FAST.

CompInflow: Compute inflow wind velocities [0, 1, or 2]

0: Use still air
1: Use InflowWind for inflow wind conditions
2: Use external wind conditions from OpenFOAM/SOWFA

In the normal FAST executable, setting **CompInflow** = 2 is not allowed.

CompAero: Compute aerodynamic loads [0 or 1]

0: Do not calculate aerodynamic loads
1: Use AeroDyn v14 for aerodynamic loads
2: Use AeroDyn v15 for aerodynamic loads

If **CompElast** is set to 2, **CompAero** must also be set to 2.

CompServo: Compute control and electrical-drive dynamics [0 or 1]

0: Do not calculate control and electrical-drive dynamics
1: Use ServoDyn for control and electrical-drive dynamics

CompHydro: Compute hydrodynamic loads [0 or 1]

0: Do not calculate hydrodynamic loads
1: Use HydroDyn for hydrodynamic loads

If **CompHydro** is not zero, FAST considers the model to be an offshore system. If **CompSub** is also non-zero, the offshore system is a fixed-bottom system. If **CompSub** is zero, the offshore system is considered a floating system.

CompSub: Compute sub-structural dynamics [0 or 1]

0: Do not calculate sub-structural dynamics
1: Use SubDyn for sub-structural dynamics

CompMooring: Compute mooring system [0, 1, 2, 3, or 4]

0: Do not model a mooring system
1: Use MAP++ to model a mooring system
2: Use FEAMooring to model a mooring system
3: Use MoorDyn to model a mooring system
4: Use OrcaFlexInterface to model a mooring system

If **CompMooring** is set to 4, **CompHydro** must be set to 0 and FAST considers the model to be an offshore floating system.

CompIce: Compute ice loads [0, 1, or 2]

0: Do not model offshore surface ice
1: Use IceFloe to model offshore surface ice
2: Use IceDyn to model offshore surface ice

If **CompIce** is not zero, both **CompHydro** and **CompSub** must be set to 1.

Input Files

The input files specified in this section of the primary FAST input file can be specified relative to the location of the FAST primary input file or specified with an absolute path. We recommend that you use quotes around the path/filenames.

EDFile: Name of file containing ElastoDyn input parameters [-]

This is the name of the ElastoDyn primary input file.

BDBldFile(1): Name of file containing BeamDyn input parameters for blade 1 [-]

This is the name of the BeamDyn primary input file for blade 1. It is not used if **CompElast** = 1.

BDBldFile(2): Name of file containing BeamDyn input parameters for blade 2 [-]

This is the name of the BeamDyn primary input file for blade 2. Different BeamDyn input files can be used between blades to model rotor structural imbalances. It is not used if **CompElast** = 1.

BDBldFile(3): Name of file containing BeamDyn input parameters for blade 3 [-]

This is the name of the BeamDyn primary input file for blade 3. Different BeamDyn input files can be used between blades to model rotor structural imbalances. It is not used if **CompElast** = 1 or for two-bladed rotors.

InflowFile: Name of file containing inflow wind input parameters [-]

This is the name of the InflowWind primary input file. It is used only if **CompInflow** = 1.

AeroFile: Name of file containing aerodynamic input parameters [-]

This is the name of the AeroDyn v14 (**CompAero** = 1) or AeroDyn v15 (**CompAero** = 2) primary input file. It is not used if **CompAero** = 0.

ServoFile: Name of file containing control and electrical-drive input parameters [-]

This is the name of the ServoDyn primary input file. It is not used if **CompServo** = 0.

HydroFile: Name of file containing hydrodynamic input parameters [-]

This is the name of the HydroDyn primary input file. It is not used if **CompHydro** = 0.

SubFile: Name of file containing sub-structural input parameters [-]

This is the name of the SubDyn primary input file. It is not used if **CompSub** = 0.

MooringFile: Name of file containing mooring system input parameters [-]

This is the name of the MAP++ (**CompMooring** = 1), FEAMooring (**CompMooring** = 2), MoorDyn (**CompMooring** = 3), or OrcaFlexInterface (**CompMooring** = 4) primary input file. It is not used if **CompMooring** = 0.

IceFile: Name of file containing ice input parameters [-]

This is the name of the IceFloe (**CompIce** = 1) or IceDyn (**CompIce** = 2) primary input file. It is not used if **CompIce** = 0.

Output

This section of the primary FAST input file deals with what can be output from a FAST simulation.

SumPrint: Print summary data to "<RootName>.sum" [T/F]

When set to "True", FAST will generate a file named "<RootName>.sum". This summary file contains the version number of all modules being used, the time steps for each module, and information about the channels being written to the time-marching output file(s). If **SumPrint** is "False", no summary file will be generated.

SttsTime: Amount of time between screen status messages [s]

During a FAST simulation, the program prints a message like this:

```
Timestep: 3 of 60 seconds.  Estimated final completion at 22:45:27.
```

SttsTime sets how frequently this message is updated. For example, if **SttsTime** is 2 seconds, you will see this message updated every 2 seconds of *simulated* time.

ChkptTime: Amount of time between creating checkpoint files for potential restart [s]

This input determines how frequently checkpoint files should be written. Checkpoint files are used for restart capability; we recommend that short simulations set **ChkptTime** to be larger than the simulation time, **TMax**. For more information on checkpoint files and restart capability in FAST, see sections "Checkpoint Files (Restart Capability)"and "Restart: Starting FAST from a checkpoint file" in this document. **ChkptTime** is ignored in the FAST-Simulink interface, and must be larger than **TMax** when using the FAST-OrcaFlex interface (**CompMooring** = 4).

DT_Out: Time step for tabular output [s]

This is the time step of the data in the tabular (time-marching) output files. **DT_Out** must be an integer multiple of **DT**. Alternatively, **DT_Out** can be entered as the string "default", which will set **DT_Out** = **DT**.

TStart: Time to begin tabular output [s]

This is the time step that must be reached before FAST will begin writing data in the tabular (time-marching) output files. Note that the output files may not actually start at **TStart** seconds if **TStart** is not an integer multiple of **DT_Out.**

OutFileFmt: Format for tabular output [1, 2, or 3]

This indicates which type of tabular (time-marching) output files will be generated. If **OutFileFmt** is 1, only a text file will be written. If **OutFileFmt** is 2, only a binary file will be written. If **OutFileFmt** is 3, both text and binary files will be written.

Text files write a line to the file each output time step. This can make the simulation run slower, but it can be useful for debugging, particularly if a simulation doesn't run to completion or if you want to look at some results before the entire simulation finishes.

Binary files are written in their entirety at the end of the simulation[‡‡]. If a lot of output channels are requested for a long simulation, this can take up a moderate amount of memory. However, they tend to run faster and the resulting files take up much less space. The binary files contain more precise output data than text files, which are limited by the chosen output format specifier—see **OutFmt** below.

We recommend you use text files for debugging and binary files for production work.

A MATLAB script for reading FAST binary output files is included in the archive (see <FAST8>/Utilities/SimulationToolbox/Utilities/ReadFASTbinary.m). The NREL post-processors Crunch and MCrunch can also read these binary files.

[‡‡] OrcaFlex and the user routines in ServoDyn may end the simulation abruptly when encountering errors. When this happens, FAST cannot write a binary file.

TabDelim: Use tab delimiters in text tabular output file? [T/F]

When **OutFileFmt** = 1 or 3, setting **TabDelim** to "True" will put tabs between columns in the text tabular output file. Otherwise, spaces will separate columns in the text tabular output file. If **OutFileFmt** = 2, **TabDelim** has no effect.

OutFmt: Format used for text tabular output, excluding the time channel [-]

When **OutFileFmt** = 1 or 3, FAST will use **OutFmt** to format the channels printed in the text tabular output file. **OutFmt** should result in a field that is 10 characters long (channel headers are 10 characters long, and NWTC post-processing software sometimes assume 10 characters). The time channel is printed using the "F10.4" format. We commonly specify **OutFmt** to be "ES10.3E2". If **OutFileFmt** = 2, **OutFmt** has no effect.

Checkpoint Files (Restart Capability)

For long FAST simulations that may not run to completion due to hardware failure or system availability, FAST has the ability to generate checkpoint files. These files can be used to restart the FAST simulation from the place the checkpoint file was written. See section, "Restart: Starting FAST from a checkpoint file" for a description on how to restart FAST from the checkpoint.

Checkpoint capability has not been added to the FAST-Simulink or FAST-OrcaFlex interfaces.

If you generate a checkpoint file, keep in mind the following caveats:

- Any Bladed-style DLL used for control must be modified for checkpoint/restart capability. We have made these modifications to the DLLs provided in the FAST archive:
    - When record 1 of the "DATA" (avrSwap) array is –8, the DLL should create a checkpoint file. The file must be named according to the file name passed in argument "INFILE" for this call. This file must contain all static data in the DLL that is necessary to start the DLL in the middle of the simulation.
    - When record 1 of the "DATA" (avrSwap) array is –9, the DLL should read the checkpoint file whose named is specified in the argument "INFILE". The data from this file should be used to set the values of any static variables contained in the DLL so that the simulation can continue from that point.
    - Source files to generate the Bladed-style DLL modified for this change are in the <FAST8>/CertTest/5MW_Baseline/ServoData/Source folder.
- **Any files that were open when the checkpoint file was created will not be open on restart.** We recommend you use only binary output files when starting from checkpoint files.
- The user-defined control routines are not available for checkpoint restart (i.e., CertTests 11-13 won't work).

Converting to FAST v8.12.x

You can find example up-to-date input files in the FAST v8.12.00a-bjj archive's CertTest folder. See the "Certification Tests" section of this document for descriptions.

Also note that we have created template input files for FAST and many of its modules. These template files can be found in the MATLAB Simulation Toolbox that is now included in the FAST archive: <FAST8>/Utilities/SimulationToolbox/ConvertFASTVersions/TemplateFiles.

See the "MATLAB Conversion Scripts" section below for help in automatically converting input files to the latest version.

Summary of Changes to Inputs

This section summarizes changes to the FAST input files between major releases.

Changes in FAST v8.12.00a-bjj

- The following differences occur in the FAST primary input file:
  - **ChkptTime** was added.
  - **CompElast** = 2, **BDBldFile(1)**, **BDBldFile(2)**, and **BDBldFile(3)** are now useable for enabling BeamDyn to model blade structural dynamics. The BeamDyn module has its own input files.
  - **CompInflow** and **InflowFile** were added. The InflowWind module now has its own input file (it is no longer part of the AeroDyn module).
  - **CompAero** = 2 is a new option, which allows the user to choose AeroDyn v15. The AeroDyn v15 module has its own input files, entirely different from AeroDyn v14.
  - **CompMooring** = 4 is a new option, which allows the user to choose the OrcaFlex interface. The OrcaFlex interface has its own input file.
  - Unused variables **CompUserPtfmLd** and **CompUserTwrLd** were removed.
- The following differences occur in the input files of the ElastoDyn module:
  - **BldNodes** was added to the ElastoDyn primary input file, used to discretize the blade into **BldNodes** equally spaced elements when AeroDyn v14 is not used (when using AeroDyn v14, ElastoDyn will use the AeroDyn v14 blade discretization instead). This number does not include nodes located at the blade root or blade tip, which are added internally.
  - **PitchAxis** in the ElastoDyn blade input file is now used only in conjunction with AeroDyn v14. The specification of aerodynamic center in AeroDyn v15 replaces the need for **PitchAxis**.
- The following differences occur in the primary input file of the AeroDyn v14 module:
  - The unused **SI** variable input line has been removed.
  - **HubHt** has been removed, effectively replaced by **RefHt** in the InflowWind module's primary input file.
  - **WindFile** has been removed, effectively replaced by inputs in the InflowWind module's primary input file.
  - One additional header line was added.
- The following differences occur in the primary input file of the ServoDyn module:
  - **DLL_InFile** was added, referring to the name of the input file that gets passed to the Bladed-style DLL when it is used. The DLL may or may not use this input file. Previously, this value was hardcoded to "DISCON.IN" in the ServoDyn source code.

- **DLL_DT** was added, referring to the time step (in seconds) at which the Bladed-style DLL is called. This value must be an integer multiple of the ServoDyn module's **DT** value. ServoDyn will use old values until the DLL is called again. Previously, the Bladed-style DLL was called at the same rate as the ServoDyn module.
- **DLL_Ramp** was added, which is a flag that determines if ServoDyn should use a linear ramp to interpolate the blade-pitch command from the Bladed-style DLL between **DLL_DT** time steps (instead of a step function). Setting this value to "True" will result in a time shift of **DLL_DT** for the blade-pitch command.
- **BPCutoff** was added, referring to the cutoff frequency (in Hz) for a first-order low-pass filter applied to the blade-pitch command from the Bladed-style DLL. Setting a large number will effectively eliminate the filter, if undesired.

- The following differences occur in the primary input file of the HydroDyn module; while no lines have been added or removed from the file, the following inputs were changed:
  - **WaveMod** = 5 and 6 are new options for using externally generated wave data; related, **GHWvFile** has been replaced with **WvKinFile**.
  - **HasWAMIT** has been replaced with the **PotMod** switch, **WAMITFile** has been replaced with **PotFile**, and **PropWAMIT** has been replaced with the **PropPot** flag for enabling potential-flow theory.

Changes in FAST v8.10.00a-bjj

The primary input file of FAST v8.10.00a-bjj is the same as that of FAST v8.09.00a-bjj, however the code accepts an additional option for **CompMooring**. The input files for the MAP++ and ServoDyn modules have been modified in this version.

- The following differences occur in the primary input file of the ServoDyn module:
  - A tuned-mass damper section has been added. A new comment line and two additional inputs for the nacelle tuned-mass damper have been added: **CompNTMD** and **NTMDfile**.
  - Switches for control modes **PCMode**, **VSContrl**, **GenModel**, **HSSBrMode**, and **YCMode** have been standardized to the following (note that not all switches are valid options for each control modes; see comments in sample input files):

Control Mode	Description
0	None (no control)
1	Simple built-in model
2	Advanced built-in model
3	User-defined from user routine
4	User-defined from Simulink/LabVIEW[§§]
5	User-defined from Bladed-style DLL

[§§] Note that the LabVIEW interface for FAST v8 has not yet been developed.

- The HSSBrTq output from ServoDyn has been renamed HSSBrTqC (high speed shaft torque command). This output was renamed to avoid confusion with the new HSSBrTq output available from ElastoDyn.

- The MAP++ input file uses newtons (N) instead of kilonewtons (kN) and the flags have changed. The new flags are listed on the MAP++ web page.

Changes in FAST v8.09.00a-bjj

The primary input file of FAST v8.09.00a-bjj is the same as that of FAST v8.08.00c-bjj. However, the HydroDyn input file in this new version has one less line that the previous version.

Changes in FAST v8.08.00c-bjj

The following list describes the differences in the primary input file of FAST v8.08.00c-bjj relative to FAST v8.03.02b-bjj.

- Many variables in the primary FAST input file have been renamed:

FAST v8.03.x Name	FAST v8.08.x Name
ADFile	AeroFile
SrvDFile	ServoFile
HDFile	HydroFile
SDFile	SubFile
MAPFile	MooringFile
CompMAP	CompMooring

- Most of the feature *flags* were changed to feature *switches* in the primary FAST input file. Instead of True/False inputs, CompAero, CompServo, CompHydro, CompSub, and CompMooring now require integer inputs.
- Inputs for coupling with modules IceFloe and IceDyn have been added: **CompIce** and **IceFile**.
- Several new inputs for future coupling of BeamDyn into FAST have been added. These inputs are **CompElast**, **BDBldFile(1)**, **BDBldFile(2)**, and **BDBldFile(3)**.
- The modules ElastoDyn, ServoDyn, AeroDyn, HydroDyn, and SubDyn allow users to input the string “Default” for their respective time steps, which will then use the time step from the FAST primary input file.

Changes in FAST v8.03.02b-bjj

The following list describes the differences in the FAST, ElastoDyn, and ServoDyn input files of FAST v8.03.02b-bjj relative to the input files of FAST v7.02.00d-bjj.

- The primary FAST input file has been converted to primary input files for FAST, ElastoDyn, and ServoDyn and some of the inputs have been reordered.
- The FAST Platform file has been eliminated, with some of the inputs now part of the ElastoDyn primary input file and some of the inputs now part of HydroDyn’s and MAP’s input files.
- All of the inputs formerly labeled “[CURRENTLY IGNORED]” have been removed.
- Switches for ADAMS preprocessing and linearization have been removed.
- Noise has been removed.

- **PtfmLdMod** has been converted to **CompUsrPtfmLd**.
- **TwrLdMod** has been converted to **CompUserTwrLd**.
- The tip-brake inputs have been removed.
- **PtfmCM** is now **PtfmCMzt**, with **PtfmCMzt** = -**PtfmCM**.
- Corresponding inputs **PtfmCMxt** and **PtfmCMyt** have been added.
- **PtfmRef** is now **PtfmRefzt**, with **PtfmRefzt** = -**PtfmRef**.
- **TwrRBHt** and **TwrDraft** have been replaced with **TowerBsHt**, with **TowerBsHt** = **TwrRBHt** – **TwrDraft**.
- The output decimation factor (**DecFact**) has been converted to **DT_out** (**DT_out** = **DT***DecFact**).
- The yaw and pitch maneuvers no longer specify end times for the maneuvers. Instead they specify a rate for the maneuver.
- The **GBRevers** variable has been removed; input **GBRatio** must now be specified as a negative number if **GBRevers** was previously set to True.
- ElastoDyn's blade input properties table no longer specifies **AeroCent**. Instead, it specifies the location of the pitch axis, **PitchAxis**, which is calculated as **PitchAxis** = 0.5 – **AeroCent** by the MATLAB conversion script; the aerodynamic center will become part of AeroDyn in a future release.
- The **OutList** variables have been divided among the various FAST modules, and several outputs are no longer valid.
- The GH Bladed Interface is now a standard option in ServoDyn, without requiring a recompile.
- Tower drag loading has been added to AeroDyn v14.02.00c-mlb with a new corresponding flag in the AeroDyn input file.
- The glue code allows options for **AbortErrLevel**, number of corrections in the predictor-corrector algorithm, and extrapolation/interpolation order of module inputs to be used for time advancement.
- For the ElastoDyn coupling to HydroDyn or SubDyn, FAST also has two inputs controlling the implicit solve (via Jacobian computed with finite differences)—see **DT_UJac** and **UJacSclFact** described in the Variables Specified in the FAST Primary Input File Section above.

MATLAB Conversion Scripts

Because the changes to the input files are significant, we have created MATLAB scripts to automatically convert FAST v7.x or FAST v8.x. The files you will need are included in the Simulation Toolbox, located in this directory of the FAST archive: <FAST8>/Utilities/SimulationToolbox/ConvertFASTVersions.

We recommend that you add the Simulation Toolbox to your MATLAB path so that you can access all of the routines defined in it. For example:

```
FASTSimulationToolbox = 'C:\Users\bjonkman\FAST\UtilityCodes\SimulationToolbox';
addpath( genpath( FASTSimulationToolbox ) );
```

An example showing how we converted the NREL CertTest input files for use with FAST v8.12.00a-bjj is included in the FAST archive:<FAST8>/CertTest/ConvertFiles.m. You can use this script as a basis for helping to convert your own input files; however, we *strongly* recommend that you make copies of all

your input files before running any scripts to convert them. Fortran and MATLAB read text input files in slightly different ways; so some things that may work in Fortran may not be read in the same way in MATLAB. We recommend that you carefully check the files after converting them.

Converting from previous versions of FAST v8

Each release of FAST v8 comes with conversion scripts to help convert from the previous released version. To convert to the latest version, you may have to run multiple MATLAB scripts.

Convert from FAST v8.03.02b-bjj to FAST v8.08.00c-bjj

If you have FAST v8.03.x input files that you wish to convert, you can use the MATLAB function, **ConvertFAST8_3to8**. This function will convert the primary FAST input file *and the primary HydroDyn input file* from FAST v8.03.x to FAST v8.08.x. You will need to provide the name of the FAST v8.03.x primary input file and the name of the directory where the new input files should be placed:

```
ConvertFAST8_3to8( inputfile, newDir );
```

Note that we do not automatically convert the SubDyn input files from FAST v8.03. These you must do by hand—or use the models provided in the FAST Certification Tests.

Convert from FAST v8.08.00c-bjj to FAST v8.09.00a-bjj

You can call **ConvertFAST8_8to9** to convert the FAST v8.08.x files for use with FAST v8.09.x:

```
ConvertFAST8_8to9( inputfile, newDir );
```

Convert from FAST v8.09.00a-bjj to FAST v8.10.00a-bjj

You can call **ConvertFAST8_9to10** to convert the FAST v8.09.x files for use with FAST v8.10.x:

```
ConvertFAST8_9to10( inputfile, newDir );
```

This script *does not* change the MAP input files, and it does not rename ServoDyn's old "HSSBrTq" output to "HSSBrTqC". You will have to make those changes by hand.

Convert from FAST v8.10.00a-bjj to FAST v8.12.00a-bjj

You can call **ConvertFAST8_10to12** to convert the FAST v8.10.x files for use with FAST v8.12.x:

```
ConvertFAST8_10to12( inputfile, newDir );
```

This script attempts to modify the HydroDyn and AeroDyn v14 input files and create new InflowWind files; however, due to some unique features of AeroDyn v14's input file, some changes may have to be made by hand (particularly if the file uses multiple Reynolds numbers.). Also, there are some instances when the script fails to identify the wind file type (e.g. when a file has a .wnd extension but isn't a full-field wind file); these files may have to be modified by hand.

If you would like to convert the AeroDyn v14 input files to AeroDyn v15, you can call the script with an optional logical argument:

```
ConvertFAST8_10to12( inputfile, newDir, true);
```

However, it should be noted that this script does not convert the AeroDyn v14 airfoil files to the format necessary for AeroDyn v15.

The script does not attempt to create AeroDyn v15 airfoil, BeamDyn, or OrcaFlexInterface input files.

Converting from FAST v7

To convert FAST v7 input files to FAST v8.12.00a-bjj, you can use the MATLAB function **ConvertFAST7to8**. This function will update the AeroDyn input file to v14 and create new primary input files for FAST, ElastoDyn, InflowWind, and ServoDyn. It also creates new blade and tower files for ElastoDyn. *It does not automatically convert hydrodynamic-, substructure- and mooring system-related inputs and does not create BeamDyn or ice-related input files*.

You will need to provide the **ConvertFAST7to8** function with the name of the FAST v7.x primary input file and the directory where the new input files should be placed. *The new directory should not be the directory where the old files are located!*

```
ConvertFAST7to8(inputfile,newDir);
```

If your input file has pitch or yaw maneuvers, you may also provide the routine with the new rates (instead of the end times previously used). We have also provided a MATLAB routine (**CalculateYawAndPitchRates**) that will calculate these rates, but you must provide the routine the name of the FAST output file that contains the previous results of the Pitch and/or Yaw channels.

```
[YawManRat, PitManRat] = CalculateYawAndPitchRates('Test09.fst', 'Test09.out');
ConvertFAST7to8(inputfile, newDir, YawManRat, PitManRat);
```

If **YawManRat** or **PitManRat** are zero, those inputs are ignored and values from the ServoDyn template file will be used instead.

If your input file was used with the custom interface to Bladed-style DLL controllers, you should also set the optional input parameter, usedBladedDLL, so that your input switches that previously called the DLL are now set to 5, the new switch for User-Defined Control from Bladed DLL.

```
ConvertFAST7to8(oldFSTName, newDir, YawManRat, PitManRat, usedBladedDLL)
```

Running FAST

FAST v8.12.00a-bjj must load the MAP++ library when the program starts. On Windows®systems, FAST_Win32.exe needs to load MAP_Win32.dll and FAST_x64.exe needs to load MAP_x64.dll. These dynamic link libraries (DLLs) must be on your Windows® path. The easiest way to do this is to make sure that the MAP DLLs are in the same directory as the FAST executables. We distribute the executables and DLLs in the bin directory of the FAST archive, so this is already done for you. However, if you choose to move the files or if you compile the code yourself, you may have to modify your path environment variable or move some files.

On Mac OS or Linux, you will need to compile FAST and the MAP++ library. Compiling instructions and tips for running on non-Windows® systems are included in the FAST archive in the <FAST8>/Compiling folder.

Normal Simulation: Starting FAST from an input file

To run FAST from a Windows command prompt, the syntax is:

```
<name of FAST executable with or without extension> <name of input file with extension>
```

To start, it easiest to open up your command window in the directory in which your FAST primary input file and FAST executable are stored. For example, if you have an input file named "Input.fst", along with "FAST_Win32.exe", stored in "C:\FileLocation", you should type:

```
C:\>cd FileLocation
C:\FileLocation> FAST_Win32 Input.fst
```

The syntax is the same for different input files. Simply change "Input.fst" to whatever input file you want.

An installation guide is available that describes how to install FAST (and the other computer-aided engineering (CAE) tools) in such a way that they will run from a command window from any folder (without moving or copying the executable around to different folders). See: Installing NWTC CAE Tools on PCs Running Windows®.

Restart: Starting FAST from a checkpoint file

If FAST generated a checkpoint file (see section, "Checkpoint Files (Restart Capability)"), the simulation may be restarted using this syntax from a Windows® command prompt:

```
<name of FAST executable > -restart <root name of checkpoint file without extension>
```

Note that the syntax for restart uses the checkpoint file **root** name instead of the full checkpoint **file** name. For example, to restart a simulation of Test18.fst at step 1800, you would type

```
C:\FileLocation> FAST_Win32 -restart Test18.1800
```

FAST would then read the file "Test18.1800.chkp" and—because this test uses a Bladed-style DLL controller—it would also read the file "Test18.1800.dll.chkp"

On restart, FAST does not read the input files again. However, if a Bladed-style DLL is used for control, the DLL will be reloaded on restart using the stored name of the DLL file. Thus, if the stored name is a relative path and you are not in the same directory, it will likely fail.

It is recommended that FAST be restarted from the same directory in which it was originally run.

Modeling Tips

If a model is numerically unstable, you can try these steps

- Add a correction step (**NumCrctn**).

- Make **DT** smaller.
- Change **InterpOrder**.
- Use the recommend value for **UJacSclFact.**
- Set better initial conditions in the module input files (particularly ElastoDyn).
- Simplify models (e.g. disable modules and eliminate DOFs) to debug problems.

Some of models (e.g., the OC4 Jacket CertTest) require more than 2GB of memory and may not run on 32-bit Windows® systems. The model does run using FAST_Win32.exe on a 64-bit Windows® system.

For simulations involving BeamDyn, you may want to recompile in double precision for better results.

Certification Tests

Table 3 lists the tests (1-26) and models available in the FAST CertTest folder:

Table 3: Certification Tests Distributed with FAST v8.12.00a-bjj

Test Name	Turbine Name	No. Blades (-)	Rotor Diameter (m)	Rated Power (kW)	Test Description
Test01	AWT-27CR2	2	27	175	Flexible, fixed yaw error, steady wind
Test02	AWT-27CR2	2	27	175	Flexible, steady wind, high-speed shaft brake shutdown
Test03	AWT-27CR2	2	27	175	Flexible, free yaw, steady wind
Test04	AWT-27CR2	2	27	175	Flexible, free yaw, turbulence
Test05	AWT-27CR2	2	27	175	Flexible, steady wind
Test06	AOC-15/50	3	15	50	Flexible, steady wind
Test07	AOC-15/50	3	15	50	Flexible, free yaw, turbulence
Test08	AOC-15/50	3	15	50	Flexible, fixed yaw error, steady wind
Test09	UAE VI downwind	2	10	20	Flexible, yaw ramp, steady wind
Test10	UAE VI upwind	2	10	20	Rigid, power curve, ramp wind
Test11	WP 1.5 MW	3	70	1500	Flexible, variable speed & pitch control, pitch failure, turbulence
Test12	WP 1.5 MW	3	70	1500	Flexible, variable speed & pitch control, ECD event
Test13	WP 1.5 MW	3	70	1500	Flexible, variable speed & pitch control, turbulence
Test14	*Not currently available*				
Test15	SWRT	3	5.8	10	Flexible, variable speed control, free yaw, EOG01 event
Test16	SWRT	3	5.8	10	Flexible, variable speed control, free yaw, EDC01 event
Test17	SWRT	3	5.8	10	Flexible, variable speed control, free yaw, turbulence
Test18	NREL 5 MW - Land-based	3	126	5000	Flexible, DLL control, tower potential flow and drag, turbulence
Test19	NREL 5 MW - OC3-Monopile	3	126	5000	Flexible, DLL control, tower potential flow, turbulence, irregular waves
Test20	NREL 5 MW - OC3-Tripod	3	126	5000	Flexible, DLL control, tower potential flow, steady wind, regular waves with 0 phase
Test21	NREL 5 MW - OC4-Jacket	3	126	5000	Flexible, DLL control, tower potential flow, turbulence, irregular waves, marine growth
Test22	NREL 5 MW - ITI Barge	3	126	5000	Flexible, DLL control, irregular & multidirectional waves, turbulence
Test23	NREL 5 MW - MIT/NREL TLP	3	126	5000	Flexible, DLL control, turbulence, irregular waves

Test Name	Turbine Name	No. Blades (-)	Rotor Diameter (m)	Rated Power (kW)	Test Description
Test24	NREL 5 MW - OC3-Hywind	3	126	5000	Flexible, DLL control, turbulence, irregular waves
Test25	NREL 5 MW - OC4-DeepCwind Semi-Submersible	3	126	5000	Shortened OC4 Load Case (LC) 3.7: steady wind, white noise second-order waves
Test26	NREL 5 MW - Land-based, with BeamDyn	3	126	5000	The same as Test 18, but using BeamDyn instead of ElastoDyn for the blade dynamics

Compiling FAST

If you want to compile the code, please read the file, "CompilingInstructions_FASTv8.pdf" located in the "Compiling" folder in the FAST archive for details. **Note that if you are running on Windows® and are not changing source code, you should not have to recompile the code.** Use the 32- or 64-bit Windows® binaries distributed in the <FAST8>/bin folder.

Unlike FAST v7 distributions, all of the source code you need to compile the project is contained in the archive's Source directory. The text files (.txt) in the source folders are input files for the FAST Registry. These files are used to generate the *_Types.f90 files for the component modules.

The compiling folder contains:

- A Microsoft Visual Studio 2010 project with all of the FAST source files and settings needed to compile in Release or Debug mode. This project places executables called "FAST_dev_Win32.exe" and "FAST_dev_debug_Win32.exe" in the <FAST8>/bin folder.[***] (renamed with "_dev" to prevent overwriting the executables created by NREL and distributed with FAST).
- A Windows® batch script that can be run from your Intel Fortran Command Prompt Window, with very little (if any) modification. This batch file creates executables named "FAST_iwin32.exe" and "FAST_iwin64.exe" in the <FAST8>/bin folder. We recommend you use the Visual Studio project instead of this batch script.
- A *makefile* for gfortran with settings to create a FAST executable in the <FAST8>/bin folder. To run this on Windows, you will need to install binaries of the LAPACK libraries. Mac OS and Linux users will also have to obtain the MAP.so library and Registry executable program to use with this *makefile*. Please see "CompilingInstructions_FASTv8.pdf" for details.

All of these tools for compiling are set up to compile and link with the appropriate settings, though you may have to modify the *makefile* to find the LAPACK libraries. (Note that FAST uses several specialized compiling/linking options and that MAP++ is distributed as a dynamic-link library.) These tools also run the FAST Registry if necessary (e.g., changes to the Registry input files or missing *_Types.f90 files).

[***] If you are using a 2015 or later version of Intel Fortran, the executable may be generated in the <FAST8>/Compiling/VisualStudio folder instead, due to differences in Visual Studio integrations.

FAST v8 Interface to Simulink

FAST v8 has also been implemented as a library that can be called from a Simulink S-Function block, using predefined inputs from Simulink. Simulink is a popular simulation tool for controls design that is distributed by The Mathworks, Inc. in conjunction with MATLAB.

Major Changes Between the FAST v7 and v8 Interfaces to Simulink

For people familiar with the FAST v7 interface to Simulink, the following differences should be noted for the FAST v8 interface to Simulink:

- Simulink no longer integrates the FAST structural states, which are now integrated in the FAST library. See the section "S-Function States" below for further information.
- MATLAB no longer reads the FAST input file, so Read_FAST_Input.m and Simsetup.m are no longer part of the FAST archive.
- FAST_SFunc no longer reads variables from the MATLAB workspace.
- The blade and tower initial conditions (ElastoDyn's **IPDefl**, **OopDefl**, **TTDspFA**, and **TTDspSS** variables) need not be zero.
- ServoDyn's **TPCOn**, **TYCOn**, and **TimGenOn** variables need not be zero.
- ServoDyn's **TimGenOf** variable need not be larger than **TMax**.
- ServoDyn's **GenTiStr** and **GenTiStp** variables need not be set to "True".
- The fraction of the maximum high-speed shaft braking torque has been added as an input to FAST_SFunc from Simulink.

Definition of the FAST v8 Interface to Simulink

The FAST v8 interface to Simulink is implemented as a Level-2 S-Function called FAST_SFunc. The interface is written in C, and it calls a DLL of FAST v8 routines, which are written in Fortran. Both 32- and 64-bit Windows® versions are supported, although the libraries must have the same addressing scheme (both addressing schemes can be used on 64-bit operating systems, but only the 32-bit version can be used on a 32-bit operating system). The calling sequence between the libraries is illustrated in Figure 5.

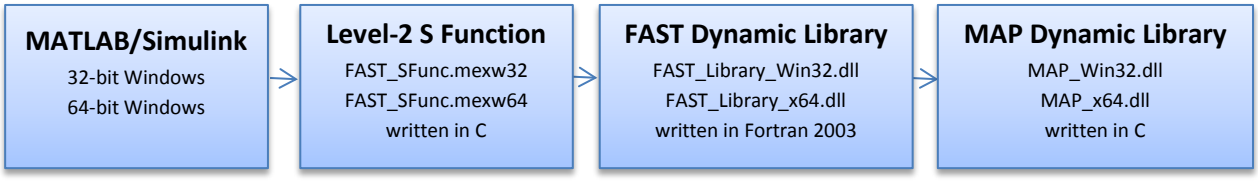

Figure 5: Libraries in the FAST - Simulink Interface

Please note that because this interface uses static variables, there can be only one instance of the FAST_SFunc mex file in any instance of MATLAB (i.e., you cannot run two different models simultaneously).

S-Function Parameters

The FAST_SFunc S-function block is designed to accept exactly three parameters from Simulink. The values for each of the parameters are required at initialization, and they must not be changed during the simulation.

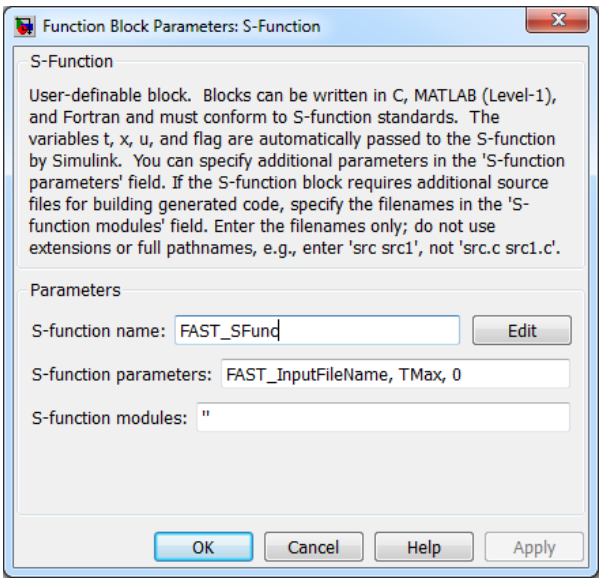

Figure 6: FAST_SFunc Block Parameters

FAST_InputFileName

The first parameter is a string, which contains the name of the FAST v8 primary input file. In the sample models, this string is contained in a variable called **FAST_InputFileName**.

TMax

The second parameter is a double-precision real value called **TMax**. This **TMax** is used in place of the **TMax** specified in the FAST v8 primary input file, except for the following two cases:

- If **TMax** in the FAST v8 primary input file is larger than the one in Simulink, the **TMax** from the primary input file will be used by the FAST modules (e.g., HydroDyn's Waves submodule).
- If **TMax** in the FAST v8 primary input file is larger than the one in Simulink, the **TMax** from the primary input file will be used to allocate space for the binary output file (if **OutFileFmt** /= 1).

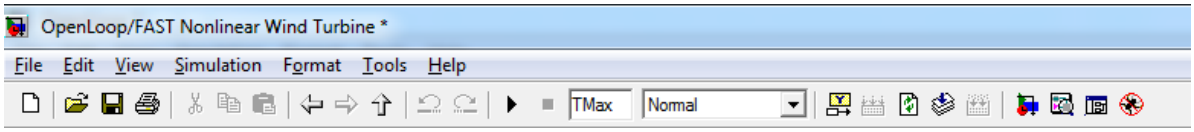

Figure 7: Using TMax to specify simulation end time in Simulink

NumAdditionalInputs

The third parameter sent to the S-Function block is **NumAdditionalInputs**. Currently, **NumAdditionalInputs** is 0 for most cases.[†††]

S-Function Inputs

The inputs to the FAST S-Function are values in an array of size 8 + **NumAdditionalInputs**. (See section "S-Function Parameters" for an explanation of **NumAdditionalInputs**.)

The values in the input array are as follows:

1) Generator torque (Nm)
2) Electrical power (W)
3) Commanded yaw position (radians)
4) Commanded yaw rate (radians/s)
5) Commanded pitch for blade 1 (radians)
6) Commanded pitch for blade 2 (radians)
7) Commanded pitch for blade 3 (radians): note that this input is unused on 2-bladed turbines, but is always required as input to the DLL
8) Fraction of maximum high-speed shaft braking torque (fractional value between 0 and 1)

Note that these inputs are passed from Simulink to ServoDyn. However, these inputs are used *only if* the appropriate switches in the ServoDyn input file are selected. The **PCMode**, **VSContrl**, **YCMode***,* and **HSSBrMode** parameters in the ServoDyn input file must be set to "4" to allow the inputs from Simulink to be used for pitch control, variable-speed control, nacelle yaw control, and/or high-speed shaft braking respectively.

[†††] For flexibility reasons (i.e., so that FAST_SFunc.c doesn't have to be recompiled for customizations to FAST_Library.f90), this third parameter can also be input as an array with no more than 11 entries. If this parameter is an array, the first element of the array is **NumAdditionalInputs**.

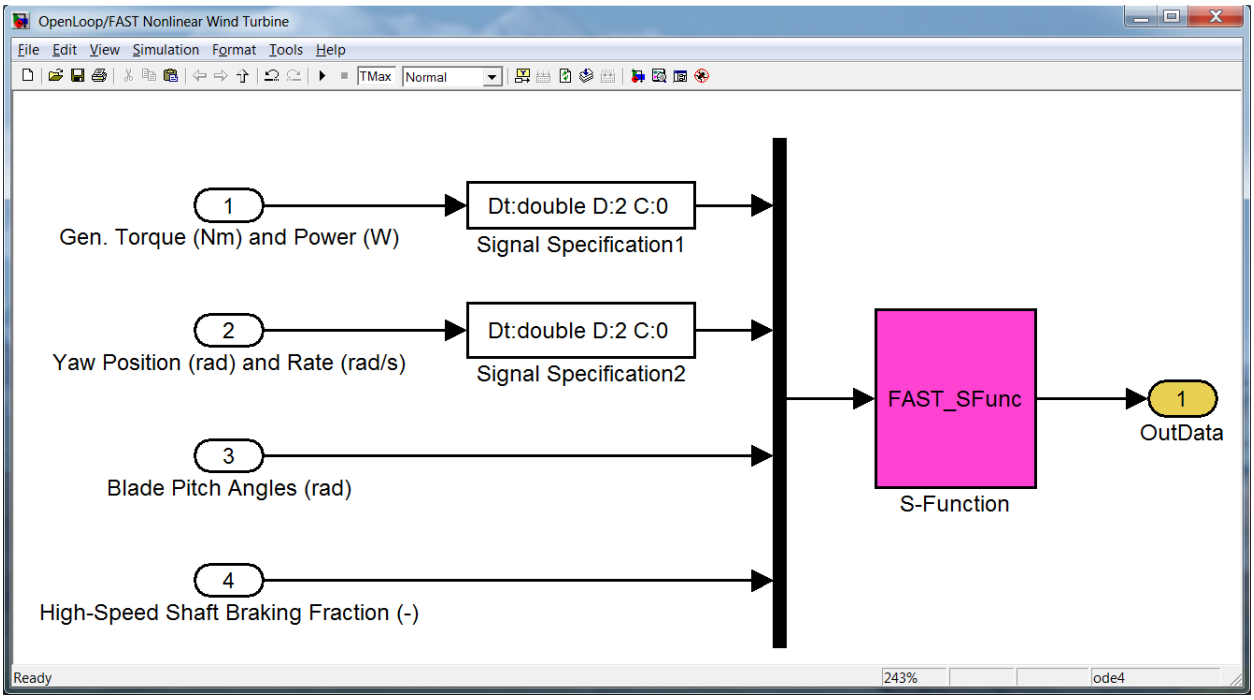

Figure 8: FAST v8 Nonlinear Wind Turbine Block in Simulink

S-Function Outputs

The outputs from FAST to Simulink are the values that are written to the FAST output file(s). The values are output to Simulink every time the S-Function is called, and thus, are not affected by FAST's **DT_Out** parameter. These outputs are the channels defined in module **OutList** variables; these channel names and units are also written in the FAST summary file. At the FAST_SFunc block initialization, FAST_SFunc writes a cell array called "OutList" containing the names of these output channels to the MATLAB base workspace. Currently, there is a limit of 1000 outputs from FAST when used with the Simulink Interface.

S-Function States

The FAST v8 S-Function does not register any states with Simulink. This effectively makes the block a discrete-time system which is solved inside the S-Function block. Whereas the FAST v7 Simulink models frequently required a time-delay block to eliminate algebraic loops to enable the system to solve, FAST v8 Simulink models should not need to implement a time-delay block.

Converting FAST v7 Simulink Models to FAST v8

Convert FAST input file(s)

Any FAST v7 input files must be converted to files for FAST v8 prior to using the FAST v8 Interface to Simulink. See the section "Converting to FAST v8.12.x" for help with this conversion.

Edit Simulink model

- Remove state integrations from the FAST Nonlinear Wind Turbine block.
- Remove time delay (if it exists).
- Add appropriate parameters to the FAST_SFunc block (see "S-Function Parameters").

- Change pitch controller to input three values instead of **NumBl** values.
- Add an input for the high-speed shaft brake fraction.
- There are no continuous states in the FAST_SFunc, so the solver may be changed to FixedStepDiscrete.
- If the Simulink model previously used states (*q*, *qdot*) from FAST v7, choose the appropriate state outputs from ElastoDyn instead (see OutListParameters.xlsx in the FAST archive).
- Replace any previously used output channels that don't exist in FAST v8.

Initialization

The Simulink interface for FAST v8 does not read *any* variables defined in the MATLAB workspace, which means the SimSetup.m and Read_FAST_Input.m files used by the Simulink interface for FAST v7 do not exist in v8. All values required for the DLL to run are passed directly through S-Function parameters (see section "S-Function Parameters") and S-Function inputs (section "S-Function Inputs").

Note that the Simulink interface for FAST v8 does not use the initial conditions from Simulink; all initial condition data comes from the FAST library.

After FAST v8 is initialized, it places two variables, **OutList**, and **DT** in the MATLAB workspace. **OutList** is a cell array of channel names corresponding to the FAST output channels, and **DT** is a double-precision real value storing the time step from the FAST v8 primary input file that may be used in the Simulink solver. (Note: **DT** is the sample rate that FAST_SFunc is called. It need not be the same value as in the Simulink solver, but Simulink requires that **DT** be an integer multiple of the Simulink solver's fundamental sample time.)

Depending on the model, you may have to set **DT** and **OutList** prior to running your Simulink model. The two sample models included in the FAST archive do not require this step, but it may be required for more complicated models; this is because the FAST_SFunc block may not be initialized before Simulink checks whether other blocks are valid. **OutList** and **DT** will be overwritten before FAST_SFunc calculates any output, so they do not necessarily have to be correct when initialized; they just have to exist so that the rest of the model can be evaluated. ***Note, however, that we have noticed some models require*** *OutList* ***to be exact before calling FAST_SFunc at initialization; the expression blocks are not getting completely reevaluated during the model's execution (when it reevaluates the expression u(strcmp('VarName', OutList)), u changes but the expression for the index, strcmp('VarName', OutList), is not reevaluated.***

Output Files

Output files from FAST v8 for Simulink are named <RootName>.SFunc.<ext>. This is in contrast to files from FAST v7, which were named <RootName>_SFunc.<ext>. Note that the extensions in FAST v8 have changed from what was used in FAST v7 (see section "FAST v8 Input and Output Files").

Running FAST in Simulink

To run the FAST S-Function from Simulink, MATLAB must be able to find the appropriate DLLs. This includes the FAST_SFunc.mex*, FAST_Library_*.dll, and MAP_*.dll files. All of these files are contained

in the FAST archive's bin directory, so the easiest way to do this is to add the <FAST8>/bin directory to the MATLAB path.

Sample Simulink Models for FAST v8

Two sample models for running FAST v8 with Simulink are provided in the FAST archive (see the <FAST8>/Simulink/Samples folder). These examples are intended to help the user understand how to use the FAST_SFunc block. It assumed that the user is already somewhat familiar with the Simulink environment.

OpenLoop

The OpenLoop sample model contains the FAST S-Function block and constant open-loop control input blocks.

The Run_OpenLoop.m script in the <FAST8>/Simulink/Samples folder allows the user to run all of the FAST Certification Tests from Simulink using the OpenLoop model without using any of the control inputs from Simulink.

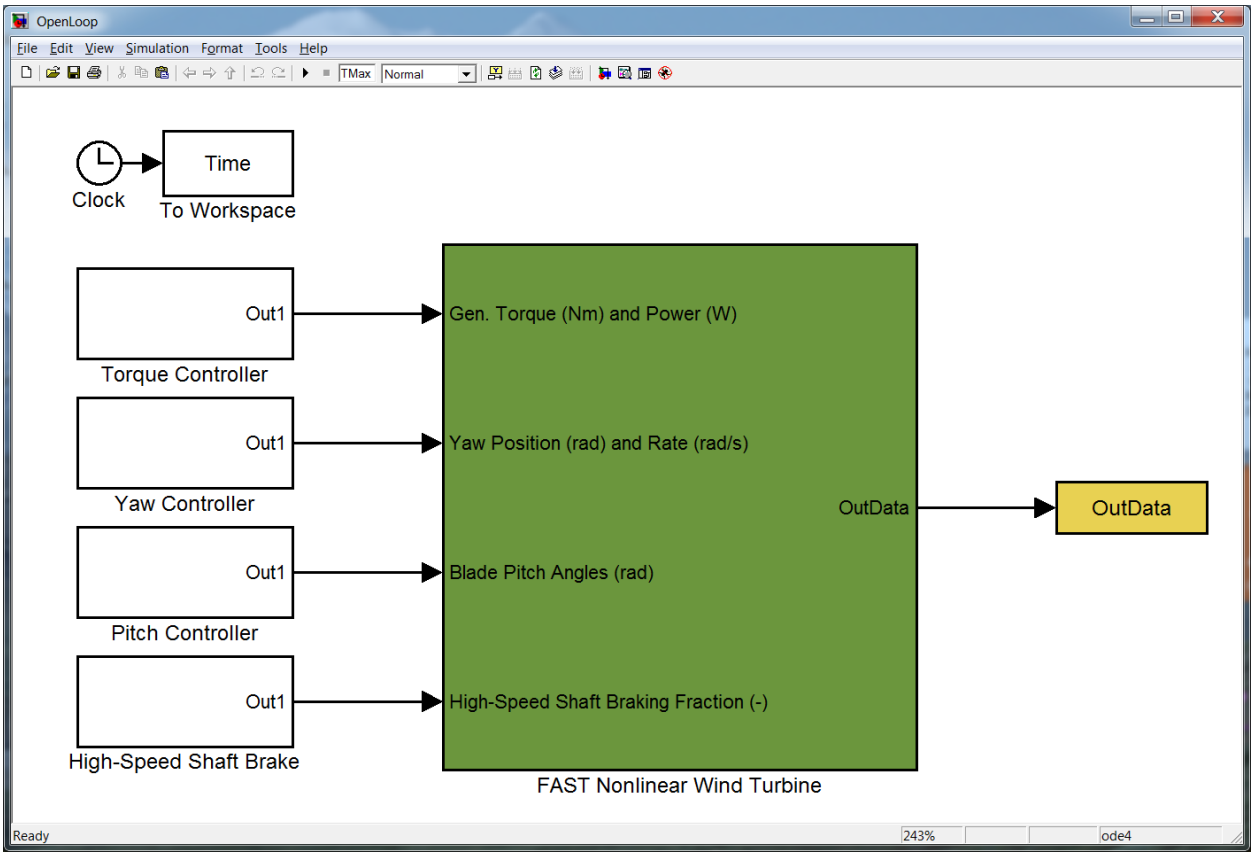

Figure 9: OpenLoop.mdl Sample Model for Simulink

Test01_SIG

The Test01_SIG.mdl file contains the FAST S-Function block and the simple induction generator model for FAST certification test #01 implemented within Simulink. To run this model, change *VSControl* to "4"

in the <FAST8>/CertTest/AWT27/Test01_ServoDyn.dat file, then run the Run_Test01_SIG.m file in the <FAST8>/Simulink/Samples folder.

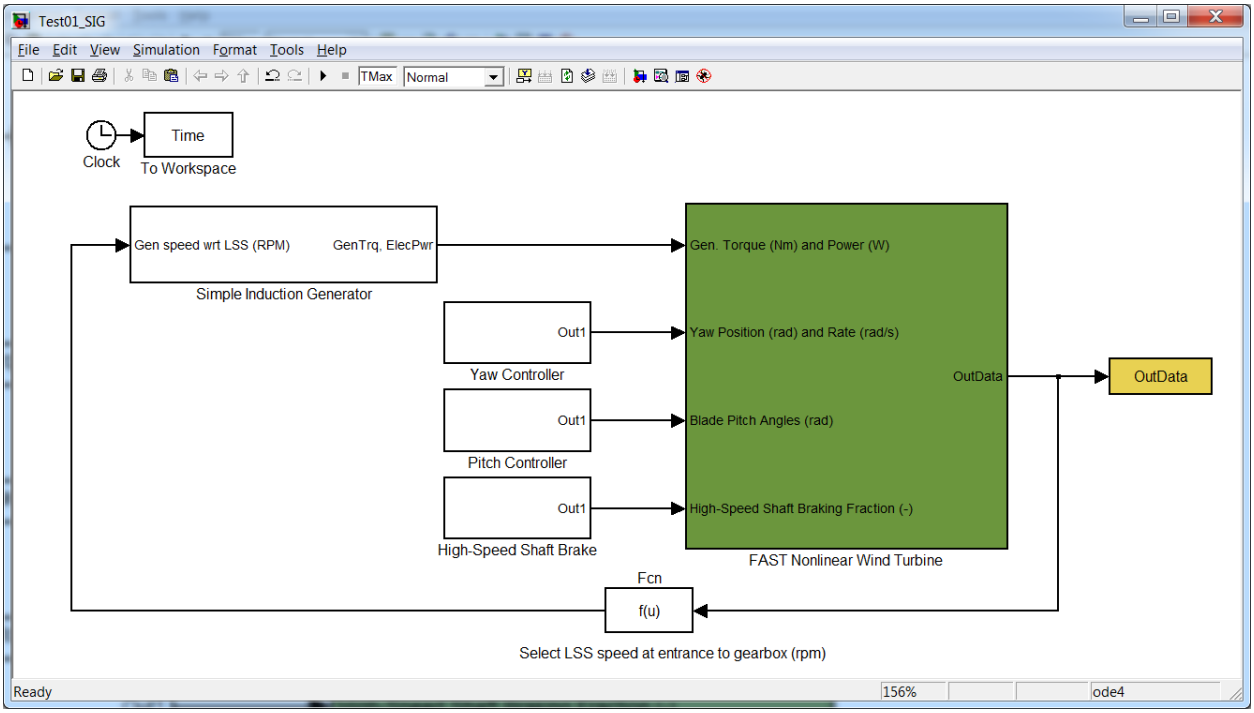

Figure 10: Test01_SIG.mdl Sample Model for Simulink

Error Messages

If your Simulink model fails to run, please make note of any error, warning, or informational windows that open. Also make sure to look at any text written to the MATLAB Command Window, which is where all messages from the FAST_Library_*.dll file will be written.

Compiling FAST for Simulink

The S-Function (mex) files contained in the FAST v8 archive were compiled with MATLAB R2014b. If you are using a different version of MATLAB, you may have to compile FAST_SFunc, but you should not have to recompile the FAST_Library dll.

FAST_Library

The FAST archive contains a sample Visual Studio Intel Fortran project (2010) that is set up to compile a dll called FAST_Library_{Win32 | x64}.dll and place it in the <FAST8>/bin folder. This project is located in the <FAST8>/Simulink/VisualStudio folder.

Compiling the FAST library is very similar to compiling the stand-alone version of FAST v8, which is described in the “Compiling” section of this document. The remainder of this section describes some things that are different

Two files need to be replaced to compile FAST as a DLL:

- FAST_Library.f90 should be used in place of FAST_Prog.f90
- SysMatlab.f90 should be used in place of SysIVF.f90. (Note that this assumes we're using the Intel Visual Fortran compiler. SysMatlab.f90 may need slight modification if using gfortran.)

A few compiling commands are different from the stand-alone application:

- The project must create a dynamic library (DLL or shared object) instead of a console application.
- The COMPILE_SIMULINK preprocessor directive must be used.
- The FAST library should be linked with libmex.lib. SysMatlab.f90 is designed to call MATLAB's mexPrintf function to print in the MATLAB Command Window. This function is part of the libmex.lib library, which is found in the %matlabroot%/extern/lib/{architecture}/{compiler}/ folder. For the 64-bit version of MATLAB R2014b, using Visual Studio, the file is located here:
  "C:\Program Files\MATLAB\R2014b\extern\lib\win64\microsoft\libmex.lib"

  If you wish to compile the library without linking with libmex.lib, use the preprocessor directive CONSOLE_FILE. This directive will write to a text file called CONSOLE.TXT instead of to the MATLAB Command Window

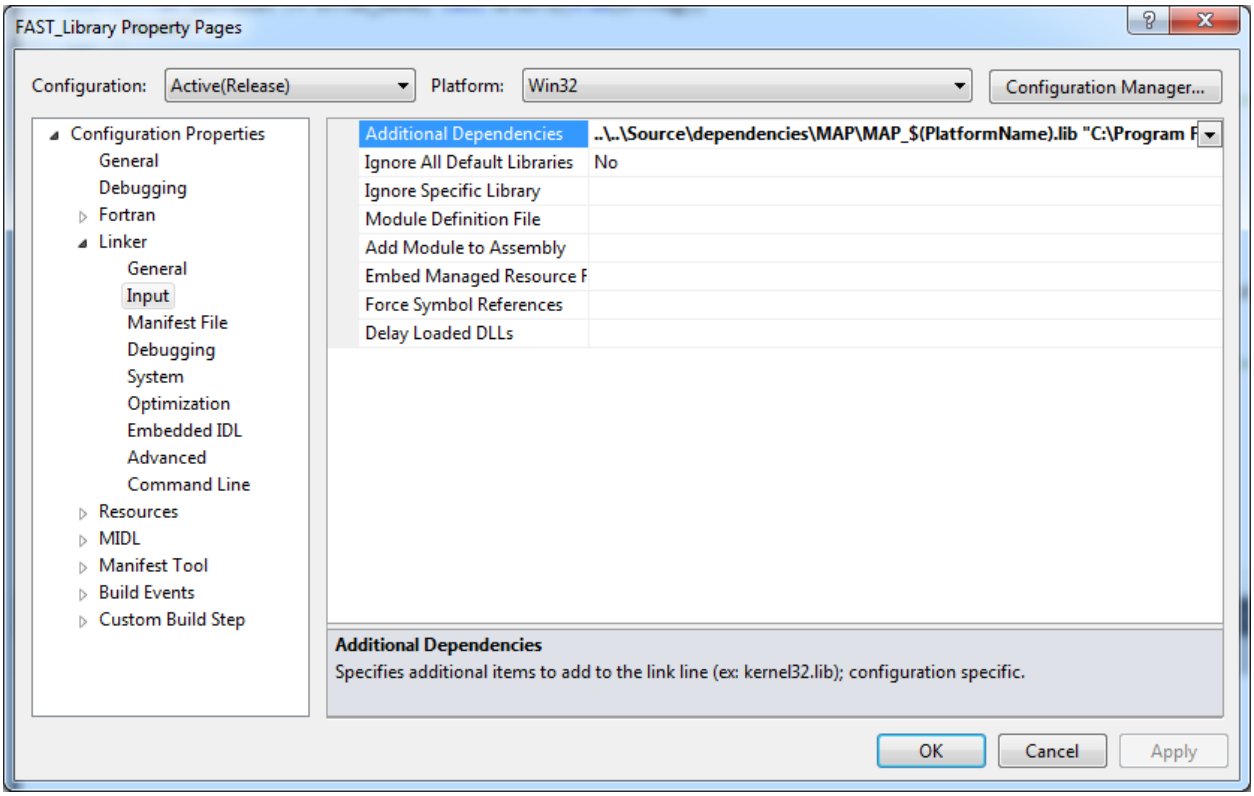

Figure 11: Visual Studio window showing additional dependencies for the linker (MAP and libmex)

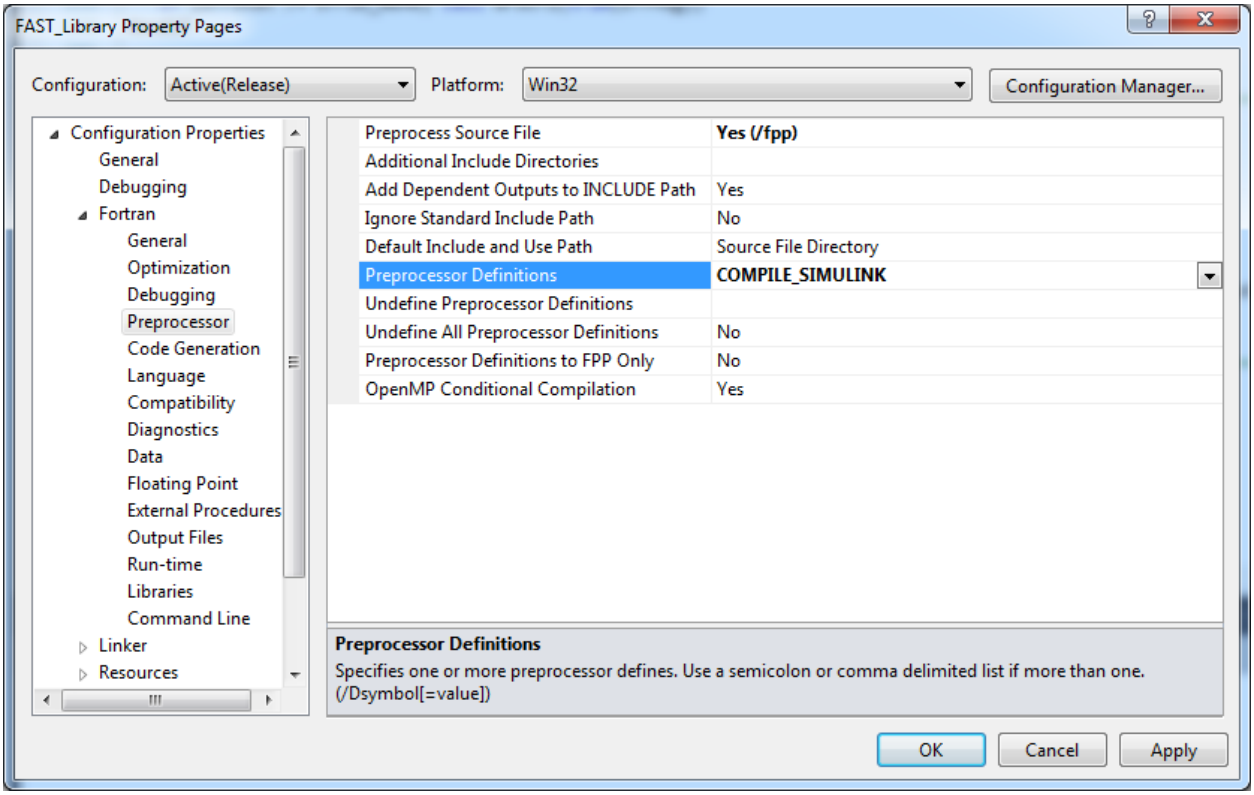

Figure 12: Preprocessor directives for FAST_Library

FAST_SFunc

To compile the FAST_SFunc S-Function, you must have a C compiler supported by the version of MATLAB you are using. The compilation is done in MATLAB via the "mex" command. If you haven't already done so, first run "mex –setup" from the MATLAB Command Window to select your compiler.

For 32-bit versions of MATLAB, the command to compile FAST_SFunc is

```
mex -L../../bin -lFAST_Library_Win32 -I../../Source ...
      -I../../Source/dependencies/OpenFOAM -outdir ../../bin FAST_SFunc.c
```

For 64-bit versions of MATLAB it is

```
mex -v -L../../bin -lFAST_Library_x64 -I../../Source ...
      -I../../Source/dependencies/OpenFOAM -outdir ../../bin FAST_SFunc.c
```

These commands are specified in the <FAST8>/Simulink/Source/create_FAST_SFunc.m file. The script assumes it is being run from the <FAST8>/Simulink/Source/ directory where the script and source files reside.

These commands compile FAST_SFunc.c (which includes header files FAST_Library.h in the ../../Source directory and OpenFOAM_Types.h in the ../../Source/dependencies/OpenFOAM directory) and link it with the FAST_Library_*.lib file (in the ../../bin directory) for the appropriate addressing scheme (32- or 64-bits). If you are using a 32-bit Windows® version of MATLAB, this command will produce

../../bin/FAST_SFunc.mexw32. On 64-bit Windows versions of MATLAB, it will produce ../../bin/FAST_SFunc.mexw64.

Because FAST_SFunc.c passes the input and output arrays directly to the FAST_Library*.dll file, FAST_SFunc should not need to be modified (or recompiled) very often. For example, if you wanted to add an additional FAST_SFunc input from Simulink, you would change the third S-Function parameter in the S-Function block in Simulink and then modify FAST_Library.f90 to handle the new input value, without touching FAST_SFunc.c; FAST_Library would need to be recompiled, but FAST_SFunc would not.

Future Work

All future developments of FAST will follow the framework.

- Introduce operating-point determination and linearization across the coupled aero-hydro-servo-elastic solution.
- Add items from FAST v7 that are not yet in FAST v8.
- Add visualization capabilities.
- Upgrade the loose-coupling algorithm of the glue code to allow each module to have its own time step, including time steps larger than the glue-code time step.
- Optimize the code, particularly BeamDyn and ElastoDyn, so that it runs faster. We have put our effort into getting the framework to work and hope to address computational efficiency later. We expect improvements in efficiency as development continues.
- Introduce tight coupling.
- And much, much, more…

Feedback

If you have questions or wish to provide feedback, please use our forums:
https://wind.nrel.gov/forum/wind/

Appendix A: Example FAST v8.12.* Input File

```
------- FAST v8.12.* INPUT FILE -------------------------------------------------
NREL 5.0 MW Baseline Wind Turbine with OC4 Jacket Configuration, for use in offshore analysis
---------------------- SIMULATION CONTROL --------------------------------------
False       Echo           - Echo input data to <RootName>.ech (flag)
"FATAL"     AbortLevel     - Error level when simulation should abort (string) {"WARNING", "SEVERE", "FATAL"}
       60   TMax           - Total run time (s)
     0.01   DT             - Recommended module time step (s)
        2   InterpOrder    - Interpolation order for input/output time history (-) {1=linear, 2=quadratic}
        1   NumCrctn       - Number of correction iterations (-) {0=explicit calculation, i.e., no corrections}
    99999   DT_UJac        - Time between calls to get Jacobians (s)
    1E+06   UJacSclFact    - Scaling factor used in Jacobians (-)
---------------------- FEATURE SWITCHES AND FLAGS ------------------------------
        1   CompElast      - Compute structural dynamics (switch) {1=ElastoDyn; 2=ElastoDyn+BeamDyn blades}
        1   CompInflow     - Compute inflow wind velocities (switch) {0=still air;1=InflowWind;2=OpenFOAM}
        1   CompAero       - Compute aerodynamic loads (switch) {0=None; 1=AeroDyn v14; 2=AeroDyn v15}
        1   CompServo      - Compute control and electrical-drive dynamics (switch) {0=None; 1=ServoDyn}
        1   CompHydro      - Compute hydrodynamic loads (switch) {0=None; 1=HydroDyn}
        1   CompSub        - Compute sub-structural dynamics (switch) {0=None; 1=SubDyn}
        0   CompMooring    - Compute mooring system (switch) {0=None;1=MAP++;2=FEAMooring;3=MoorDyn;4=OrcaFlex}
        0   CompIce        - Compute ice loads (switch) {0=None; 1=IceFloe; 2=IceDyn}
---------------------- INPUT FILES ---------------------------------------------
"OC4Jacket_ElastoDyn.dat"  EDFile       - Name of file containing ElastoDyn input parameters (quoted string)
"unused"                   BDBldFile(1) - Name of file containing BeamDyn blade 1 inputs (quoted string)
"unused"                   BDBldFile(2) - Name of file containing BeamDyn blade 2 inputs (quoted string)
"unused"                   BDBldFile(3) - Name of file containing BeamDyn blade 3 inputs (quoted string)
"OC4Jacket_InflowWind.dat" InflowFile   - Name of file containing inflow wind input parameters (quoted string)
"OC4Jacket_AeroDyn.dat"    AeroFile     - Name of file containing aerodynamic input parameters (quoted string)
"OC4Jacket_ServoDyn.dat"   ServoFile    - Name of file with control/electric-drive inputs (quoted string)
"OC4Jacket_HydroDyn.dat"   HydroFile    - Name of file containing hydrodynamic inputs (quoted string)
"OC4Jacket_SubDyn.dat"     SubFile      - Name of file containing sub-structural inputs (quoted string)
"unused"                   MooringFile  - Name of file containing mooring system inputs (quoted string)
"unused"                   IceFile      - Name of file containing ice input parameters (quoted string)
---------------------- OUTPUT --------------------------------------------------
True        SumPrint       - Print summary data to "<RootName>.sum" (flag)
        1   SttsTime       - Amount of time between screen status messages (s)
    99999   ChkptTime      - Amount of time between creating checkpoint files for potential restart (s)
     0.05   DT_Out         - Time step for tabular output (s)(or "default")
        0   TStart         - Time to begin tabular output (s)
        2   OutFileFmt     - Format for tabular (time-marching) output file (switch) {1:out, 2:outb, 3:both}
True        TabDelim       - Use tab delimiters in text tabular output file? (flag)
"ES10.3E2"  OutFmt         - Format used for text tabular output, excluding the time channel. (quoted string)
```

Figure 13: Example FAST v8.12.00a-bjj Input File