WEC-Sim Training Course

Online Training Materials

PRESENTED BY

Dominic D. Forbush, Sandia

Code Structure

WEC-Sim Directory Structure

WEC-Sim **source code** consists of:

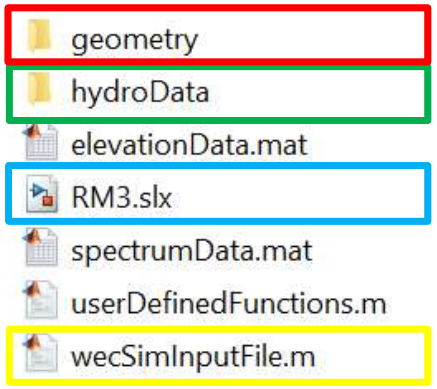
File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

*These are required inputs, but a number of optional additional inputs are possible, like custom wave spectra/ time-series.
See User Manual → Code Structure → spectrumImport

Use of Adv. Features (e.g. PTO-Sim and Moordyn) have additional requirements.
See Advanced Features

WEC-Sim **model files*** consist of:

File Type	File Name	Directory
Input File	wecSimInputFile.m	$<Case>
Simulink Model	<simulinkModelName>.slx	$<Case>
Hydrodynamic Data	<hydroDataName>.h5	$<Case/hydroDataDir>
Geometry File	<stlFileName>.stl	$<Case/geomDataDir>



WEC-Sim Source Code

WEC-Sim Source Code

WEC-Sim **source code** consists of:

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

- Source code is included in the MATLAB path
- Can be executed from any directory
- <object>Class.m methods require the <object> to be initialized before use

WEC-Sim Executable

WEC-Sim/source/wecSim.m

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

Running ***wecSim.m***

- clears needed variables: **this will delete unsaved outputs from previous runs**
- Calls *initializeWecSim.m,* which reads *wecSimInputFile.m* and:
  - performs preprocessing calculations in each of the classes
  - selects and initializes variant subsystems in the Simulink model
- runs the time-domain simulation of the Simulink model

View ***wecSim.m*** and ***initializeWecSim.m*** from MATLAB Command Window
>>edit <filename>

*** See Training Materials → Theory and Workflow for detailed walkthrough**

WEC-Sim Objects

WEC-Sim/source/objects/

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

Define classes in the *wecSimInputFile.m*

The following classes create the WEC-Sim objects

- ***simulationClass.m, waveClass.m, bodyClass.m, constraintClass.m, ptoClass.m, mooringClass.m, cableClass.m***

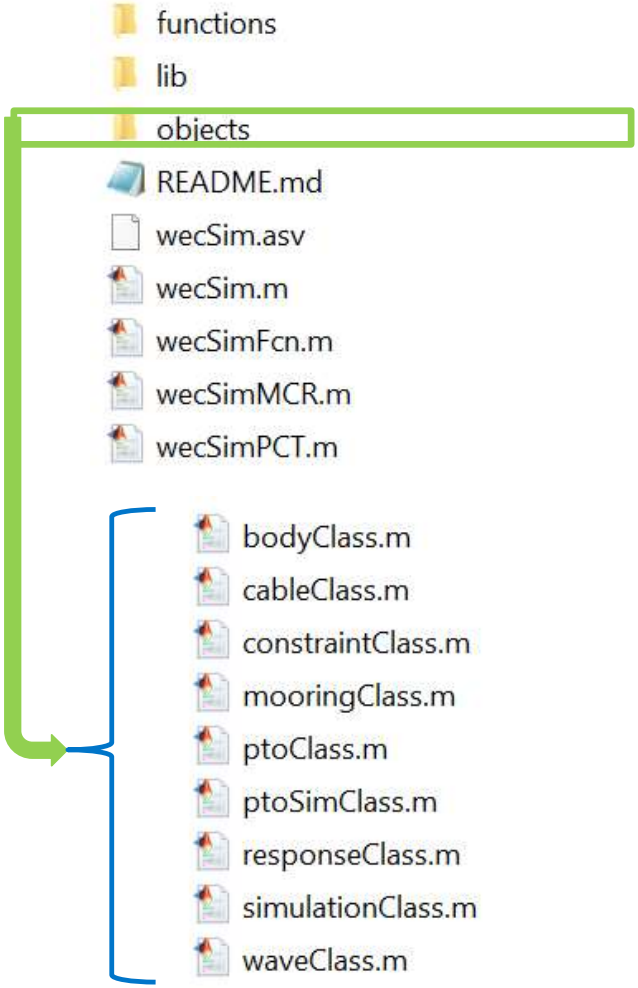
WEC-Sim objects are required to run WEC-Sim simulations

- ***simu, waves, body(i), pto(i) OR constraint(i)***
- Additional object types can be defined if desired

View properties or open classes from MATLAB Command Window
>> doc <className>
>> open <className>

***See User Manual→Code Structure → Source Detail for more**

WEC-Sim/source



WEC-Sim Library Blocks

WEC-Sim/source/lib/

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

WEC-Sim source code includes WEC-Sim library blocks:

- ***Body Elements, Constraints, Frames, Moorings, PTOs, Cables***

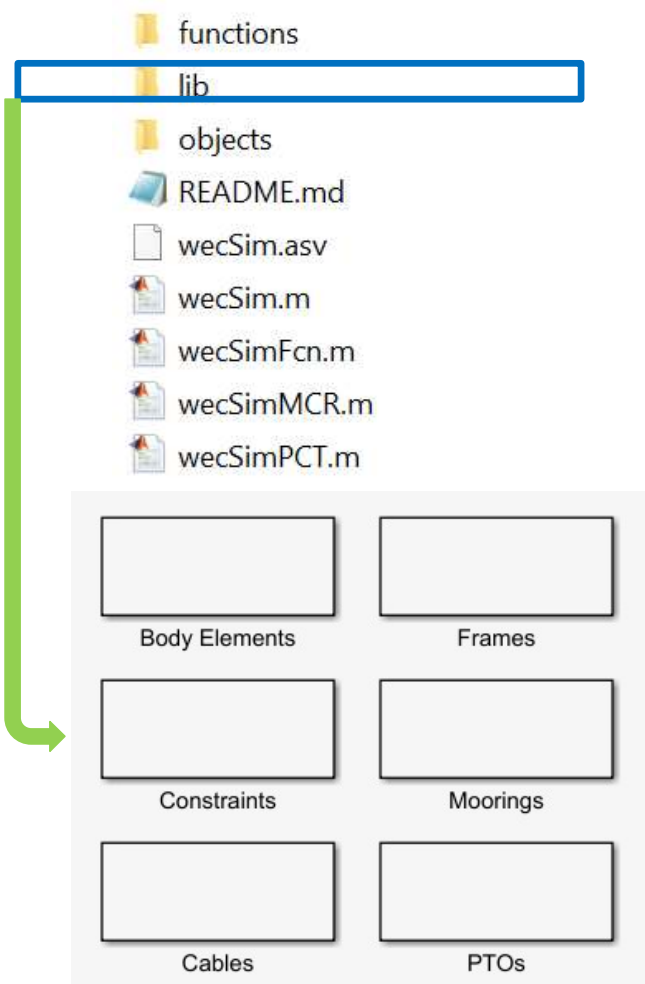
Define WEC dynamics in WEC-Sim Simulink model using WEC-Sim Library Blocks

- ***<SimulinkModelName>.slx***

View properties by double clicking on blocks, Ctrl+U to look under mask

All objects defined in the *wecSimInputFile* should also be blocks used in the Simulink model.

WEC-Sim/source



Simulink Mask/Model Functions

WEC-Sim/source/lib/

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

WEC-Sim/source/functions/simulink

To facilitate version control, within the Simulink model, mask functions and MATLAB functions reference externally-housed functions.

Name of the MATLAB function call is the same as the external function.

Only necessary to change the external function to affect model.

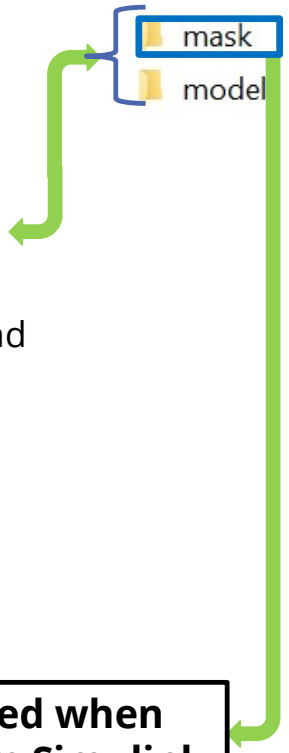

```
function dispPhase  = calcDispPhase(disp, enable, frequency, wavenumber, direction)
% initialize
dispPhase = zeros(length(frequency),length(direction));
% execute
dispPhase = calcDispPhase(disp, enable, direction, frequency, wavenumber);

end
```

Simulink Mask/Model Functions

WEC-Sim/source/lib/

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

WEC-Sim/source/functions/simulink



To facilitate version control, within the Simulink model, mask functions and MATLAB functions reference externally-housed functions.

Name of the MATLAB function call is the same as the external function.

Only necessary to change the external function to affect model.

*** $/source/functions/simulink/mask functions are mostly used when running from the Simulink GUI. See Adv. Features → Run from Simulink**

WEC-Sim Model Files

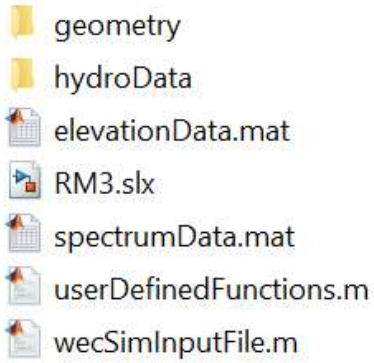
WEC-Sim Model Files

WEC-Sim **model files** consist of:

File Type	File Name	Directory
Input File	wecSimInputFile.m	$<Case>
Simulink Model	<simulinkModelName>.slx	$<Case>
Hydrodynamic Data	<hydroDataName>.h5	$<Case/hydroDataDir>
Geometry File	<stlFileName>.stl	$<Case/geomDataDir>

- Model files are located in the case directory
- ***WEC-Sim models must be executed from the case directory***

WEC-Sim/examples/RM3



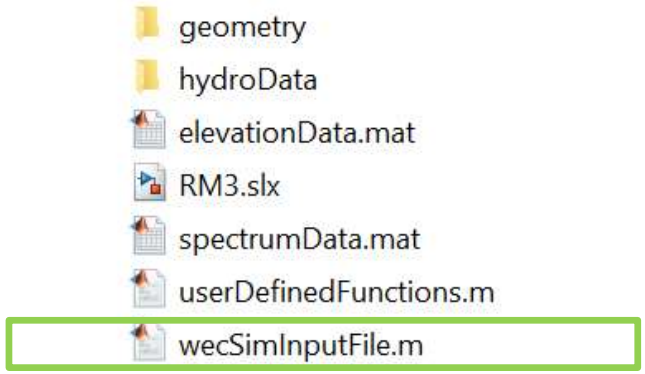
An example of a WEC-Sim case directory

WEC-Sim Input File

wecSimInputFile.m

File Type	File Name	Directory
Input File	wecSimInputFile.m	$<Case>
Simulink Model	<simulinkModelName>.slx	$<Case>
Hydrodynamic Data	<hydroDataName>.h5	$<Case/hydroDataDir>
Geometry File	<stlFileName>.stl	$<Case/geomDataDir>

Case Directory



- Necessary by default (see Adv. Feat.→Sim. Feat.→Running from Simulink)
- Located in the case directory
- Initialize and define classes in the WEC-Sim input file
  - ***wecSimInputFile.m***
- WEC-Sim objects are required to run WEC-Sim simulations
  - ***simu, waves, body(i), pto(i)*** **OR** ***constraint(i)***

no (i): Only one per simulation

(i): More than one OK

*** Additional optional objects are also defined in the wecSimInputFile**

See User Manual→Code Structure→WEC-Sim Classes

WEC-Sim Input File

wecSimInputFile.m

- Initialize Simulation Class
- Set Properties of Simulation Class

```
%% Simulation Data
simu = simulationClass();
simu.simMechanicsFile = 'RM3.slx';
simu.mode = 'normal';
simu.explorer = 'on';
simu.startTime = 0;
simu.rampTime = 100;
simu.endTime = 400;
simu.solver = 'ode4';
simu.dt = 0.1;
```

- Initialize Wave Class
- Set Properties of Wave Class

```
%% Wave Information
% % noWaveCIC, no waves with radiation CIC
waves = waveClass('regularCIC');
waves.period = 6; % seconds
waves.height = 1; % meters
```

- Initialize Body Class Instances
- Set Properties of Body Class Instances

```
%% Body Data
% Float
body(1) = bodyClass('hydroData/rm3.h5');
body(1).geometryFile = 'geometry/float.stl';
body(1).mass = 'equilibrium';
body(1).inertia = [20907301 21306090.66 37085481.11];

% Spar/Plate
body(2) = bodyClass('hydroData/rm3.h5');
body(2).geometryFile = 'geometry/plate.stl';
body(2).mass = 'equilibrium';
body(2).inertia = [94419614.57 94407091.24 28542224.82];
```

- Initialize Constraint Class
- Set Properties of Constraint Class
- Initialize PTO Class
- Set Properties of PTO Class

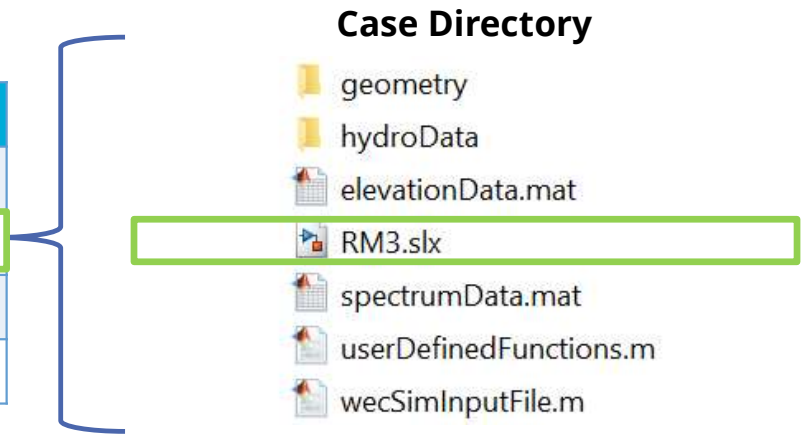
```
%% PTO and Constraint Parameters
% Floating (3DOF) Joint
constraint(1) = constraintClass('Constraint1');
constraint(1).location = [0 0 0];

% Translational PTO
pto(1) = ptoClass('PTO1');
pto(1).stiffness = 0;
pto(1).damping = 1200000;
pto(1).location = [0 0 0];
```

*** One can also define these parameters directly from the Simulink GUI if desired using the "Running From Simulink" work flow. See Adv. Features → Run from Simulink**

WEC-Sim Simulink File

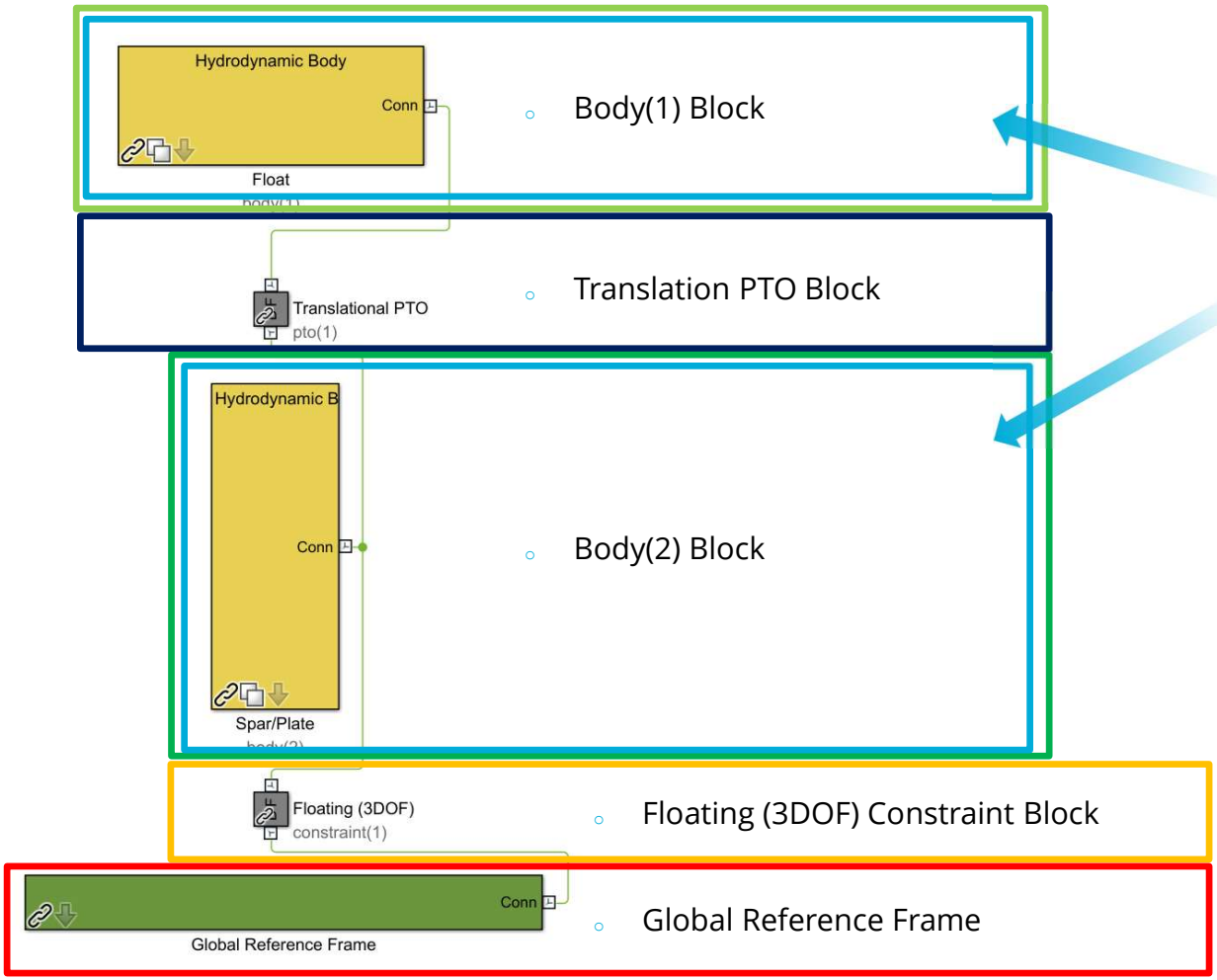
<simulinkModelName>.slx

File Type	File Name	Directory
Input File	wecSimInputFile.m	$<Case>
Simulink Model	<simulinkModelName>.slx	$<Case>
Hydrodynamic Data	<hydroDataName>.h5	$<Case/hydroDataDir>
Geometry File	<stlFileName>.stl	$<Case/geomDataDir>



- Located in the case directory
- Define model file using WEC-Sim Library Blocks
  - ***<simulinkModelName>.slx***

WEC-Sim Simulink File



- Body(1) Block
- Translation PTO Block
- Body(2) Block
- Floating (3DOF) Constraint Block
- Global Reference Frame

```
%% Simulation Data
simu = simulationClass();
simu.simMechanicsFile = 'RM3.slx';
simu.mode = 'normal';
simu.explorer = 'on';
simu.startTime = 0;
simu.rampTime = 100;
simu.endTime = 400;
simu.solver = 'ode4';
simu.dt = 0.1;
```

```
%% Wave Information
% regular waves with radiation CIC
waves = waveClass('regularCIC');
waves.period = 6; % seconds
waves.height = 1; % meters
```

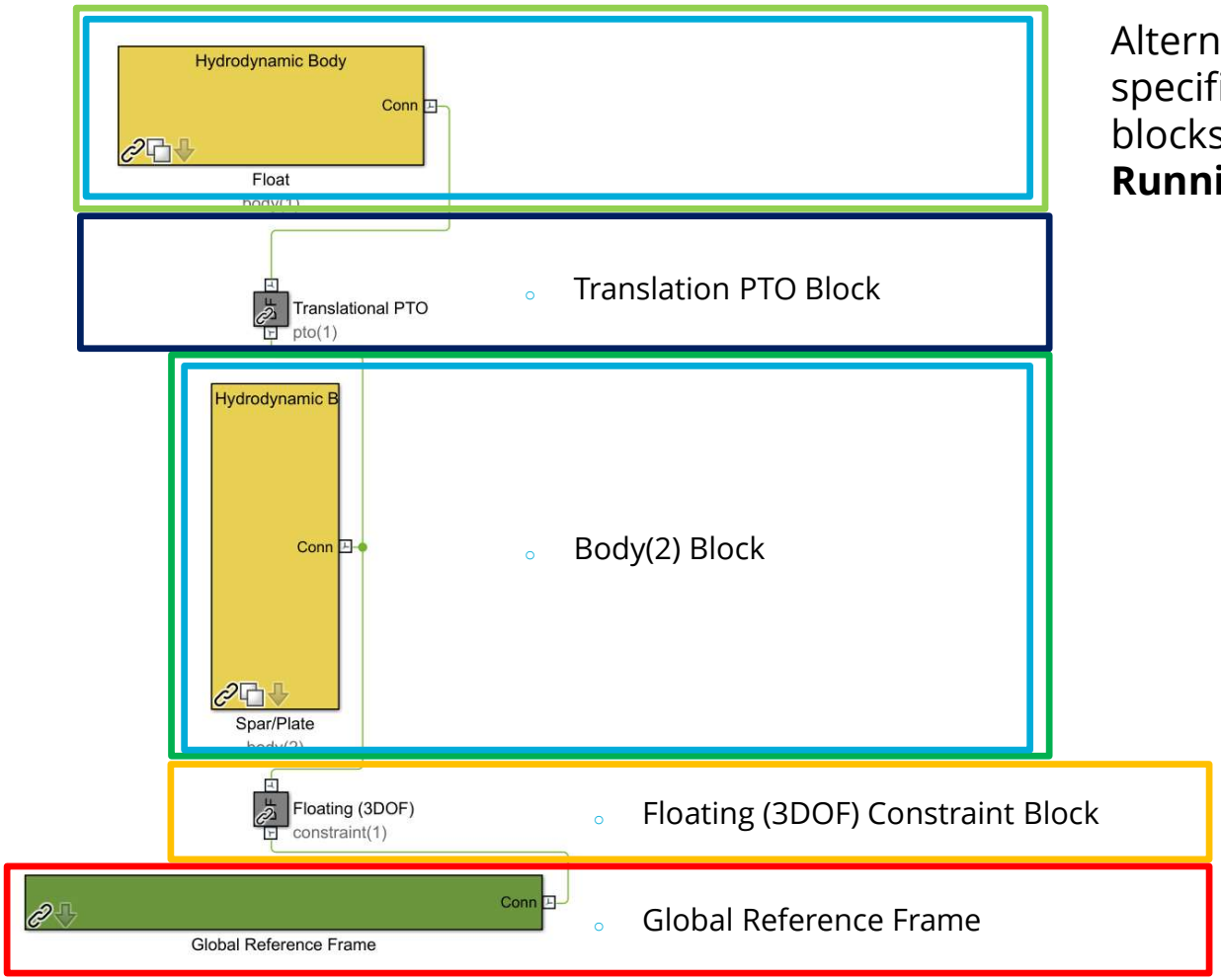
```
%% Body Data
% Float
body(1) = bodyClass('hydroData/rm3.h5');
body(1).geometryFile = 'geometry/float.stl';
body(1).mass = 'equilibrium';
body(1).inertia = [20907301 21306090.66 37085481.11];
```

```
% Spar/Plate
body(2) = bodyClass('hydroData/rm3.h5');
body(2).geometryFile = 'geometry/plate.stl';
body(2).mass = 'equilibrium';
body(2).inertia = [94419614.57 94407091.24 28542224.82];
```

```
%% PTO and Constraint Parameters
% Floating (3DOF) Joint
constraint(1) = constraintClass('Constraint1');
constraint(1).location = [0 0 0];
```

```
% Translational PTO
pto(1) = ptoClass('PTO1');
pto(1).stiffness = 0;
pto(1).damping = 1200000;
pto(1).location = [0 0 0];
```

WEC-Sim Simulink File



Alternatively, the parameters can be specified directly in the relevant Simulink blocks using **Advanced Features → Running from Simulink**

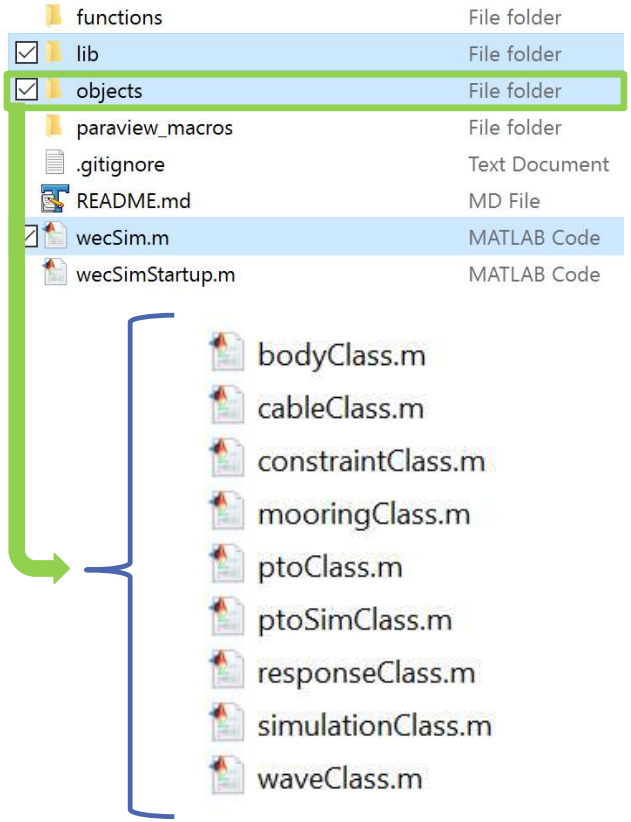
WEC-Sim Objects

WEC-Sim Objects

WEC-Sim/source/objects/

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/Simulink

WEC-Sim/source



Define classes in the *wecSimInputFile.m*

The following classes create the WEC-Sim objects

- ***simulationClass.m, waveClass.m, bodyClass.m, constraintClass.m, ptoClass.m, mooringClass.m, cableClass.m***

WEC-Sim objects are required to run WEC-Sim simulations

- ***simu, waves, body(i), pto(i) OR constraint(i)***
- Additional object types can be defined if desired

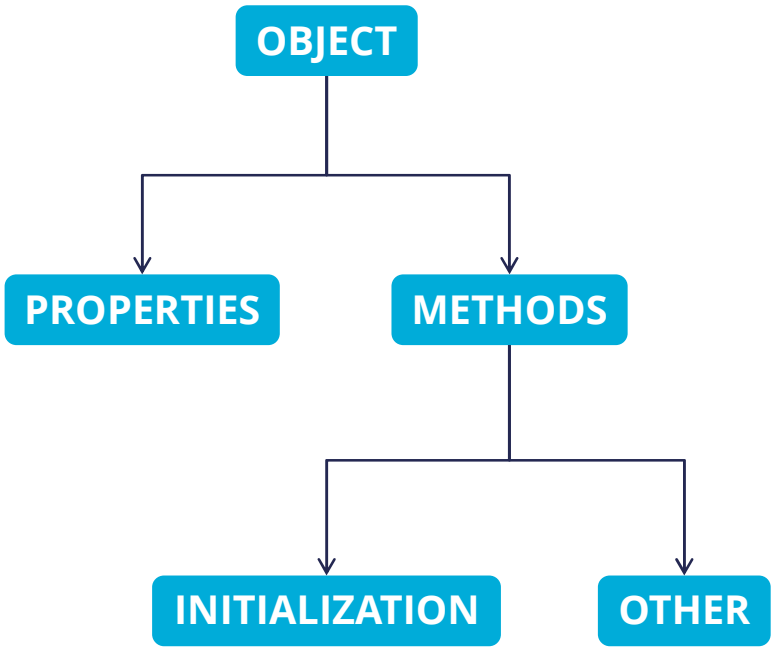
View properties or open classes from MATLAB Command Window
>> doc <className>
>> open <className>

***See User Manual→Code Structure → Source Detail for more**

WEC-Sim Objects

WEC-Sim has several different classes

- ***simulationClass.m***
- ***waveClass.m***
- ***bodyClass.m***
- ***constraintClass.m***
- ***ptoClass.m***
- ***mooringClass.m***
- ***responseClass.m***
- ***cableClass.m***



Each class contains:

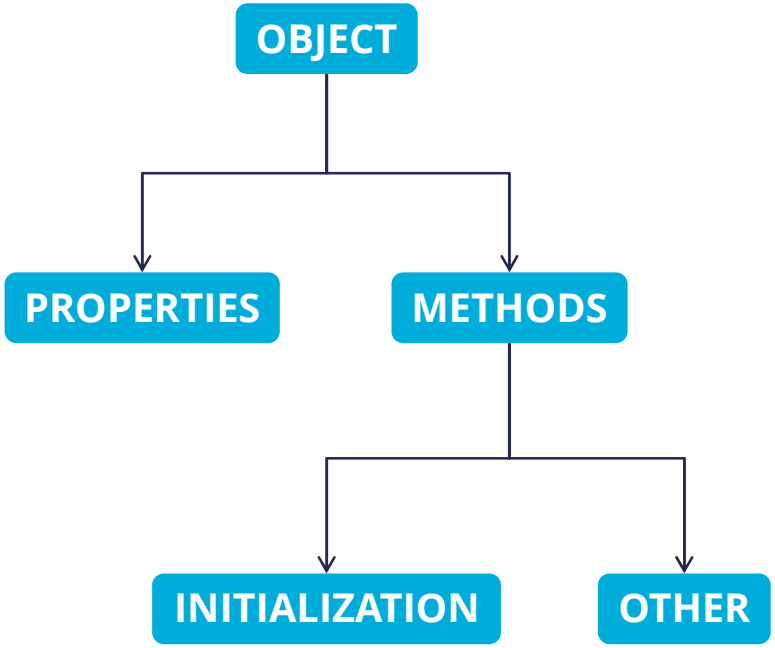
- Properties that can be defined and/or calculated
- Methods (aka functions) that can be executed

WEC-Sim input file determines which properties are defined and methods are executed

WEC-Sim Objects

Each class creates a corresponding object that will appear in the workspace

- ***simulationClass.m*** ➡ ***simu***
- ***waveClass.m*** ➡ ***waves***
- ***bodyClass.m*** ➡ ***body(i)***
- ***constraintClass.m*** ➡ ***constraint(i)***
- ***ptoClass.m*** ➡ ***pto(i)***
- ***mooringClass.m*** ➡ ***mooring(i)***
- ***responseClass.m*** ➡ ***output***
- ***cableClass.m*** ➡ ***cable(i)***



Some properties are used to specify a variant subsystem, e.g.

- ***simu.b2b = 1;***
- ***body(i).nhBody = 1;***
- ***waves = waveClass('regular');***

For help, >>doc <name>Class

See also User Manual→ Code Structure → WEC-Sim Classes

WEC-Sim Class Descriptions

Simulation Class

simulationClass.m

The simulation class contains the simulation parameters and solver settings necessary to execute the WEC-Sim code.

Required Properties:

- simMechanicsFile
- startTime, endTime, dt, rampTime, cicEndTime
  - (many have default values)

*** See User Manual → Code Structure → Simulation Class**

and

API → Simulation Class

>>simu

```
simu =

  simulationClass with properties:

      adjMassFactor: 2
                b2b: 0
              cicDt: 0.1000
         cicEndTime: 60
         domainSize: 200
                 dt: 0.1000
              dtOut: 0.1000
            endTime: 400
           explorer: 'on'
            gravity: 9.8100
          mcrMatFile: [1×0 char]
        mcrExcelFile: [1×0 char]
               mode: 'normal'
          morisonDt: 0.1000
        nonlinearDt: 0.1000
           paraview: [1×1 struct]
           pressure: 0
           rampTime: 100
     rateTransition: 'on'
       reloadH5Data: 0
                rho: 1000
      saveStructure: 0
           saveText: 0
      saveWorkspace: 1
```

Wave Class

waveClass.m

The wave class contains all wave information necessary to define the incident wave condition for the WEC-Sim time-domain simulation. In the Simulink model, wave forces are applied inside the body(i) blocks.

Required Properties:

- type
- Each wave 'type' has different required properties

Wave Type	Required Properties
`noWave`	`waves.period`
`noWaveCIC`	
`regular`	`waves.height` , `waves.period`
`regularCIC`	`waves.height` , `waves.period`
`irregular`	`waves.height` , `waves.period` , `waves.spectrumType`
`spectrumImport`	`waves.spectrumFile`
`elevationImport`	`waves.elevationFile`

*** See Training Videos→ Wave Implementation, User Manual → Code Structure → Wave Class, and API → Wave Class**

>>waves

```
waveClass with properties:

              bem: [1×1 struct]
          current: [1×1 struct]
        direction: 0
    elevationFile: 'NOT DEFINED'
            gamma: [1×0 double]
           height: 1
           marker: [1×1 struct]
           period: 6
        phaseSeed: 0
    spectrumFile: 'NOT DEFINED'
    spectrumType: 'NOT DEFINED'
             viz: [1×1 struct]
      waterDepth: 200
           spread: 1
        amplitude: 0.5000
        deepWater: 1
           dOmega: 0
            omega: 1.0472
            phase: 0
            power: 5.7437e+03
         spectrum: []
             type: 'regularCIC'
          typeNum: 11
      waveAmpTime: [4001×2 double]
   waveAmpTimeViz: []
       wavenumber: 0.1118
```

Body Class

bodyClass.m

The body class contains the mass and hydrodynamic properties of each body that comprises the WEC being simulated.

Required Properties:

- mass: value, 'equilibrium'
- inertia
- product of inertia (v5.1.0 release)
- geometryFile (**This is used for visualization and some Adv. Feat.)**
- h5File (**This contains hydrodynamic data from BEM)**

***See Training Videos → Body Class Implementation**

User Manual→Code Structure → Source Details Body Class

And

API → Body Class

>>body

```
body =

  1×2 bodyClass array with properties:

    centerBuoyancy
    centerGravity
    dof
    excitationIRF
    flex
    gbmDOF
    geometryFile
    h5File
    hydroStiffness
    inertia
    initial
    largeXYDisplacement
    linearDamping
    mass
    meanDrift
    morisonElement
    name
    nonHydro
    nonlinearHydro
    quadDrag
    paraview
    viz
    volume
    yaw
    dofEnd
    dofStart
```

Constraint and PTO Classes

constraintClass.m ***ptoClass.m***

Constraint blocks connect WEC bodies to one another (and possibly to the seabed) by constraining DOFs. PTOs do the same and can also apply force along their DOF of action. Unique blocks are available for different DOF restriction (e.g., rotational, translational, spherical)

Constraint and PTO Class required properties:

- name
- location

Additional PTO Class properties that describe applied force. The length of these fields must match the number of unconstrained DOF in the PTO.

- stiffness (*non-negative)*
- pretension
- damping

*** For additional information, see:**
User Manual → Code Structure → Constraint Class
User Manual → Code Structure → PTO Class
API → Constraint Class
API→ PTO Class

*** For component-level PTO design, see also Adv. Features → PTO-Sim**

>>constraint

```
constraint =

  constraintClass with properties:

      hardStops: [1×1 struct]
        initial: [1×1 struct]
       location: [0 0 0]
           name: 'Constraint1'
    orientation: [1×1 struct]
         number: 1
```

>> pto

```
>> pto

pto =

  ptoClass with properties:

                damping: 1200000
  equilibriumPosition: 0
            hardStops: [1×1 struct]
              initial: [1×1 struct]
             location: [0 0 0]
                 name: 'PTO1'
          orientation: [1×1 struct]
           pretension: 0
            stiffness: 0
               number: 1
```

Mooring Class

mooringClass.m

Mooring class defines the mooring system as either a linear mooring matrix or a MoorDyn model. It is designed to couple a WEC body, PTO, or Constraint to the sea-bed

Mooring types:

- matrices
- MoorDyn

Properties for matrix:

- name
- location
- Matrix
  - stiffness
  - damping
  - pretension

For additional information, see:

User Manual→ Code Structure→ Mooring Class

Advanced Features → Mooring Features→ MoorDyn

API→ Mooring Class

>>mooring

```
mooring =

  mooringClass with properties:

          initial: [1×1 struct]
         location: [0 0 0]
           matrix: [1×1 struct]
          moorDyn: 0
      moorDynLines: 0
      moorDynNodes: [1×0 double]
              name: 'Mooring1'
       orientation: [0 0 0 0 0 0]
            number: 1
```

>>mooring.matrix

```
ans =

  struct with fields:

        damping: [6×6 double]
      stiffness: [6×6 double]
     preTension: [0 0 100 0 0 0]
```

Cable Class

cableClass.m

Cable class describes a compliant cable that connects two constraints or PTOs. The constraint/PTO defines how the cable connection is allowed to move. If the cable is not in tension, it does not transmit force between the connection points.

Cable required properties:

- stiffness
- damping

By default, cable length and end locations, are determined from the connected constraints/PTOs, assuming zero pretension.

See also: WEC-Sim Applications/Cable

Advanced Features →Cable Features

User Manual → Code Structure → Cable Class

```
>>cable

cable =

  cableClass with properties:

        damping: 100
        inertia: [1 1 1]
        initial: [1×1 struct]
    cableLength: 17.8000
  linearDamping: [0 0 0 0 0 0]
           mass: 1
           name: 'Cable'
    orientation: [1×1 struct]
       paraview: 1
     preTension: 0
       quadDrag: [1×1 struct]
      stiffness: 1000000
            viz: [1×1 struct]
           base: [1×1 struct]
       follower: [1×1 struct]
       location: [999 999 999]
         volume: []
         number: []
```

Response Class (Output Structure)

responseClass.m

'output' created at the end of a WEC-Sim simulation. It contains all the output time-series and methods to plot and interact with the results.

output = responseClass instance

- Contains all time series from simulation
- Contains all time-series calculations
- Methods for quick plotting

Properties are all defined objects, each with their own sub-fields.

This structure is created before *userDefinedFunctions* runs, so *userDefinedFunctions* can reference **output.**

For additional information, see:
User Manual → Code Structure → Response Class and API → Response Class

>>output

```
output =

  responseClass with properties:

        bodies: [1×2 struct]
        cables: [1×1 struct]
   constraints: [1×1 struct]
       moorDyn: [1×1 struct]
       mooring: [1×1 struct]
          ptos: [1×1 struct]
       ptosim: [1×1 struct]
          wave: [1×1 struct]
```

>>output.ptos(1)

```
ans =

  struct with fields:

                    name: 'PTO1'
                    time: [4001×1 double]
                position: [4001×6 double]
                velocity: [4001×6 double]
            acceleration: [4001×6 double]
              forceTotal: [4001×6 double]
           forceActuation: [4001×6 double]
          forceConstraint: [4001×6 double]
     forceInternalMechanics: [4001×6 double]
     powerInternalMechanics: [4001×6 double]
```

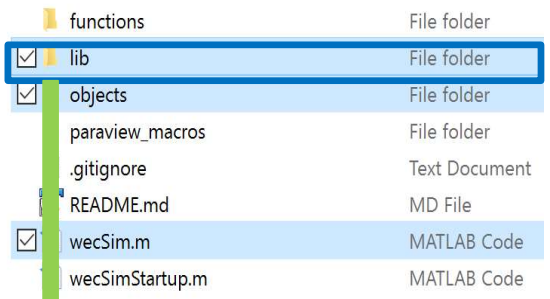
WEC-Sim Library

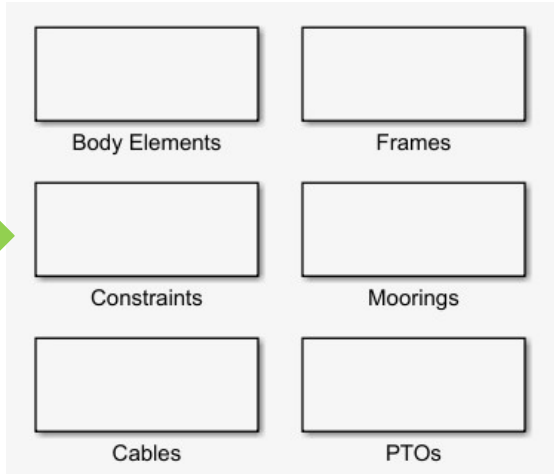
WEC-Sim Library Blocks

WEC-Sim/source/lib/

File Type	File Name	Directory
WEC-Sim Executable	wecSim.m	$source
WEC-Sim MATLAB Objects	<object>Class.m	$source/objects
WEC-Sim Simulink Library	<object>_Lib.slx	$source/lib/WEC-Sim
Simulink Mask/Model Functions	<functionName>.m	$source/functions/simulink

WEC-Sim/source





WEC-Sim source code includes WEC-Sim library blocks:

- ***Body Elements, Constraints, Frames, Moorings, PTOs, Cables***

Define WEC dynamics in WEC-Sim Simulink model using WEC-Sim Library Blocks

- ***<SimulinkModelName>.slx***

View properties by double clicking on blocks, Ctrl+U to look under mask
https://www.mathworks.com/help/simulink/block-masks.html

All objects defined in the *wecSimInputFile* should also be blocks used in the Simulink model.

WEC-Sim Library

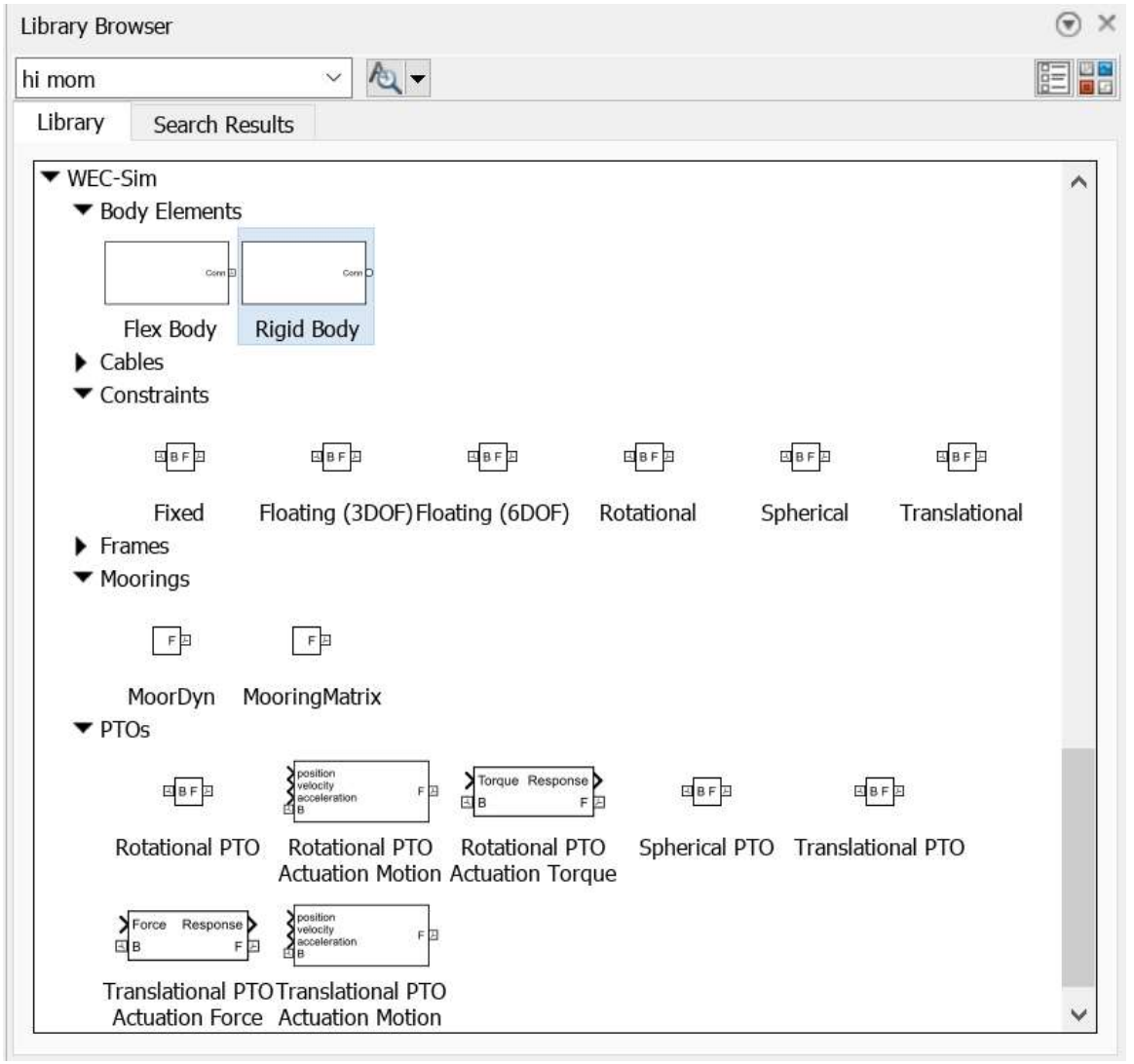
WEC-Sim/source/lib/

WEC-Sim Library

- Drag & Drop library
- “Source Code” blocks

Simulink Model

- Made of WEC-Sim library blocks
- Blocks cannot have the same name: model will automatically number repeated block types.

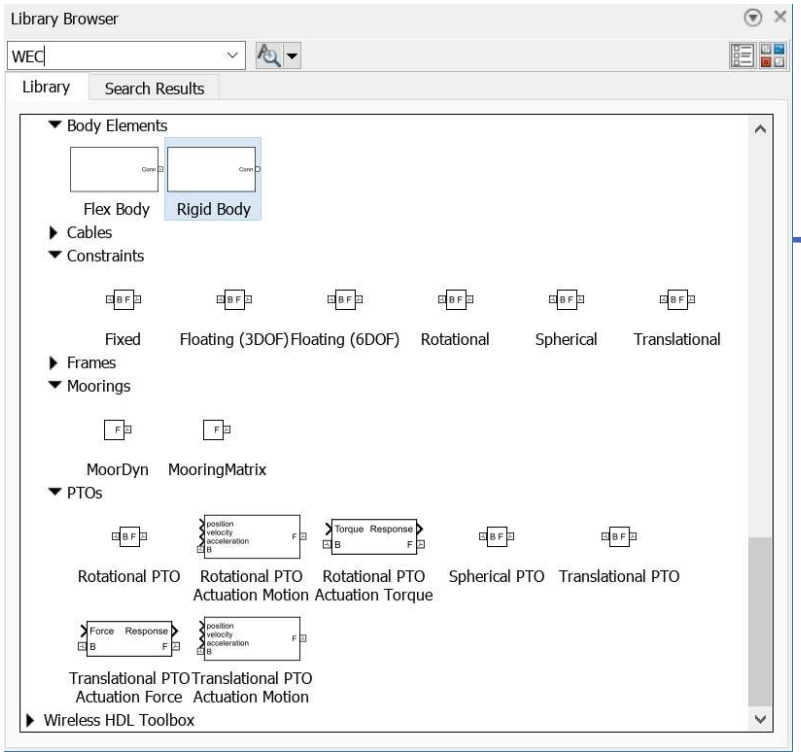


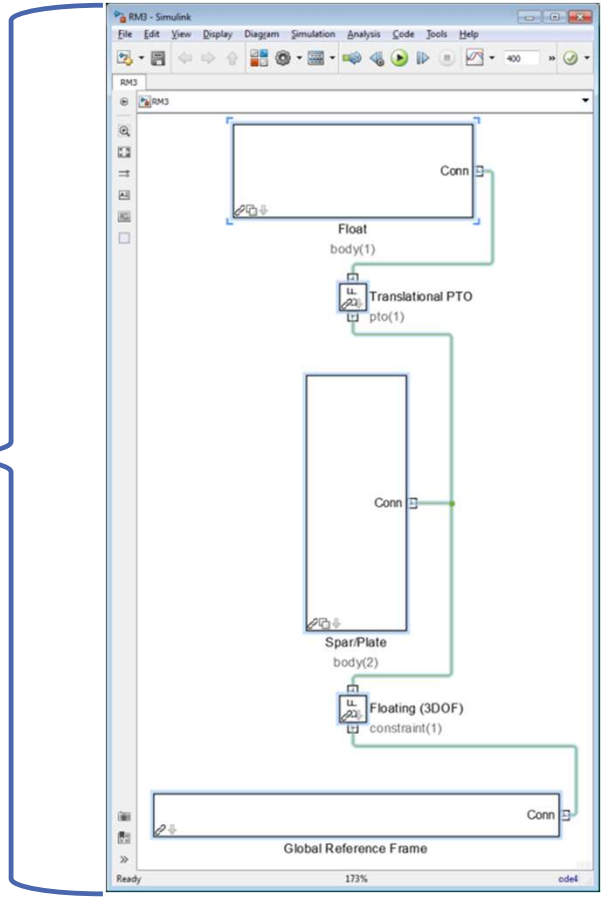
WEC-Sim Simulink File

<simulinkModelName>.slx

WEC-Sim Simulink Model

- Created with WEC-Sim Simulink Library Blocks
- Free to incorporate other Simscape/Simulink components



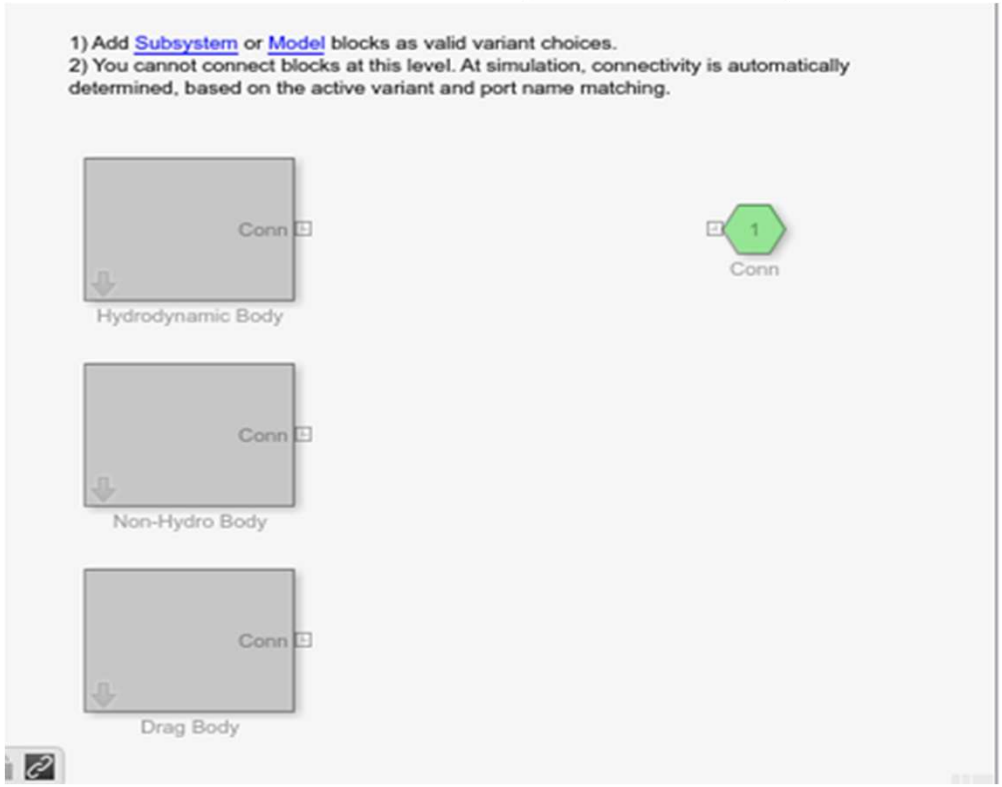


Variant Subsystems

WEC-Sim/source/lib/

Many library blocks contain 'Variant Subsystems'

- Variant subsystems allow multiple implementations to exist within a single model, with one active at a time.
- You can programmatically swap out the active implementation and replace it with one of the other implementations without modifying the model.
- The specification of active subsystems happens within *initializeWecSim.m*.

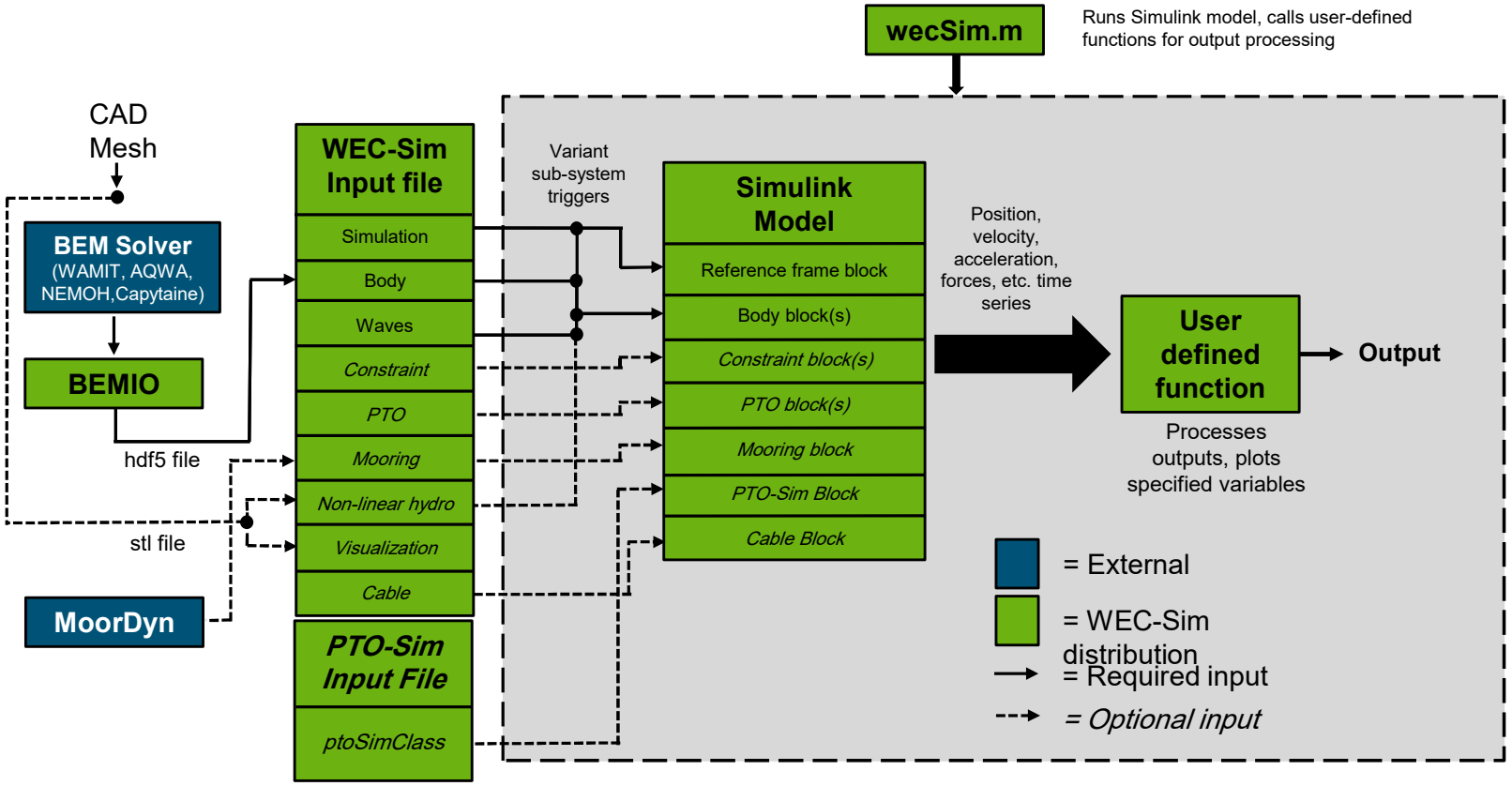


*** See Training Materials → Theory and Workflow for more information**

https://www.mathworks.com/help/simulink/examples/variant-subsystems.html

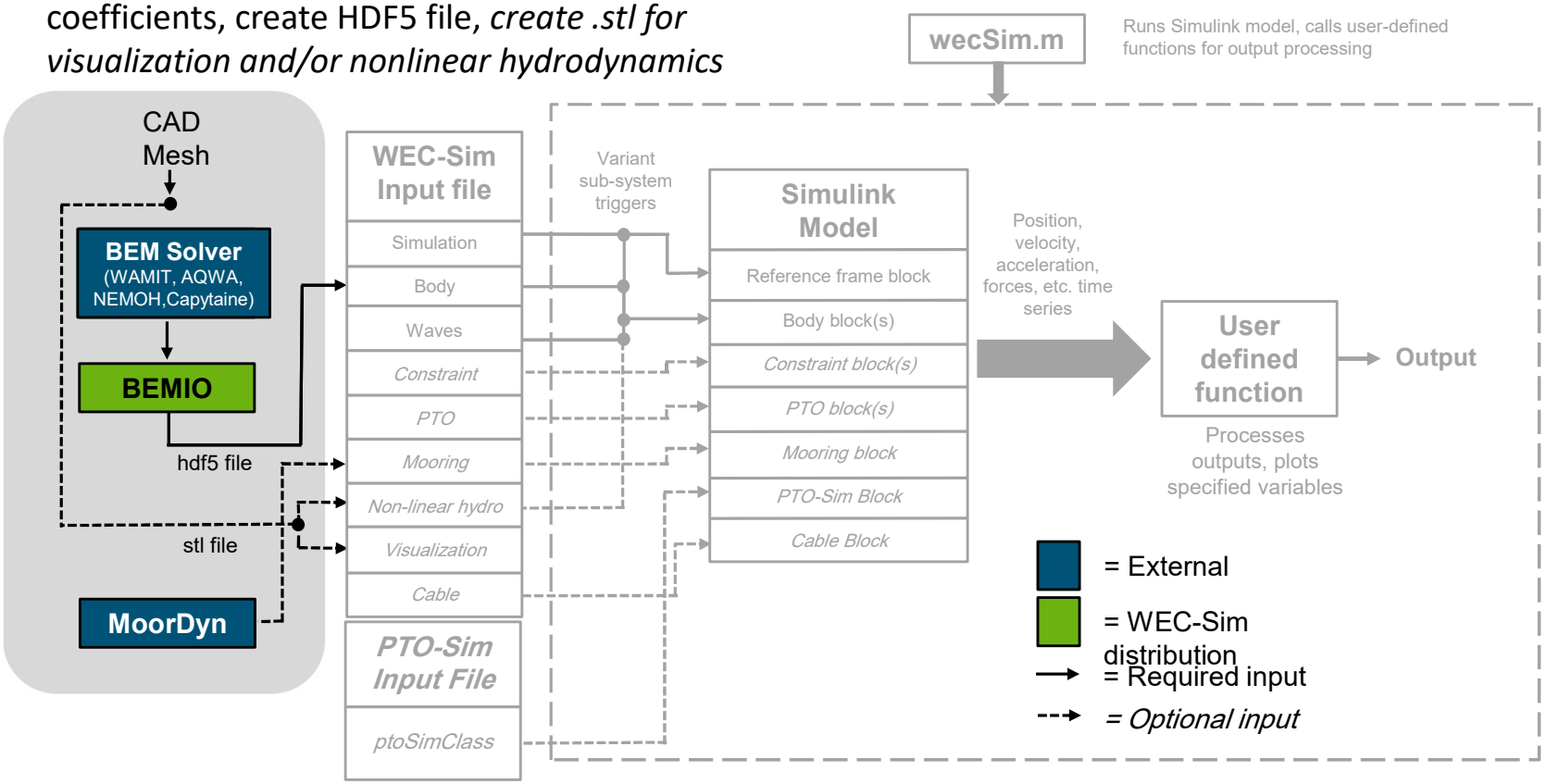
In conclusion…

WEC-Sim Workflow



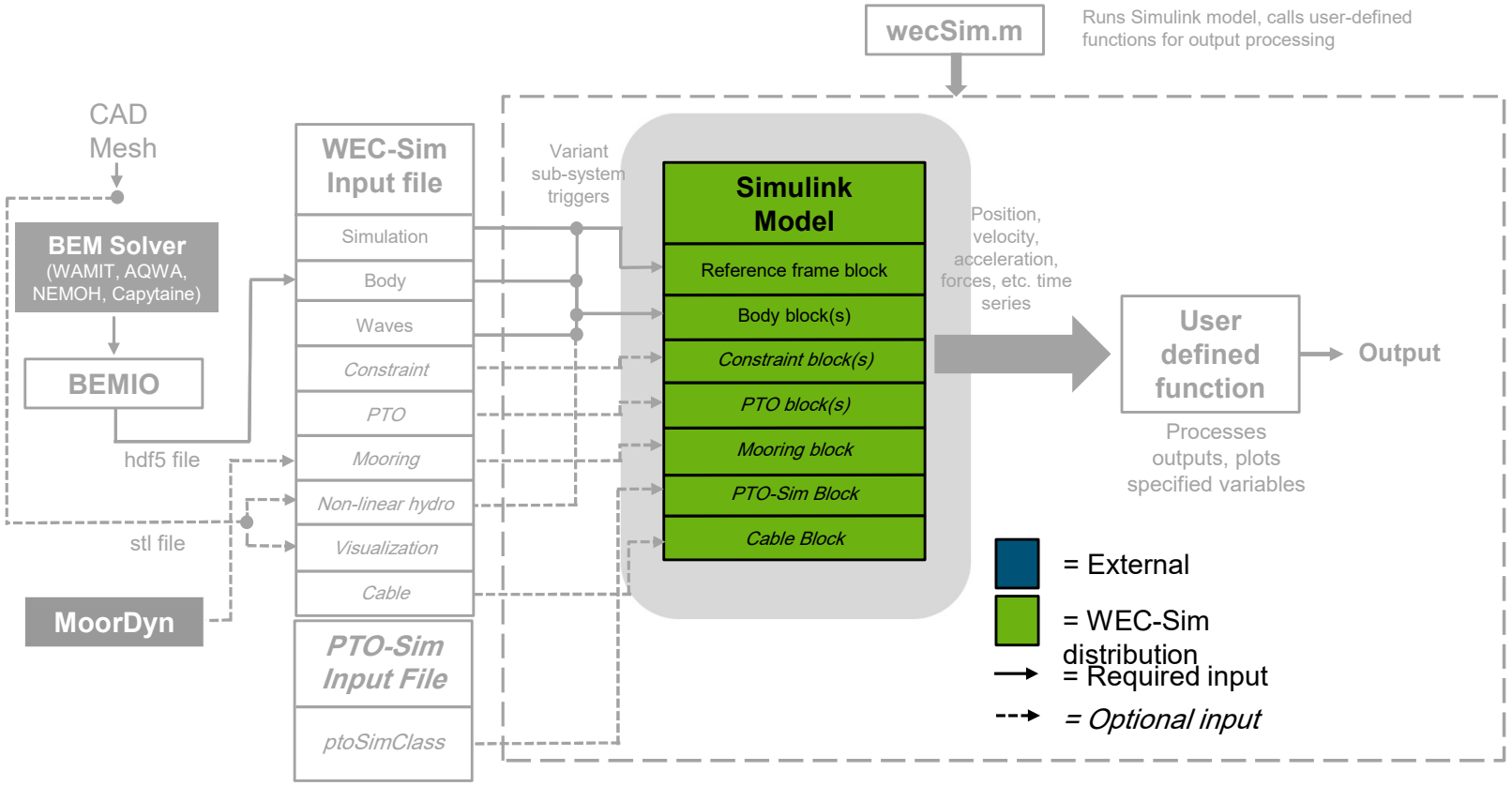
WEC-Sim Workflow

1). Generate 3-D mesh, calculate hydrodynamic coefficients, create HDF5 file, *create .stl for visualization and/or nonlinear hydrodynamics*



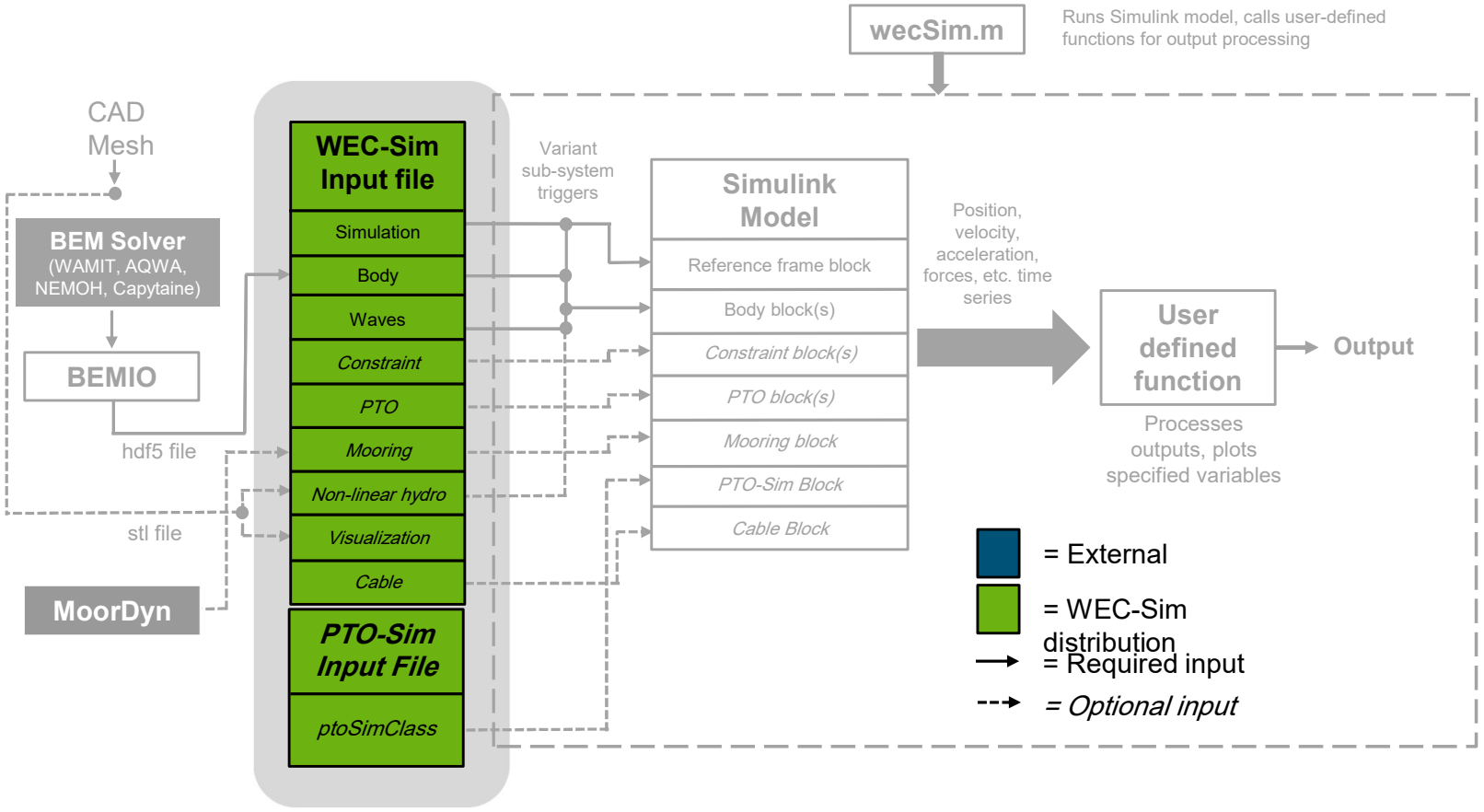
WEC-Sim Workflow

2). Build WEC-Sim model in Simulink



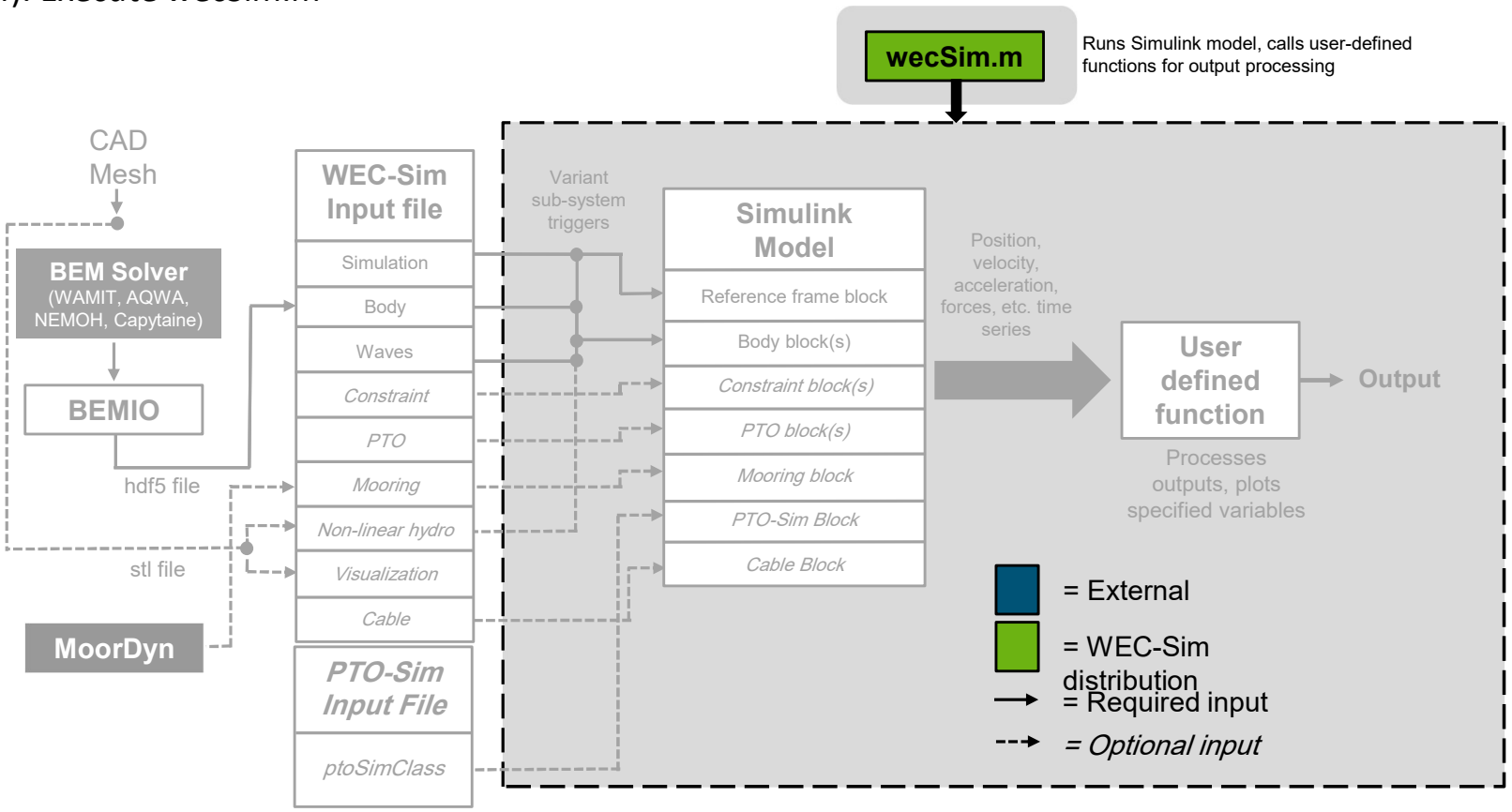
WEC-Sim Workflow

3). Write WEC-Sim input file



WEC-Sim Workflow

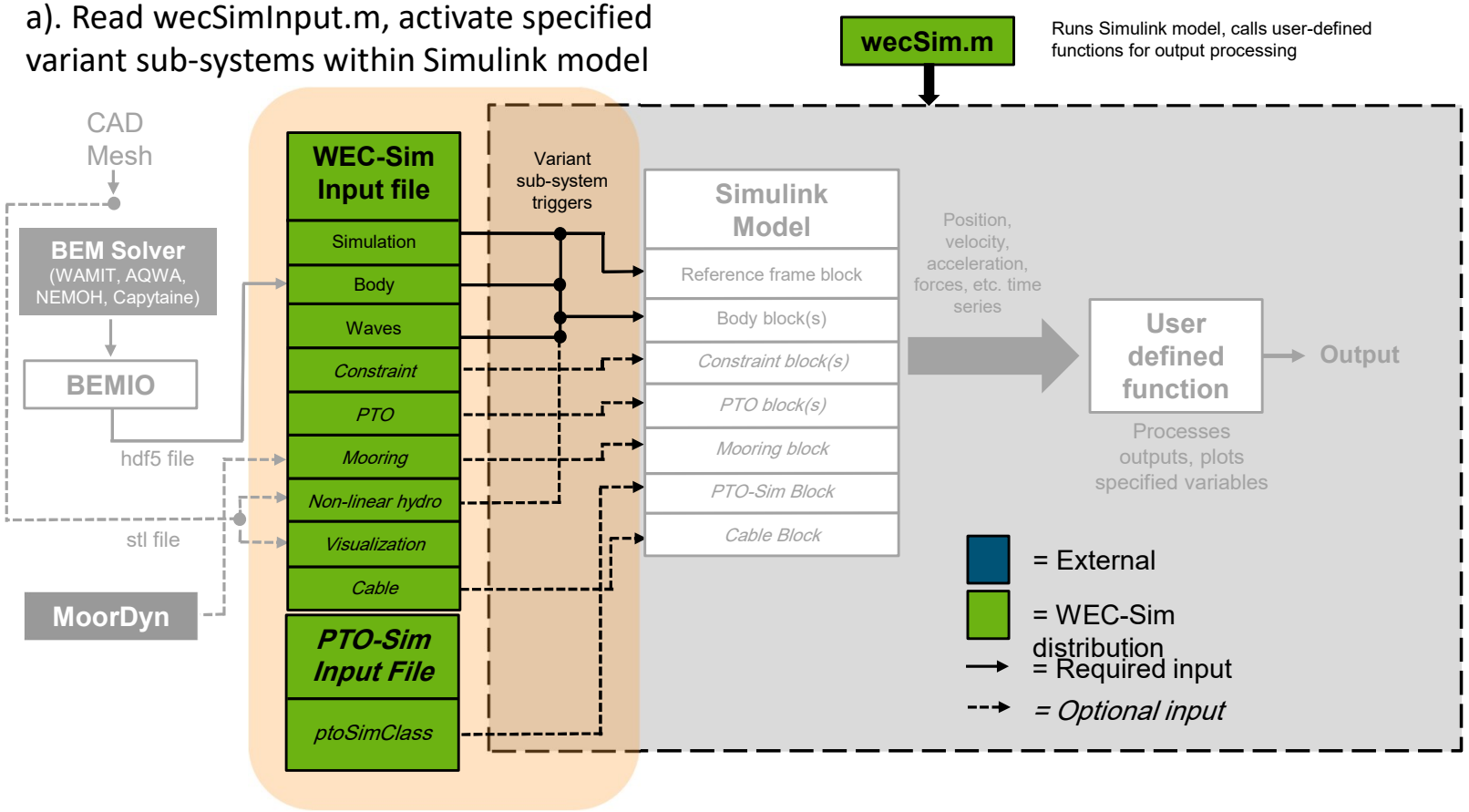
4). Execute wecSim.m



WEC-Sim Workflow

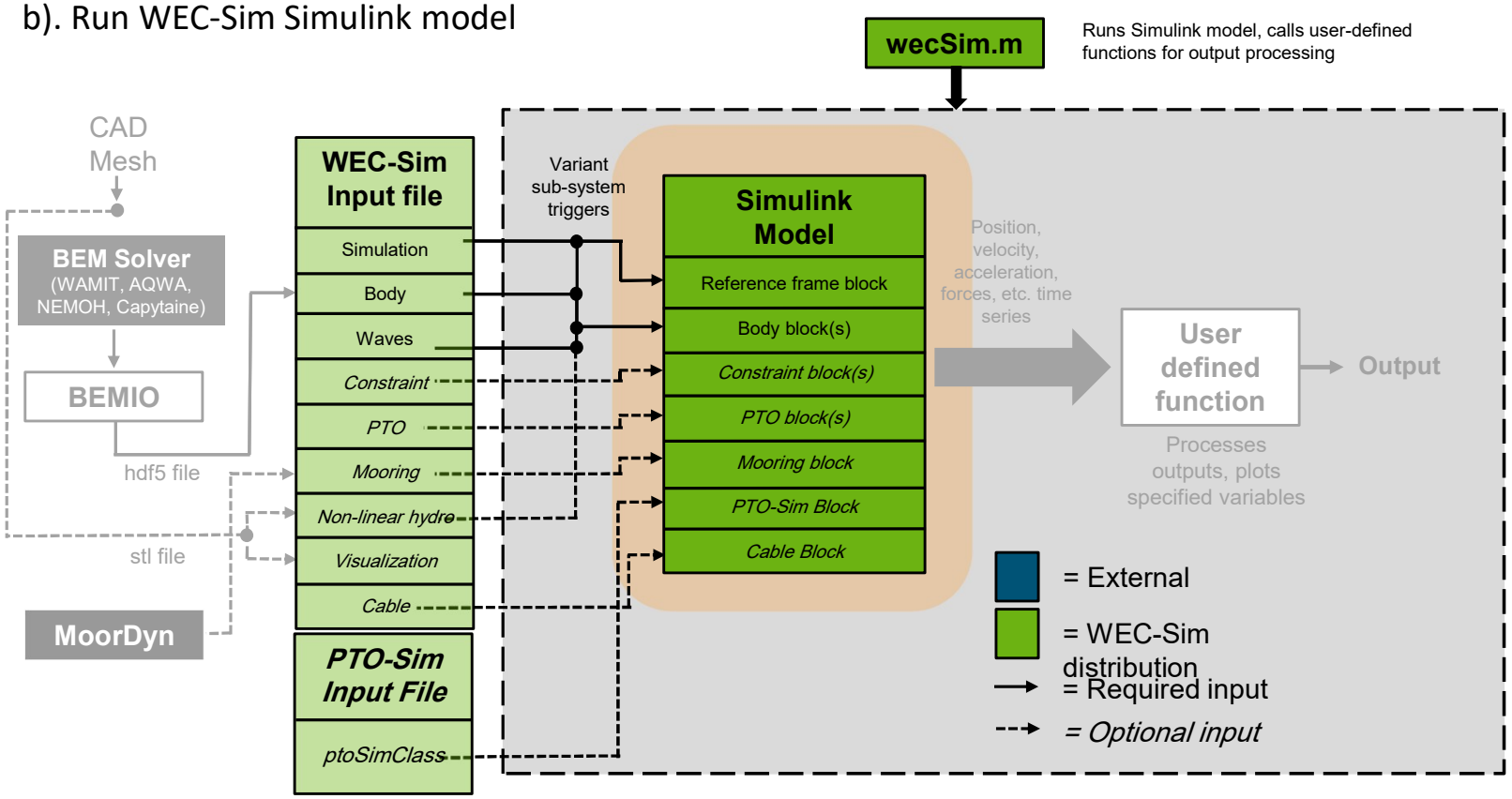
4). Execute wecSim.m

a). Read wecSimInput.m, activate specified variant sub-systems within Simulink model



WEC-Sim Workflow

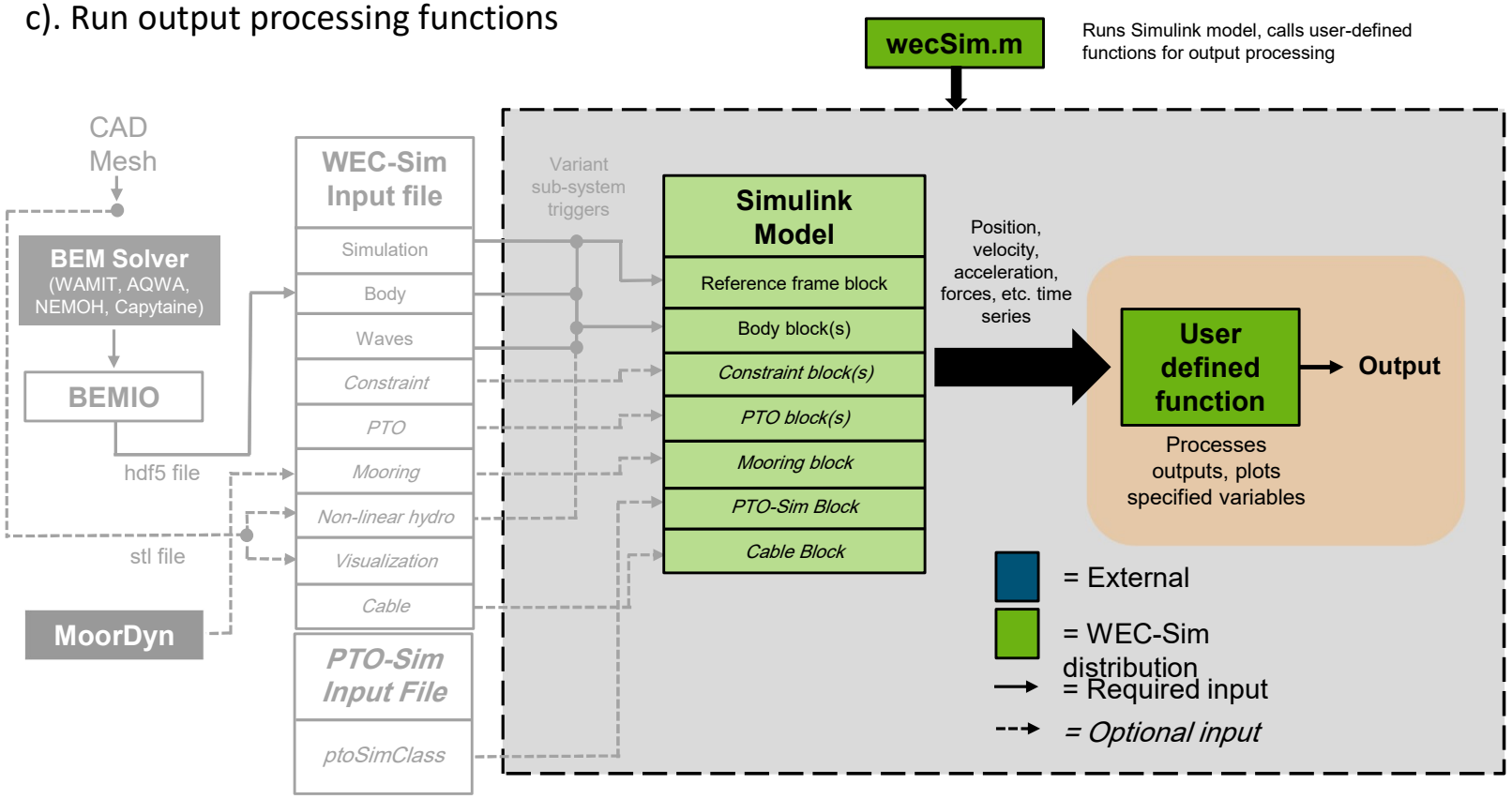
4). Execute wecSim.m
b). Run WEC-Sim Simulink model



WEC-Sim Workflow

4). Execute wecSim.m

c). Run output processing functions



Thank you

For more information please visit the WEC-Sim website:

http://wec-sim.github.io/WEC-Sim

If you have questions on this presentation please reach out to any of the WEC-Sim Developers on GitHub:

https://github.com/WEC-Sim/WEC-Sim

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

This work was authored in part by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308.

Funding provided by the U.S. Department of Energy Office of Energy Efficiency and Renewable Energy Water Power Technologies Office. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.