IceFloe Package Integration with the FAST Framework

February 2014 T. McCoy

# FAST to IceFloe API

Currently only the standard FAST Framework Init, CalcOutput, and End subroutines are used by the IceFloe package. The state update subroutines used for loose coupling are included as stubs. Stubs for the tight coupling routines are not included. The names of the respective routines are:

* IceFloe\_Init
* IceFloe\_CalcOutput
* IceFloe\_End
* IceFloe\_UpdateStates
* IceFloe\_UpdateDiscState

Note most input parameters are only used for initialization and the only thing that is saved is the load time series and ice floe direction. The major exception is for the coupled crushing model where the load is dependent on the local velocity of the support structure.

# ISSUES

* I had trouble compiling with the pre-processor commands found in ModMesh so I commented them out.
* In order to allow for a more stand alone version of IceFloe I have used pointer variables to map arrays of data saved in the FAST framework parameter type to IceFloe variables. These pointers are allocated and de-allocated at the beginning and end of the routine IceFloe\_CalcOutput respectively.

# FAST Registry Variables

Table 1 Registry Type: InitInputType

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Variable Type** | **Description** |
| InputFile | CHARACTER(1024) | Name of file containing input parameters for ice load calculations |
| simLength | ReKi | Length of simulation in seconds, needed since load time series is pre-calculated for most loading types. |
| Tower or leg diameter (at waterline) | ReKi | I wasn’t sure if FAST could provide this so I have not implemented it yet here. Could be multiple values if there are multiple legs with different diameters. Currently these are read in from the parameter file. |
| OTHER data e.g. leg positions, gravity, water density |  | Not implement since I’m not sure what can be passed from FAST. |

Table 1 Registry Type: InitOutputType

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Variable Type** | **Description** |
| DummyInitVar | ReKi | Not used but could provide something. Suggestions? |

Table 1 Registry Type: ContinuousStateType

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Variable Type** | **Description** |
| DummyContStateVar | ReKi | Not used, no continuous states |

Table 1 Registry Type: DiscreteStateType

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Variable Type** | **Description** |
| DummyDiscStateVar | ReKi | Not used, no discrete states |

Table 1 Registry Type: ConstraintStateType

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Variable Type** | **Description** |
| DummyConstrStateVar | ReKi | Not used, no constraint states |

Table 1 Registry Type: OtherStateType

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Variable Type** | **Description** |
| DummyOtherStateVar | ReKi | Not used, no other states |

Table 1 Registry Type: ParameterType

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Variable Type** | **Description** |
| loadSeries {:}{:} | ReKi | Pre-calculated time series of loads, dimension (numTimeSteps, numLegs) |
| iceVel | ReKi | Ice floe velocity (constant from input parameters) |
| iceDirection | ReKi | Ice floe direction of movement (constant from input parameters) |
| minStrength | ReKi | Ice crushing strength for positive relative velocity between tower and ice(constant from input parameters) |
| minStrengthNegVel | ReKi | Ice crushing strength for negative relative velocity between tower and ice (constant from input parameters) |
| minStressRate | ReKi | stress rate at which minimum strength is reached for negative velocity, computed at initialization |
| crushArea | ReKi | Computed parameter for coupled crushing load calcs |
| coeffStressRate | ReKi | Computed parameter for coupled crushing load calcs |
| C(4) | ReKi | Cubic coefficients for transition curve to minimum strength at negative velocity |
| dt | ReKi | Time step |
| legX {:} | ReKi | Leg position x coordinate, one per leg (used to calculate equivalent loads for multi-leg structures, constant, not a state) |
| legY {:} | ReKi | Leg position y coordinate, one per leg (used to calculate equivalent loads for multi-leg structures, constant, not a state) |
| ks {:} | ReKi | Multi-leg sheltering factors, one per leg |
| numLegs | IntKi | Number of legs (=1 for monopile) |
| iceType | IntKi | Ice loading type parameter: 1 to 7 (see user’s manual) |
| logUnitNum | IntKi | Save this as the log file remains open for warning messages that may occur during update calls |
| singleLoad | Logical | Switch to indicate whether a multi-leg structure is approximated by a single beam. If so equivalent horizontal forces (Fx, Fy) are calculated as the sum of the forces from each leg. Also a torsion is calculated based on leg positions and individual leg forces. |
| initFlag | Logical | True for a successful initialization |

Table 1 Registry Type: InputType

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Variable Type** | **Description** |
| iceMesh | MeshType | Pass the local tower leg (at the water line) horizontal velocities (Vx, Yy) in global/inertial coordinates. Number of nodes = numLegs. Only required for coupled model. |

Table 1 Registry Type: OutputType

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Variable Type** | **Description** |
| iceMesh | MeshType | Pass the local tower leg (at the water line) horizontal forces and the torsion (Fx, Fy, Mz) in global/inertial coordinates. Number of nodes = numLegs. |