oFreq

0.4

Generated by Doxygen 1.8.2

Sun Apr 6 2014 08:54:43

Contents

Chapter 1

Summary

Frequency Based Dynamic Analysis

oFreq has two modes of operation. It performs frequency based linear response analysis. Or, it calculates resonant frequencies and mode shapes. The program incorporates user forces and hydrodynamic forces.

Features

Dynamic Response Analysis

The program conducts dynamic response analysis. The program accepts a collection of hydrodynamic and user defined forces, and body mass. These forces are then combined to solve for response motions. Response amplitude and phase are calculated for each individual degree of freedom The program is flexible enough to allow definition of any system of motion, with unlimited degrees of freedom.

Resonant Frequency Analysis

All the features of response analysis, but this time the program solves for resonant frequency. It provides efficient calculation of resonant frequencies and resulting mode shapes for each resonant frequency. Relative response amplitude and relative response forces are presented.

Multiple Body Support

oFreq supports unlimited numbers of body definitions! Each body can have its own location and orientation. Body interactions are possible through hydrodynamic forces or custom user defined forces.

Hydrodynamic Forces Support

oFreq supports hydrodynamic interaction from previous calculations of oHydro. The user simply specifies the path to the oHydro run directory, and oFreq automatically integrates the following hydrodynamic forces.

- Hydrodynamic Active Forces: Forces independent of body responses and frequency. Provides inputs to drive the motion of the body.
- Hydrodynamic Reactive Forces: Forces dependent on body responses. Defined through linear coefficients. Each coefficient can be specified for an individual equation, variable in that equation, and order of derivative associated with that variable. Forces are defined up to second order derivatives. This includes hydrostatic forces, added damping, and added mass.

- Hydrodynamic Cross-Body Forces: Reactive forces that depend on the motion variables of another body. Body coupling are two-way by default, but structure of force definition does allow one-way coupling.

Custom User Forces

The user can define custom forces acting on the body. All forces offer fine level definition down to the individual equation of motion, variable, and order of derivative. Three force types are available.

- User Active Forces: Forces independent of body responses and frequency. Provides inputs to drive the motion of the body.
- User Reactive Forces: Forces dependent on body responses. Defined through linear coefficients. Each coefficient can be specified for an individual equation, variable in that equation, and order of derivative associated with that variable. Unlimited orders of derivative are allowed. Typical uses may only go up to definition for a second order derivative (acceleration), but you can go to higher orders without limit. This is especially useful for modeling active control systems.
- User Cross-Body Forces: Reactive forces that depend on the motion variables of another body. Body coupling can be one-way or two-way. All the features of reactive forces are still available: equation, variable, and derivative specification; unlimited derivatives, etc.

The method of force definition also allows the user to create a library of custom forces and select from individual force sets within the library. Multiple force sets can be selected.

Standard Motion Models

oFreq implements the following standard motion models by default. The user can select a different motion model for each body defined.

- Six degree of freedom.

User Customized Motion Models

Want to do more? oFreq supports up to 25 different motion model definitions. The user can reprogram these motion models to create a custom equation of motion. A working knowledge of C++ programming is required, but the models are isolated from the rest of the program. It is possible to create customized models without reprogramming the entire application.

Custom motion models can access all hydrodynamic forces, user defined forces, and body mass properties. They can be created in any combination desired. The program permits an unlimited number of equations, with any combination of mathematical operators for equation definition. This allows unlimited flexibility. No matter what the system, if you can write the mathematics for it, oFreq can solve the dynamics.

Plethora of Outputs

All outputs are ASCII text files by default. Nothing is hidden in binary files. All information is accessible and reviewable by the user. These files are formatted to easily copy into spreadsheet programs for additional post processing. And there are plenty of outputs to choose from. The program provides the following outputs by default.

- Global motion amplitude and phase for each body
- Global velocity amplitude and phase for each body
- Global acceleration amplitude and phase for each body
- Amplitude and phase of reactive forces for each motion and each order of derivative. (Still in development)

- Wave frequencies, formatted for copying into spreadsheets
- Wave directions, formatted for copying into spreadsheets
- Wave energy spectrum generated, for each direction (Still in development)
- Amplitude and phase of hydrodynamic force for wave amplitudes specified. (Still in development)
- Wave amplitudes for frequencies specified. (Still in development)
- Local motion amplitude and phase for each body. Multiple local locations can be defined. (Still in development)
- Local velocity amplitude and phase for each body. Multiple local locations can be defined. (Still in development)
- Local acceleration amplitude and phase for each body. Multiple local locations can be defined. (Still in development)
- Calculated energy extracted for each degree of freedom. (Still in development)

Chapter 2

Format Standards

Header 1

2.1 Header Comment Standard

/∗-----------------------------------—∗- C++ -∗---—∗\ | O pen | OpenSea: The Open Source Seakeeping Suite | | S eakeeping | Web: www.opensea.dmsonline.us | | E valuation | | | A nalysis | | ∗--—

2.2 CPP File Comment Standard

Format of Coding for .CPP Files

Commenting of Files

Required Text for Every .CPP File

Place the following text at the top of every cpp file. This is a legal protection to ensure we don't get a lawsuit. Otherwise, comment styles are completely free within the file. Please comment heavily to explain your code.

2.3 Object Naming Paradigm

Please use the following object paradigm when naming any new methods in oFreq.

1.) Methods are described as action, followed by subject, all as one word. Example: getObject 2.) The action verb is not capitalized. The subject is capitalized. If the subject is made of multiple words, write it all as one word, capitalizing each word.

3.) Use the following conventions for verbs on objects: get: Retrieve something from the object. Retrieved variable passed by value. ref: Retrieve something from the object. Retrieved variable passed by reference. set: Change some information into the object. Inserted variable passed by value. add: Insert some new information into the object. Usually associated with a vector. list: Access a vector from the object. list(index): Access a specific item in the vector from the object.

2.4 Qt Platform Specific Code

We have one set of source code, which compiles differently, depending on which platform gets selected. Bear this in mind when you write new source code. There are two methods in Qt to change source code at compile time, depending on platform.

Little Pieces of Code

You can specify compiler options which depend on Qt system variables.

For little pieces of code you can simply use the following construct.

> #if defined(Q_OS_WIN32) #elif defined(Q_OS_MACX) #elif etc... #endif

The variables you need are as follows (I included other OS's just for completeness, but we only use linux and windows:

1. Q_OS_WIN32: Any Windows OS. Q_OS_LINUX: Any linuux OS. Q_OS_UNIX: Unix OS. Q_OS_MAX: Mac OS.

All these macros are included as part of the <QtGlobal> header file.

http://qt-project.org/doc/qt-5.0/qtcore/qtglobal.html

Whole files

You can also specify whole individual files to be included or excluded depending on which build is selected.

If you have bigger “implementation details” you can separate things in different cpp files one for each platform i.e. : mycoolwidget.cpp <- common implementation mycoolwidget_win.cpp <- windows specific stuff mycoolwidget_-unix.cpp <- linux/os x stuff if both can use the same code etc. . .

Then in your pro file, use scopes to build the correct set of files. If you need your own defines for a platform theres the DEFINES variable

Chapter 3

Format of Inputs

3.1 Text File Inputs

Why Text File Interface?

A major criticism of ofreq is that is does not natively use a GUI interface. The subject of text file interaces was given careful consideration. There are several reasons why text file interfaces were used. THe first and primary reason is:

- Provide a neutral interface between people with software engineering specialization and people with marine engineering specialization.

Many sections of ofreq require detailed knowledge of marine engineering and seakeeping mathematics. This is knowledge that most software engineers do not normally posses. On the other hand, most marine engineers are not competent enough with programming to handle coding a GUI interface. But the marine engineer can normally handle a text file interface. The text file interfaces provide a neutral format that is easily handled by people in both skill sets: marine engineering and software engineering.

The intent is always to provide a GUI for ofreq. However, this GUI will stand on its own and support the entire OpenSEA software suite.

There are several other reasons for the text file interface.

1. Some advanced users (whole companies) may want to incorporate ofreq into their own calculation software. In that case, the user will create their own interace. A text based interface allows users to develop third-party GUI's without re-coding and recompiling ofreq.
2. Many times users need to extract individual pieces of information from the outputs. The text file interface makes it very easy to extract information and paste into spreadsheet applications.
3. Users may log into remote computers to use this software. Advanced engineering software, such as OpenSEA, is often installed on power machines devoted exclusively for intense computations. Sometimes these remote connections lack a GUI interface. In that case, oFreq can still be used through console only.
4. Eventually, OpenSEA will include a batch program that creates batch runs of these programs and edits the text files for both input and output. A text interface leaves open the most options for that batch program.

For all of these reasons, the text interface is the way I decided to go. It only makes sense because I intend to eventualy supplement this with a GUI interface.

3.2 Input Syntax

Input File Structure

Dynamic Structure

3.3 List of Input Values

Chapter 4

ProgramExecution

Command Line Execution

text

Command Line Options

text

About

text

File Path

text

Chapter 5

File Reader Process

Todo UML File Reader Diagram

Chapter 6

UML Process Diagrams

Header 1

6.1 Overall Process

Todo UML Overall process diagram

6.2 File Writer Process

Todo file writer process diagram. May need to combine this with the Outputs Calculation UML process diagram.

6.3 Motion Model Process

Todo UML Motion Model Process Diagram

6.4 Motion Solver Process

Todo UML Motion Solver Process

6.5 Resonant Solver Process

Todo UML resonant solver process diagram

6.6 Wave Calculations Process

Todo UML wave Calculations process diagram.

create wave calculations code

6.7 Outputs Calculation

Todo UML Outputs Calculation Process Diagram

Chapter 7

Validation

Valication is an important part of the development for software engineering. This is a formalized method of quality control checks to ensure the program works correctly. The formalized method is a two step process:

1. Verification
2. Validation

Verification

Verification is quality control on the equations as implemented within the program. This focuses on two main elements. First, confirming that the equations were actually coded into the program as they should be. A typo can make a large difference for equations.

The second process is to ensure algebraic implementations match the original equations. In engineering software, many of the equations involve differential equations and calculus. However, computers do not have any native understanding of calculus. So, we represent the differential equations as a series of algebraic equations, using a branch of science called `finite difference mathematics`.

The algebraic representations are sometimes very complicated. Verification involves checking each algebraic equation and working backwards to ensure it matches the original differential equation. This is largely a theoretical exercise and not covered in the Developer's Manual. You will find the full details of verification in the Theory Manual.

Validation

Validation essentially checks for bugs in the software, specifically focused on the equations. This is typically done by comparing the software to a series of test cases and comparing results. The following test cases were used for development of ofreq. They start as simple cases to validate basic functionality, and move to progressively more complex cases that systematically test all of ofreq's features.

Single Body Tests

These test focus only on single body performance for ofreq.

Simple Test 1

Simple Test 2

Simple Test 3

Simple Test 4

Multiple Body Tests

These tests repeat all the features tested in the simple version. But now they focus on multiple-bodies present in the analysis.

Multi-body Test 1

Multi-body Test 2

Multi-body Test 3

Multi-body Test 4

Other Tests

Tests that do not fit nicely into any of the other categories, but they are still just as important.

Frequency Variation Test

7.1 Simple Test 1

Test Post test resuls from Simple Test 1.

Purpose

Methodology

Results

Conclusion

7.2 Simple Test 2

Test Run Simple Test 2 and post test results.

Purpose

Methodology

Results

Conclusion

7.3 Simple Test 3

Test Run Simple Test 3 and post test results.

Purpose

Methodology

Results

Conclusion

7.4 Simple Test 4

Test Run Simple Test 4 and post test results.

Purpose

Methodology

Results

Conclusion

7.5 Multi-body Test 1

Test Run Multi-Body Test 1 and post test results.

Purpose

This is like the simple test 1. But it utilizes multiple bodies and ensures all features of multi-body support are working. Includes checking of cross-body forces.

Methodology

Results

Conclusion

7.6 Multi-body Test 2

Test Run Multi-Body Test 2 and post test results.

Purpose

This is like the simple test 2. But it utilizes multiple bodies and ensures all features of multi-body support are working. Includes checking of cross-body forces.

Methodology

Results

Conclusion

7.7 Multi-body Test 3

Test Run Multi-Body Test 3 and post test results.

Purpose

This is like the simple test 3. But it utilizes multiple bodies and ensures all features of multi-body support are working. Includes checking of cross-body forces.

Methodology

Results

Conclusion

7.8 Multi-body Test 4

Test Run Multi-Body Test 4 and post test results.

Purpose

This is like the simple test 4. But it utilizes multiple bodies and ensures all features of multi-body support are working. Includes checking of cross-body forces.

Methodology

Results

Conclusion

7.9 Frequency Variation Test

Test Run Frequency Variation Test and post test results.

Purpose

Methodology

Results

Conclusion

Chapter 8

Test List

Page Frequency Variation Test
Run Frequency Variation Test and post test results.

Page Multi-body Test 1
Run Multi-Body Test 1 and post test results.

Page Multi-body Test 2
Run Multi-Body Test 2 and post test results.

Page Multi-body Test 3
Run Multi-Body Test 3 and post test results.

Page Multi-body Test 4
Run Multi-Body Test 4 and post test results.

Page Simple Test 1
Post test resuls from Simple Test 1.

Page Simple Test 2
Run Simple Test 2 and post test results.

Page Simple Test 3
Run Simple Test 3 and post test results.

Page Simple Test 4
Run Simple Test 4 and post test results.

Chapter 9

Todo List

Page File Reader Process

UML File Reader Diagram

Page File Writer Process

file writer process diagram. May need to combine this with the Outputs Calculation UML process diagram.

Page Motion Model Process

UML Motion Model Process Diagram

Page Motion Solver Process

UML Motion Solver Process

Page Outputs Calculation

UML Outputs Calculation Process Diagram

Page Overall Process

UML Overall process diagram

Page Resonant Solver Process

UML resonant solver process diagram

Page Wave Calculations Process

UML wave Calculations process diagram.

create wave calculations code

Chapter 10

Namespace Index

10.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 11

Hierarchical Index

11.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 12

Class Index

12.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 13

File Index

13.1 File List

Here is a list of all files with brief descriptions:

Chapter 14

Namespace Documentation

14.1 osea Namespace Reference

Namespaces

- namespace ofreq

Classes

- class Dictionary
- class FileReader
- class ObjectGroup

  The ObjectGroup class contains groupings of object definitions captured from an input file. It is a data container to hold the segmented input file for interpretation. The container contains three things: 1.) Object class name (as specified by input file) 2.) Vector of keyword names 3.) Vector of keyword values. Each entry in the vector of values is also a vector. This allows the definition of lists. A list will be as long as it needs to be for specification of all values in the list. The index of the value is specified by its position in the vector list. The value is the entry.
- class Parser

  The Parser class takes an input segment of strings and segments that segment. It strips out comments. It recognizes quotation marks and groups those segments together. Parser finally returns a series of ObjectGroup objects. Each ObjectGroup contains the classname and a list of keyword value pairs.

Typedefs

- typedef std::vector
  $<$ std::vector$<$ std::string $>$ $>$ vecValue

  Type definition used to store key values. Must be a vector of vectors because a value may also be a list of values.
- typedef std::vector$<$ std::string $>$ vecKeyword

  Type defintion used to store keywords.
- typedef std::vector
  $<$ ObjectGroup $*$ $>$ vecObject

14.1.1 Detailed Description

The namespace for all code created under the OpenSEA project. There are also several sub-namespaces, one associated with each primary program under osea. 1.) ohydro: Code associated with the program ohydro. 2.) ofreq: Code associated with the program ofreq. 3.) otime: Code associated with the program otime. 4.) ofourier: Code associated with the program ofourier. 5.) obatch: Code associated with the program obatch. 6.) guisea: Code assocaited with the GUI that interacts with all OpenSEA programs. Any code that may have common utility amongst all programs, such as file reading objects, goes under the generic osea namespace. Any code that is only useful within the specific program it serves, goes under the specific namespace. When in doubt, default to just the osea namespace.

The namespaces are not intended to create an organizational structure. They are only intended to prevent name conflicts.

14.1.2 Typedef Documentation

14.1.2.1 typedef std::vector< std::string > **osea::vecKeyword**

Type defintion used to store keywords.

Definition at line 83 of file objectgroup.h.

14.1.2.2 typedef std::vector<**ObjectGroup**∗> **osea::vecObject**

Definition at line 85 of file objectgroup.h.

14.1.2.3 typedef std::vector< std::vector< std::string > > **osea::vecValue**

Type definition used to store key values. Must be a vector of vectors because a value may also be a list of values.

Definition at line 70 of file objectgroup.h.

14.2 osea::ofreq Namespace Reference

Classes

- class GlobalAcceleration
- class GlobalMotion
- class GlobalSolution
- class GlobalVelocity
- class OutputDerived
- class OutputsBody
- class dictBodies
- class dictControl
- class dictForces
- class FileWriter
- class Body
- class Derivative
- class Equation
- class Force
- class ForceActive
- class ForceCross
- class ForceReact
- class oFreqCore

    The core oFreq class. All oFreq classes inherit from this class.
- class Solution
- class SolutionSet
- class System
- class EqnRotation

    The EqnRotation class.

- class EqnTranslation

  The EqnTranslation class.
- class EquationofMotion
- class Model6DOF

  The motion model for standard six-degree of freedom rigid-body dynamics problems.
- class MotionModel
- class matBody
- class matForceActive
- class matForceCross
- class matForceReact
- class MotionSolver

Typedefs

- typedef std::complex< double > complexDouble
- typedef std::vector
  < std::complex< double > > cx_vector

  Type definition for a vector of complex numbers. Used to return the calculated output.
- typedef std::vector< Solution ∗ > ptSoln

14.2.1 Detailed Description

The namespace of all code specifically associated with ofreq.

14.2.2 Typedef Documentation

14.2.2.1 typedef std::complex< double > **osea::ofreq::complexDouble**

Simple typedef for complexDouble represents std::complex<double>

Definition at line 82 of file globalsolution.h.

14.2.2.2 typedef std::vector< std::complex<double> > **osea::ofreq::cx_vector**

Type definition for a vector of complex numbers. Used to return the calculated output.

Definition at line 93 of file outputsbody.h.

14.2.2.3 typedef std::vector<**Solution**∗> **osea::ofreq::ptSoln**

Type definition for a vector of pointers to Solution object.

Definition at line 82 of file solutionset.h.

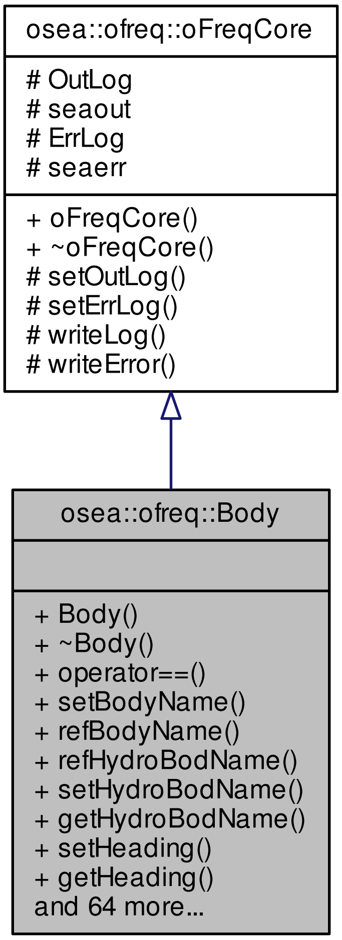
Chapter 15

Class Documentation

15.1 osea::ofreq::Body Class Reference

```
#include <body.h>
```

Inheritance diagram for osea::ofreq::Body:



Public Member Functions

- Body ()
- ∼Body ()
- bool operator== (Body &bodIn)

    Overload for operator == to compare two Body objects. Comparison is based on body names.
- void setBodyName (std::string)
- std::string & refBodyName ()

    Exposes the body name property for operation.
- std::string & refHydroBodName ()

    Exposes the hydro body name property for operation.
- void setHydroBodName (std::string)
- std::string getHydroBodName ()
- void setHeading (double)
- double getHeading ()

    Gets the heading for the body. Heading is measured in radians. Zero heading is True North, proceeding counter clockwise around the compass rose.

- double & refHeading ()

    Exposes the heading property for operations.
- void setMass (double)
- double getMass ()

    Returns the mass of the body.
- void setMomIxx (double)
- double getMomIxx ()

    Returns the mass moment of inertia on the XX axis for the body.
- void setMomIyy (double)
- double getMomIyy ()

    Returns the mass moment of inertia on the YY axis for the body.
- void setMomIzz (double)
- double getMomIzz ()

    Returns the mass moment of inertia on the ZZ axis for the body.
- void setMomIxy (double)
- double getMomIxy ()

    Returns the cross product of intertia on the XY axis for the body.
- void setMomIxz (double)
- double getMomIxz ()

    Returns the cross product of intertia on the XZ axis for the body.
- void setMomIyz (double)
- double getMomIyz ()

    Returns the cross product of intertia on the YZ axis for the body.
- void setCenX (double)
- arma::Mat< double > getMassMatrix ()

    Gets the mass matrix for the body.
- arma::Mat< double > & MassMatrix ()

    Implements the method getMassMatrix(), just under a different name.
- void setMassMatrix (arma::Mat< double > MassMatIn)

    Set the mass matrix for the body.
- double getCenX ()

    Returns the centroid of the body mass, X-axis.
- void setCenY (double)
- double getCenY ()

    Returns the centroid of the body mass, Y-axis.
- void setCenZ (double)
- double getCenZ ()

    Returns the centroid of the body mass, Z-axis.
- arma::Mat< double > getCen ()

    Returns the entire mass centroid matrix.
- void setPosnX (double input)

    Sets the body position in the X-axis.
- double getPosnX ()

    Gets the body position in the X-axis.
- void setPosnY (double input)

    Sets the body position in the Y-axis.
- double getPosnY ()

    Gets the body position in the Y-axis.
- void setPosnZ (double input)

    Sets the body position in the Z-axis.
- double getPosnZ ()

Gets the body position in the Z-axis.

- arma::Mat< double > getPosn ()

  Returns the entire matrix for position of the body.
- arma::Mat< double > & refPosn ()

  Exposes the position property for operation. The entire matrix for position of the body.
- std::string getBodyName ()
- void setSolnMat (arma::cx_mat input)

  Set the solution matrix for the body.
- arma::cx_mat getSolution ()

  Get the solution matrix for the body.
- arma::cx_mat & refSolution ()

  Get the solution matrix for the body.
- arma::cx_mat & refDataSolution ()

  The same things as the refSolution() function, just under a different name.
- complexDouble & refDataSolution (int varIndexIn)

  Returns a single solution value, based on the variable requested.
- Body Copy ()

  Copies the body object, complete with all current data.
- std::vector< ForceActive ∗ > & listForceActive_user ()

  The list of active user forces.
- ForceActive ∗ listForceActive_user (int forceIn)

  A single active user force.
- std::vector< ForceActive ∗ > & listForceActive_hydro ()

  The list of active hydrodynamic forces.
- ForceActive ∗ listForceActive_hydro (int forceIn)

  A single active hydrodynamic force.
- std::vector< ForceReact ∗ > & listForceReact_user ()

  The list of reactive user forces.
- ForceReact ∗ listForceReact_user (int forceIn)

  A single reactive user force.
- std::vector< ForceReact ∗ > & listForceReact_hydro ()

  The list of reactive hydrodynamic forces.
- ForceReact ∗ listForceReact_hydro (int forceIn)

  A single reactive hydrodynamic force.
- std::vector< ForceCross ∗ > & listForceCross_user ()

  The list of user cross-body forces.
- ForceCross ∗ listForceCross_user (int forceIn)

  A single cross-body user force.
- std::vector< ForceCross ∗ > & listForceCross_hydro ()

  The list of hydrodynamic cross-body forces.
- ForceCross ∗ listForceCross_hydro (int forceIn)

  A single cross-body hydrodynamic force.
- std::vector< Body ∗ > & listCrossBody_user ()

  The list of linked bodies for user cross-body forces.
- Body & listCrossBody_user (int index)

  Returns reference to individual linked Body for the user cross-body force.
- std::vector< Body ∗ > & listCrossBody_hydro ()

  The list of linked bodies for hydrodynamic cross-body forces.
- Body & listCrossBody_hydro (int index)

  Returns reference to individual linked Body for the hydro cross-body force.
- std::vector< std::string > & listNamedLink_user ()

The list of names of linked bodies for user cross-body forces. This is a list of names of other bodies that a cross-body force references. This corresponds to the vector listForceCross_usr. The indices of the two vectors should match. So that when a force gets added at index 5 in the listForceCross_user, it should have a matching entry at index 5 in listNamedLink_usr. The list of names only is a temporary list used during the input stage of bodies. This is required because the linked body may name a body which is not yet read from the input file. Thus, the body is not currently defined. Once all Bodies are defined, the System object calls a function to read through each name in the list and assign corresponding pointers in the listLinkedBody_usr.

- std::string & listNamedLink_user (unsigned int varIn)

  The list of names of linked bodies for user cross-body forces. This is a list of names of other bodies that a cross-body force references. This corresponds to the vector listForceCross_usr. The indices of the two vectors should match. So that when a force gets added at index 5 in the listForceCross_user, it should have a matching entry at index 5 in listNamedLink_usr. The list of names only is a temporary list used during the input stage of bodies. This is required because the linked body may name a body which is not yet read from the input file. Thus, the body is not currently defined. Once all Bodies are defined, the System object calls a function to read through each name in the list and assign corresponding pointers in the listLinkedBody_usr.

- std::vector< std::string > & listNamedLink_hydro ()

  The list of names of linked bodies for hydro cross-body forces. This is a list of names of other bodies that a cross-body force references. This corresponds to the vector listForceCross_hydro. The indices of the two vectors should match. So that when a force gets added at index 5 in the listForceCross_hydro, it should have a matching entry at index 5 in listNamedLink_hydro. The list of names only is a temporary list used during the input stage of bodies. This is required because the linked body may name a body which is not yet read from the input file. Thus, the body is not currently defined. Once all Bodies are defined, the System object calls a function to read through each name in the list and assign corresponding pointers in the listLinkedBody_hydro.

- std::string & listNamedLink_hydro (unsigned int varIn)

  The list of names of linked bodies for hydro cross-body forces. This is a list of names of other bodies that a cross-body force references. This corresponds to the vector listForceCross_hydro. The indices of the two vectors should match. So that when a force gets added at index 5 in the listForceCross_hydro, it should have a matching entry at index 5 in listNamedLink_hydro. The list of names only is a temporary list used during the input stage of bodies. This is required because the linked body may name a body which is not yet read from the input file. Thus, the body is not currently defined. Once all Bodies are defined, the System object calls a function to read through each name in the list and assign corresponding pointers in the listLinkedBody_hydro.

- void setMotionModel (ofreq::MotionModel &modelIn)

  Sets the motion model for lookup later.

- ofreq::MotionModel & getMotionModel ()

  Gets the motion model.

- int getEquationCount ()

  Gets the number of equations used in the body.

- void initMassMat ()

  Initializes the mass matrix. Resizes it to the correct value. Only acts if the motion model is already set. And does not override any current values of the mass matrix.

Additional Inherited Members

15.1.1 Detailed Description

This class holds all of the data for the Body Input File.

Definition at line 103 of file body.h.

15.1.2 Constructor & Destructor Documentation

15.1.2.1 Body::Body ( )

The default constructor

Definition at line 44 of file body.cpp.

15.1.2.2 Body::∼Body ( )

The default destructor, nothing happens here.

Definition at line 53 of file body.cpp.

15.1.3 Member Function Documentation

15.1.3.1 Body Body::Copy ( )

Copies the body object, complete with all current data.

Returns

Returns a copy of the body object, complete with all current data. Passed by value, not reference.

Definition at line 730 of file body.cpp.

15.1.3.2 string Body::getBodyName ( )

Get the name of the body.

Returns

The name of the body.

Definition at line 690 of file body.cpp.

15.1.3.3 Mat< double > Body::getCen ( )

Returns the entire mass centroid matrix.

Returns

Returns the entire mass centroid matrix. Output is a 3x1 matrix of the body centroid, relative to body coordinate system.

Definition at line 623 of file body.cpp.

15.1.3.4 double Body::getCenX ( )

Returns the centroid of the body mass, X-axis.

Returns the centroid of the body mass, X-axis. Centroid is relative to body coordinates. This includes body rotation and translation.

Returns

Returns the centroid of the body mass, X-axis. Centroid is relative to body coordinates. This includes body rotation and translation.

Definition at line 587 of file body.cpp.

15.1.3.5 double Body::getCenY ( )

Returns the centroid of the body mass, Y-axis.

Returns the centroid of the body mass, Y-axis. Centroid is relative to body coordinates. This includes body rotation and translation.

Returns

Returns the centroid of the body mass, Y-axis. Centroid is relative to body coordinates. This includes body rotation and translation.

Definition at line 602 of file body.cpp.

15.1.3.6 double Body::getCenZ ( )

Returns the centroid of the body mass, Z-axis.

Returns the centroid of the body mass, Z-axis. Centroid is relative to body coordinates. This includes body rotation and translation.

Returns

Returns the centroid of the body mass, Z-axis. Centroid is relative to body coordinates. This includes body rotation and translation.

Definition at line 617 of file body.cpp.

15.1.3.7 int Body::getEquationCount ( )

Gets the number of equations used in the body.

Gets the number of equations used in the body.

Returns

Integer number representing the number of equations used in the body.

Definition at line 999 of file body.cpp.

15.1.3.8 double Body::getHeading ( )

Gets the heading for the body. Heading is measured in radians. Zero heading is True North, proceeding counter clockwise around the compass rose.

Returns

Returns double variable. Heading of the Body object. Variable passed by value.

Definition at line 193 of file body.cpp.

15.1.3.9 string Body::getHydroBodName ( )

Gets the name of the hydro body.

Returns

Returns std::string. The name of the hydrobody object associated with the body. Variable passed by value.

Definition at line 181 of file body.cpp.

15.1.3.10 double Body::getMass ( )

Returns the mass of the body.

Returns

Returns the mass of the body. Output is in units of kilograms.

Definition at line 253 of file body.cpp.

15.1.3.11 Mat< double > Body::getMassMatrix ( )

Gets the mass matrix for the body.

Returns

Returns the mass matrix for the body, as a single matrix. Returned by value.

Definition at line 548 of file body.cpp.

15.1.3.12 double Body::getMomIxx ( )

Returns the mass moment of inertia on the XX axis for the body.

Returns

Returns the mass moment of inertia on the XX axis for the body. Output is in units of kilogram-m^2.

Definition at line 311 of file body.cpp.

15.1.3.13 double Body::getMomIxy ( )

Returns the cross product of intertia on the XY axis for the body.

Returns

Returns the cross product of intertia on the XY axis for the body. OUtput is in units of kilogram-m^2.

Definition at line 438 of file body.cpp.

15.1.3.14 double Body::getMomIxz ( )

Returns the cross product of intertia on the XZ axis for the body.

Returns

Returns the cross product of intertia on the XZ axis for the body. OUtput is in units of kilogram-m^2.

Definition at line 482 of file body.cpp.

15.1.3.15 double Body::getMomIyy ( )

Returns the mass moment of inertia on the YY axis for the body.

Returns

Returns the mass moment of inertia on the YY axis for the body. Output is in units of kilogram-m^2.

Definition at line 353 of file body.cpp.

15.1.3.16 double Body::getMomIyz ( )

Returns the cross product of intertia on the YZ axis for the body.

Returns

Returns the cross product of intertia on the YZ axis for the body. OUtput is in units of kilogram-m^2.

Definition at line 526 of file body.cpp.

15.1.3.17 double Body::getMomIzz ( )

Returns the mass moment of inertia on the ZZ axis for the body.

Returns

Returns the mass moment of inertia on the ZZ axis for the body. Output is in units of kilogram-m^2.

Definition at line 395 of file body.cpp.

15.1.3.18 MotionModel & Body::getMotionModel ( )

Gets the motion model.

Returns the motion model object used by this body object.

Returns

Returns MotinModel object. Variable passed by reference.

Definition at line 993 of file body.cpp.

15.1.3.19 Mat< double > Body::getPosn ( )

Returns the entire matrix for position of the body.

Returns the entire matrix for the position of the body. Output is a 3x1 matrix with double point precision. First entry (1,1) = Position in X-axis. Second entry (2,1) = Position in Y-axis. Third entry (3,1) = Position in Z-axis. Units are in meters. Position is relative to the orientation of the world coordinate system.

Returns

Returns the entire matrix for the position of the body. Output is a 3x1 matrix with double point precision. First entry (1,1) = Position in X-axis. Second entry (2,1) = Position in Y-axis. Third entry (3,1) = Position in Z-axis. Units are in meters. Position is relative to the orientation of the world coordinate system.

Definition at line 674 of file body.cpp.

15.1.3.20 double Body::getPosnX ( )

Gets the body position in the X-axis.

Gets the body position in the X-axis. Position is set relative to the world coordinate system. Units are in meters.

Returns

Double precision floating number specifying the position on the X-axis, units of meters.

Definition at line 638 of file body.cpp.

15.1.3.21 double Body::getPosnY ( )

Gets the body position in the Y-axis.

Gets the body position in the Y-axis. Position is set relative to the world coordinate system. Units are in meters.

Returns

Double precision floating number specifying the position on the Y-axis, units of meters.

Definition at line 653 of file body.cpp.

15.1.3.22 double Body::getPosnZ ( )

Gets the body position in the Z-axis.

Gets the body position in the Z-axis. Position is set relative to the world coordinate system. Units are in meters.

Returns

Double precision floating number specifying the position on the Z-axis, units of meters.

Definition at line 668 of file body.cpp.

15.1.3.23 cx_mat Body::getSolution ( )

Get the solution matrix for the body.

Gets the solution matrix for the body. Used to store the solution from the motion solver. This variable is initially empty on body creation. It gets filled with the output from the motion solver. Output is a column matrix (n by 1) of complex numbers. Output is in units of meters.

Returns

Column matrix of complex numbers. Matrix size is not hard coded. Number of rows in matrix must match number of equations for body property.

Definition at line 702 of file body.cpp.

15.1.3.24 void Body::initMassMat ( )

Initializes the mass matrix. Resizes it to the correct value. Only acts if the motion model is already set. And does not override any current values of the mass matrix.

Definition at line 736 of file body.cpp.

15.1.3.25 vector< Body * > & Body::listCrossBody_hydro ( )

The list of linked bodies for hydrodynamic cross-body forces.

The list of linked bodies for hydrodynamic cross-body forces. This is a list of pointers to the other bodies. This corresponds with the vector listForceCross_usr. The indices of the two vectors should match. The indices of the two lists should match. So that when a force gets added at index 5 in the listForceCross_hydro, it should have a matching entry at index 5 in the listLinkedBody_hydro.

Returns

A list of pointers to various linked bodies for hydro cross-body forces.

Definition at line 939 of file body.cpp.

15.1.3.26 Body & Body::listCrossBody_hydro (int *index*)

Returns reference to individual linked Body for the hydro cross-body force.

Returns reference for linked Body specified by the index. The index corresponds to the index of the cross-body force. So that when a cross-body force is stored in its list at index 5, the linked Body can be retrieved from this method with index 5. Body stored internally as a pointer to the Body object.

Parameters

index	Integer. The index of the linked Body to return.

Returns

Returns reference to a Body object. The reference points to the linked Body object that corresponds to the ForceCross object at the same index.

Definition at line 945 of file body.cpp.

15.1.3.27 vector< Body * > & Body::listCrossBody_user ( )

The list of linked bodies for user cross-body forces.

The list of linked bodies for user cross-body forces. This is a list of pointers to the other bodies. This corresponds with the vector listForceCross_usr. The indices of the two vectors should match. The indices of the two lists should match. So that when a force gets added at index 5 in the listForceCross_usr, it should have a matching entry at index 5 in the listLinkedBody_usr.

Returns

A list of pointers to various linked bodies for user cross-body forces.

Definition at line 927 of file body.cpp.

15.1.3.28 Body & Body::listCrossBody_user (int *index*)

Returns reference to individual linked Body for the user cross-body force.

Returns reference for linked Body specified by the index. The index corresponds to the index of the cross-body force. So that when a cross-body force is stored in its list at index 5, the linked Body can be retrieved from this method with index 5. Body stored internally as a pointer to the Body object.

Parameters

index	Integer. The index of the linked Body to return.

Returns

Returns reference to a Body object. The reference points to the linked Body object that corresponds to the ForceCross object at the same index.

Definition at line 933 of file body.cpp.

15.1.3.29 vector< ForceActive * > & Body::listForceActive_hydro ( )

The list of active hydrodynamic forces.

The list of active hydrodynamic forces. A vector of pointers directing to the active hydrodynamic forces. Warning that these forces may be linked to other bodies as well and should not be changed.

Returns

A vector of pointes to various hydrodynamic active forces.

Definition at line 793 of file body.cpp.

15.1.3.30 ForceActive ∗ Body::listForceActive_hydro (int *forceIn*)

A single active hydrodynamic force.

A single active hydrodynamic force. A pointer directing to the active hydrodynamic force. Warning that these forces may be linked to other bodies as well and should not be changed.

Parameters

forceIn	Integer. Index of the ForceActive object requested.

Returns

A single pointer to the user active forces requested by parameter forceIn. Pointer passed by value.

Definition at line 799 of file body.cpp.

15.1.3.31 vector< ForceActive ∗ > & Body::listForceActive_user ( )

The list of active user forces.

The list of active user forces. A vector of pointers directing to the active user forces. Warning that these forces may be linked to other bodies as well and should not be changed.

Returns

A vector of pointers to various user active forces.

Definition at line 767 of file body.cpp.

15.1.3.32 ForceActive ∗ Body::listForceActive_user (int *forceIn*)

A single active user force.

A single active user force. A pointer directing to the active user force. Warning that these forces may be linked to other bodies as well and should not be changed.

Parameters

forceIn	Integer. Index of the ForceActive object requested.

Returns

A single pointer to the user active forces requested by parameter forceIn. Pointer passed by value.

Definition at line 773 of file body.cpp.

15.1.3.33 vector< ForceCross ∗ > & Body::listForceCross_hydro ( )

The list of hydrodynamic cross-body forces.

The list of hydrodynamic cross-body forces. A vector of pointers directing to the hydrodynamic cross-body forces. Warning that these forces may be linked to other bodies as well and should not be changed. There is another vector: the listLinkedBody_usr. That determines which body each cross-body force links to. The indices of the two lists should match. So that when a force gets added at index 5 in the listForceCross_hydro, it should have a matching entry at index 5 in the listLinkedBody_hydro.

Returns

A list of pointers to various hydrodynamic cross-body forces.

Definition at line 897 of file body.cpp.

15.1.3.34 ForceCross ∗ Body::listForceCross_hydro (int *forceIn*)

A single cross-body hydrodynamic force.

A single cross-body hydrodynamic force. A pointer directing to the cross-body hydrodynamic force. Warning that these forces may be linked to other bodies as well and should not be changed.

Parameters

forceIn	Integer. Index of the ForceCross object requested.

Returns

A single pointer to the user cross-body forces requested by parameter forceIn. Pointer passed by value.

Definition at line 903 of file body.cpp.

15.1.3.35 vector< ForceCross ∗ > & Body::listForceCross_user ( )

The list of user cross-body forces.

The list of user cross-body forces. A vector of pointers directing to the user cross-body forces. Warning that these forces may be linked to other bodies as well and should not be changed. There is another vector: the listLinkedBody_usr. That determines which body each cross-body force links to. The indices of the two lists should match. So that when a force gets added at index 5 in the listForceCross_usr, it should have a matching entry at index 5 in the listLinkedBody_usr.

Returns

A list of pointers to various user cross-body forces.

Definition at line 871 of file body.cpp.

15.1.3.36 ForceCross ∗ Body::listForceCross_user (int *forceIn*)

A single cross-body user force.

A single cross-body user force. A pointer directing to the cross-body user force. Warning that these forces may be linked to other bodies as well and should not be changed.

Parameters

forceIn	Integer. Index of the ForceCross object requested.

Returns

A single pointer to the user cross-body forces requested by parameter forceIn. Pointer passed by value.

Definition at line 877 of file body.cpp.

15.1.3.37 vector< ForceReact * > & Body::listForceReact_hydro ( )

The list of reactive hydrodynamic forces.

The list of reactive hydrodynamic forces. A vector of pointers directing to the reactive hydrodynamic forces. Warning that these forces may be linked to other bodies as well and should not be changed.

Returns

A vector of pointers to various hydrodynamic reactive forces.

Definition at line 865 of file body.cpp.

15.1.3.38 ForceReact * Body::listForceReact_hydro (int *forceIn*)

A single reactive hydrodynamic force.

A single reactive hydrodynamic force. A pointer directing to the reactive hydrodynamic force. Warning that these forces may be linked to other bodies as well and should not be changed.

Parameters

forceIn	Integer. Index of the ForceReact object requested.

Returns

A single pointer to the user reactive forces requested by parameter forceIn. Pointer passed by value.

Definition at line 845 of file body.cpp.

15.1.3.39 vector< ForceReact * > & Body::listForceReact_user ( )

The list of reactive user forces.

The list of reactive user forces. A vector of pointers directing to the reactive user forces. Warning that these forces may be linked to other bodies as well and should not be changed.

Returns

A vector of pointers to various user reactive forces.

Definition at line 819 of file body.cpp.

15.1.3.40 ForceReact * Body::listForceReact_user (int *forceIn*)

A single reactive user force.

A single reactive user force. A pointer directing to the reactive user force. Warning that these forces may be linked to other bodies as well and should not be changed.

Parameters

forceIn	Integer. Index of the ForceReact object requested.

Returns

A single pointer to the user reactive forces requested by parameter forceIn. Pointer passed by value.

Definition at line 825 of file body.cpp.

15.1.3.41 vector< string > & Body::listNamedLink_hydro ( )

The list of names of linked bodies for hydro cross-body forces. This is a list of names of other bodies that a cross-body force references. This corresponds to the vector listForceCross_hydro. The indices of the two vectors should match. So that when a force gets added at index 5 in the listForceCross_hydro, it should have a matching entry at index 5 in listNamedLink_hydro. The list of names only is a temporary list used during the input stage of bodies. This is required because the linked body may name a body which is not yet read from the input file. Thus, the body is not currently defined. Once all Bodies are defined, the System object calls a function to read through each name in the list and assign corresponding pointers in the listLinkedBody_hydro.

Returns

Returns the list of named bodies linked to the Cross-Body forces. Returned object is a vector of std::string objects. Returned variable passed by reference.

See Also

listLinkedBody_hydro()
System

Definition at line 969 of file body.cpp.

15.1.3.42 string & Body::listNamedLink_hydro (unsigned int *varIn*)

The list of names of linked bodies for hydro cross-body forces. This is a list of names of other bodies that a cross-body force references. This corresponds to the vector listForceCross_hydro. The indices of the two vectors should match. So that when a force gets added at index 5 in the listForceCross_hydro, it should have a matching entry at index 5 in listNamedLink_hydro. The list of names only is a temporary list used during the input stage of bodies. This is required because the linked body may name a body which is not yet read from the input file. Thus, the body is not currently defined. Once all Bodies are defined, the System object calls a function to read through each name in the list and assign corresponding pointers in the listLinkedBody_hydro.

Parameters

varIn	Integer input specifying exactly which item in the list to return.

Returns

Returns the named body linked to the Cross-Body forces. Returned object is a std::string object. Returned variable passed by reference.

See Also

listLinkedBody_hydro()
System

Definition at line 975 of file body.cpp.

15.1.3.43 vector< string > & Body::listNamedLink_user ( )

The list of names of linked bodies for user cross-body forces. This is a list of names of other bodies that a cross-body force references. This corresponds to the vector listForceCross_usr. The indices of the two vectors should match. So that when a force gets added at index 5 in the listForceCross_user, it should have a matching entry at index 5 in listNamedLink_usr. The list of names only is a temporary list used during the input stage of bodies. This is required because the linked body may name a body which is not yet read from the input file. Thus, the body is not currently defined. Once all Bodies are defined, the System object calls a function to read through each name in the list and assign corresponding pointers in the listLinkedBody_usr.

Returns

Returns the list of named bodies linked to the Cross-Body forces. Returned object is a vector of std::string objects. Returned variable passed by reference.

See Also

listLinkedBody_user()
System

Definition at line 951 of file body.cpp.

15.1.3.44 string & Body::listNamedLink_user (unsigned int *varIn*)

The list of names of linked bodies for user cross-body forces. This is a list of names of other bodies that a cross-body force references. This corresponds to the vector listForceCross_usr. The indices of the two vectors should match. So that when a force gets added at index 5 in the listForceCross_user, it should have a matching entry at index 5 in listNamedLink_usr. The list of names only is a temporary list used during the input stage of bodies. This is required because the linked body may name a body which is not yet read from the input file. Thus, the body is not currently defined. Once all Bodies are defined, the System object calls a function to read through each name in the list and assign corresponding pointers in the listLinkedBody_usr.

Parameters

varIn	Integer input specifying exactly which item in the list to return.

Returns

Returns the named body linked to the Cross-Body forces. Returned object is a std::string object. Returned variable passed by reference.

See Also

listLinkedBody_user()
System

Definition at line 957 of file body.cpp.

15.1.3.45 Mat< double > & Body::MassMatrix ( )

Implements the method getMassMatrix(), just under a different name.

Returns

Returns the mass matrix for the body, as a single matrix.

Definition at line 557 of file body.cpp.

15.1.3.46 bool Body::operator== (Body & *bodIn*)

Overload for operator == to compare two Body objects. Comparison is based on body names.

Parameters

bodIn	The other body to compare to.

Returns

Returns true if the body names are equal. Returned variable is passed by value.

Definition at line 58 of file body.cpp.

15.1.3.47 string & Body::refBodyName ( )

Exposes the body name property for operation.

Returns

Pointer to the body name property.

Definition at line 163 of file body.cpp.

15.1.3.48 cx_mat & Body::refDataSolution ( )

The same things as the refSolution() function, just under a different name.

Returns

Reference to column matrix of complex numbers. Value returned by reference. Matrix size is not hard coded. Number of rows in matrix must match number of equations for body property.

See Also

body::refSolution()

Definition at line 714 of file body.cpp.

15.1.3.49 std::complex< double > & Body::refDataSolution (int *varIndexIn*)

Returns a single solution value, based on the variable requested.

Variable is requested by the data index, not vector occurrence index.

Parameters

varIndexIn	Integer. The variable's data index

Returns

Returns a complex<double> variable. This is the value of the solution object for the variable requested Returned variable passed by reference.

Definition at line 720 of file body.cpp.

15.1.3.50 double & Body::refHeading ( )

Exposes the heading property for operations.

Returns

Pointer to the heading property.

Definition at line 199 of file body.cpp.

15.1.3.51 string & Body::refHydroBodName ( )

Exposes the hydro body name property for operation.

Returns

Pointer to the hydro body name property.

Definition at line 169 of file body.cpp.

15.1.3.52 Mat< double > & Body::refPosn ( )

Exposes the position property for operation. The entire matrix for position of the body.

Returns the entire matrix for the position of the body. Output is a 3x1 matrix with double point precision. First entry (1,1) = Position in X-axis. Second entry (2,1) = Position in Y-axis. Third entry (3,1) = Position in Z-axis. Units are in meters. Position is relative to the orientation of the world coordinate system.

Returns

Returns a pointer to the the entire matrix for the position of the body. Output is a 3x1 matrix with double point precision. First entry (1,1) = Position in X-axis. Second entry (2,1) = Position in Y-axis. Third entry (3,1) = Position in Z-axis. Units are in meters. Position is relative to the orientation of the world coordinate system.

Definition at line 680 of file body.cpp.

15.1.3.53 cx_mat & Body::refSolution ( )

Get the solution matrix for the body.

Gets the solution matrix for the body. Used to store the solution from the motion solver. This variable is initially empty on body creation. It gets filled with the output from the motion solver. Output is a column matrix (n by 1) of complex numbers. Output is in units of meters.

Returns

Reference to column matrix of complex numbers. Value returned by reference. Matrix size is not hard coded. Number of rows in matrix must match number of equations for body property.

Definition at line 708 of file body.cpp.

15.1.3.54 void Body::setBodyName (std::string)

Sets the bodyName.

Parameters

newName	The std::string passed in sets bodyName.

Definition at line 157 of file body.cpp.

15.1.3.55 void Body::setCenX (double *newCenX*)

Sets the Centroid X.

Parameters

newCenX	The double passed in sets centroidX.

Definition at line 578 of file body.cpp.

15.1.3.56 void Body::setCenY (double *newCenY*)

Sets the Centroid Y.

Parameters

newCenY	The double passed in sets centroidY.

Definition at line 593 of file body.cpp.

15.1.3.57 void Body::setCenZ (double *newCenZ*)

Sets the Centroid Z.

Parameters

newCenZ	The double passed in sets centroidZ.

Definition at line 608 of file body.cpp.

15.1.3.58 void Body::setHeading (double *newHeading*)

Sets the heading.

Parameters

newHeading	The double passed in sets the heading.

Definition at line 187 of file body.cpp.

15.1.3.59 void Body::setHydroBodName (std::string)

Sets the hydroBody.

Parameters

newName	The std::string passed in sets the hydroBody.

Definition at line 175 of file body.cpp.

15.1.3.60 void Body::setMass (double *newMass*)

Sets the mass.

Parameters

newMass	The double passed in sets the mass.

Definition at line 208 of file body.cpp.

15.1.3.61 void Body::setMassMatrix (arma::Mat< double > *MassMatIn*)

Set the mass matrix for the body.

Parameters

MassMatIn	The input mass matrix for the body. A 6x6 matrix.

Definition at line 566 of file body.cpp.

15.1.3.62 void Body::setMomIxx (double *newXX*)

Sets the Moment of Inertia XX (Ixx)

Parameters

newXX	The double passed in sets momentOfInertiaXX.

Definition at line 291 of file body.cpp.

15.1.3.63 void Body::setMomIxy (double *newXY*)

Sets the Product of Inertia XY (Ixy)

Parameters

newXY	The double passed in sets setCrossMomentXY.

Definition at line 416 of file body.cpp.

15.1.3.64 void Body::setMomIxz (double *newXZ*)

Sets the Product of Inertia XZ (Ixz)

Parameters

newXZ	The double passed in sets setCrossMomentXZ.

Definition at line 460 of file body.cpp.

15.1.3.65 void Body::setMomIyy (double *newYY*)

Sets the Moment of Inertia YY (Iyy)

Parameters

newYY	The double passed in sets momentOfInertiaYY.

Definition at line 332 of file body.cpp.

15.1.3.66 void Body::setMomIyz (double *newYZ*)

Sets the Product of Inertia YZ (Iyz)

Parameters

newYZ	The double passed in sets setCrossMomentYZ.

Definition at line 504 of file body.cpp.

15.1.3.67 void Body::setMomIzz (double *newZZ*)

Sets the Moment of Inertia ZZ (Izz)

Parameters

newZZ	The double passed in sets momentOfInertiaZZ.

Definition at line 374 of file body.cpp.

15.1.3.68 void Body::setMotionModel (ofreq::MotionModel & *modelIn*)

Sets the motion model for lookup later.

Parameters

modelIn	Variable input that is the motion model object. Variable passed by reference. Stored internally as a pointer.

Definition at line 987 of file body.cpp.

15.1.3.69 void Body::setPosnX (double *input*)

Sets the body position in the X-axis.

Sets the body position in the X-axis. Position is set relative to the world coordinate system. Units are in meters.

Parameters

input	Double input specifying the position on the X-axis, units of meters.

Definition at line 629 of file body.cpp.

15.1.3.70 void Body::setPosnY (double *input*)

Sets the body position in the Y-axis.

Sets the body position in the Y-axis. Position is set relative to the world coordinate system. Units are in meters.

Parameters

input	Double input specifying the position on the Y-axis, units of meters.

Definition at line 644 of file body.cpp.

15.1.3.71 void Body::setPosnZ (double *input*)

Sets the body position in the Z-axis.

Sets the body position in the Z-axis. Position is set relative to the world coordinate system. Units are in meters.

Parameters

input	Double input specifying the position on the Z-axis, units of meters.

Definition at line 659 of file body.cpp.

15.1.3.72 void Body::setSolnMat (arma::cx_mat *input*)

Set the solution matrix for the body.

Sets the solution matrix for the body. Used to store the solution from the motion solver. This variable is initially empty on body creation. It gets filled with the output from the motion solver. Output is a column matrix (n by 1) of complex numbers. Output is in units of meters.

Parameters

input	Column matrix of complex numbers. Matrix size is not hard coded. Number of rows in matrix must match number of equations for body property.

Definition at line 696 of file body.cpp.

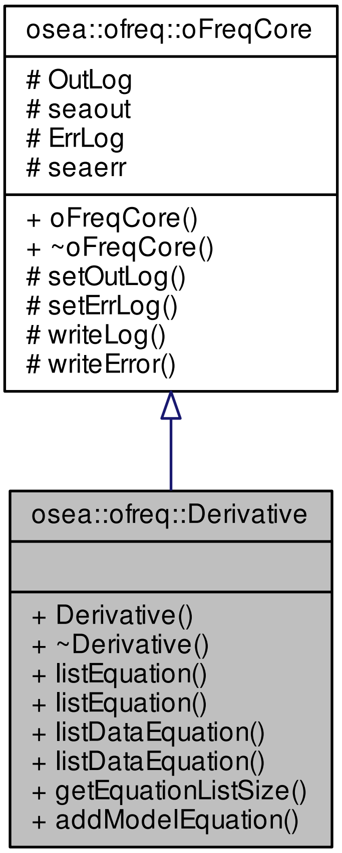
The documentation for this class was generated from the following files:

- bin/ofreq/global_objects/body.h
- bin/ofreq/global_objects/body.cpp

15.2 osea::ofreq::Derivative Class Reference

```
#include <derivative.h>
```

Inheritance diagram for osea::ofreq::Derivative:



Public Member Functions

- Derivative ()
- ∼Derivative ()
- std::vector< Equation > & listEquation ()

  The list of equations.
- Equation & listEquation (unsigned int number)

  Retrieve the equation specified by the number.
- Equation & listDataEquation (int indexIn)

  Returns the equation requested. Only specified by the data index property of the equation object.
- std::vector< Equation > & listDataEquation ()

  Returns the list of equation objects.
- int getEquationListSize ()
- void addModelEquation (std::vector< double > listCoeffsIn, int EqnDataIn=-1)

  Creates a new equation object and adds it to the list of equation objects contained in this derivative.

Additional Inherited Members

15.2.1 Detailed Description

This class holds data for a derivative.

Definition at line 84 of file derivative.h.

15.2.2 Constructor & Destructor Documentation

15.2.2.1 Derivative::Derivative ( )

This default constructor creates a Body object.

Definition at line 36 of file derivative.cpp.

15.2.2.2 Derivative::∼Derivative ( )

The default destructor, nothing happens here.

Definition at line 41 of file derivative.cpp.

15.2.3 Member Function Documentation

15.2.3.1 void Derivative::addModelEquation (std::vector< double > *listCoeffsIn*, int *EqnDataIn* = −1)

Creates a new equation object and adds it to the list of equation objects contained in this derivative.

New equation object is created automatically within this function. Function merely takes the list of input coefficients and creates all equation objects necessary from that.

Parameters

EqnDataIn	Integer. The data index of the equation object. If no input is provided, the function assumes the data index to be the index of the equation's current place in the vector.
listCoeffsIn	Vector of doubles. The list of coefficients. Each coefficient corresponds to a single variable. List of coefficients is organized by *data index*. The coefficient's position in the list is it's data index.

Definition at line 103 of file derivative.cpp.

15.2.3.2 int Derivative::getEquationListSize ( )

Retrieve the size of the equation list.

Returns

The size of the equation list.

Definition at line 97 of file derivative.cpp.

15.2.3.3 Equation & Derivative::listDataEquation (int *indexIn*)

Returns the equation requested. Only specified by the data index property of the equation object.

Returns the equation requested. Only specified by the data index property of the equation object.

Parameters

indexIn	The integer describing the data index for the equation requested.

Returns

Equation object specified by the DataIndex of indexIn. Value returned is by value.

Definition at line 67 of file derivative.cpp.

15.2.3.4 std::vector< Equation > & Derivative::listDataEquation ( )

Returns the list of equation objects.

This is the same as the listEquation() function, just under a different name.

Returns

Returns a vector of Equation objects. Returned value passed by reference.

Definition at line 90 of file derivative.cpp.

15.2.3.5 vector< Equation > & Derivative::listEquation ( )

The list of equations.

Returns

Returns a vector of Equation objects. Returned value passed by reference.

Definition at line 47 of file derivative.cpp.

15.2.3.6 Equation & Derivative::listEquation (unsigned int *number*)

Retrieve the equation specified by the number.

Retrieves the equation specified by the number. Value returned is a reference to the equation object. Allows editting of the equation object, or just data access.

Parameters

number	Integer representing which equation number should be returned.

Returns

Value returned is a reference to the equation object. Allows editting of the equation object, or just data access.

Definition at line 54 of file derivative.cpp.

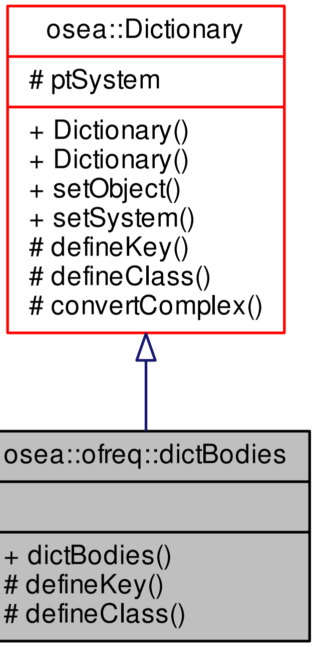
The documentation for this class was generated from the following files:

- bin/ofreq/global_objects/derivative.h
- bin/ofreq/global_objects/derivative.cpp

15.3 osea::ofreq::dictBodies Class Reference

```
#include <dictbodies.h>
```

Inheritance diagram for osea::ofreq::dictBodies:



Public Member Functions

- dictBodies ()

Protected Member Functions

- int defineKey (std::string keyIn, std::vector< std::string > valIn)

  Function that defines how to interpret the key values. Contains a list of key names and corresponding actions to take for interpretting each key value.
- int defineClass (std::string nameIn)

  Function that defines how to interpret the class name. Class name implies declaration of a new object of the class named by the class name. This is a separate set of definitions to handle class declarations.

Additional Inherited Members

15.3.1 Detailed Description

The dictBodies class defines the key-word value pairs associated with the Bodies.in input file. Just as a normal dictionary defines the meaning of words, the dictBodies class works in the same way. The dictBodies class takes individual pairs of keywords and values. It has a definition for each of these keywords. The definition is whatever actions are necessary to process the value of key-pair and apply it to the program. This may include variable type conversions. It will also use slots and signals to retrieve pointers to any appropriate objects that the dictBodies object needs to interact with. It will use the properties of those objects to apply the values it finds in the key-value pair. Any objects created in the dictBodies class can be safely deleted once all file reading is done.

Note: The code for the dictBodies object always references the last object in the list. This assumes that no other commands get issued in the input file between the creation of an object and the definition of key-value pairs associated with that object. Currently, I can not imagine any situation where this assumption would be violated. But do consider this when planning error recovery methods.

See Also

Dictionary
FileReader

Definition at line 96 of file dictbodies.h.

15.3.2 Constructor & Destructor Documentation

15.3.2.1 dictBodies::dictBodies ( )

Definition at line 67 of file dictbodies.cpp.

15.3.3 Member Function Documentation

15.3.3.1 int dictBodies::defineClass (std::string *nameIn*) [protected], [virtual]

Function that defines how to interpret the class name. Class name implies declaration of a new object of the class named by the class name. This is a separate set of definitions to handle class declarations.

Parameters

nameIn	std::string, variable passed by value. The name of the class name.

Returns

Returns status of assigning key. Returned value is an integer, passed by value. See list of return codes below: 0: Key definition found. Success. 1: No key found. / General error message. 2: Key is invalid within current active object. 99: Function virtual definition only. Not currently defined.

Reimplemented from osea::Dictionary.

Definition at line 328 of file dictbodies.cpp.

15.3.3.2 int dictBodies::defineKey (std::string *keyIn*, std::vector< std::string > *valIn*) [protected], [virtual]

Function that defines how to interpret the key values. Contains a list of key names and corresponding actions to take for interpretting each key value.

Parameters

keyIn	std::string containing the key name. Variable passed by value.
valIn	Vector of strings containing the key values. Variable passed by value.

Returns

Returns status of assigning key. Returned value is an integer, passed by value. See list of return codes below: 0: Key definition found. Success. 1: No key found. / General error message. 2: Key is invalid within current active object. 99: Function virtual definition only. Not currently defined.

Reimplemented from osea::Dictionary.

Definition at line 83 of file dictbodies.cpp.

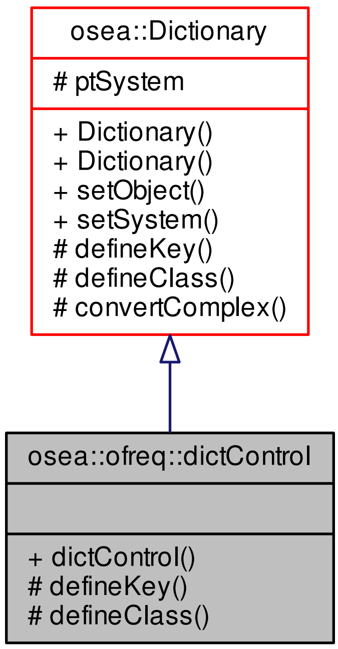
The documentation for this class was generated from the following files:

- bin/ofreq/file_reader/dictbodies.h
- bin/ofreq/file_reader/dictbodies.cpp

15.4 osea::ofreq::dictControl Class Reference

```
#include <dictcontrol.h>
```

Inheritance diagram for osea::ofreq::dictControl:



Public Member Functions

- dictControl ()

Protected Member Functions

- int defineKey (std::string keyIn, std::vector< std::string > valIn)

  Function that defines how to interpret the key values. Contains a list of key names and corresponding actions to take for interpretting each key value.
- int defineClass (std::string nameIn)

  Function that defines how to interpret the class name. Class name implies declaration of a new object of the class named by the class name. This is a separate set of definitions to handle class declarations.

Additional Inherited Members

15.4.1 Detailed Description

The dictControl class defines the key-word value pairs associated with the Control.in input file. Just as a normal dictionary defines the meaning of words, the dictControl class works in the same way. The dictControl class takes individual pairs of keywords and values. It has a definition for each of these keywords. The definition is whatever actions are necessary to process the value of key-pair and apply it to the program. This may include variable type conversions. It will also use slots and signals to retrieve pointers to any appropriate objects that the dictControl object needs to interact with. It will use the properties of those objects to apply the values it finds in the key-value pair. Any objects created in the dictControl object can be safely deleted once all file reading is done.

See Also

Dictionary
FileReader

Definition at line 90 of file dictcontrol.h.

15.4.2 Constructor & Destructor Documentation

15.4.2.1 dictControl::dictControl ( )

Definition at line 50 of file dictcontrol.cpp.

15.4.3 Member Function Documentation

15.4.3.1 int dictControl::defineClass (std::string *nameIn*) [protected], [virtual]

Function that defines how to interpret the class name. Class name implies declaration of a new object of the class named by the class name. This is a separate set of definitions to handle class declarations.

Parameters

nameIn	std::string, variable passed by value. The name of the class name.

Returns

Returns status of assigning key. Returned value is an integer, passed by value. See list of return codes below: 0: Key definition found. Success. 1: No key found. / General error message. 2: Key is invalid within current active object. 99: Function virtual definition only. Not currently defined.

Reimplemented from osea::Dictionary.

Definition at line 132 of file dictcontrol.cpp.

15.4.3.2 int dictControl::defineKey (std::string *keyIn*, std::vector< std::string > *valIn*) [protected], [virtual]

Function that defines how to interpret the key values. Contains a list of key names and corresponding actions to take for interpretting each key value.

Parameters

keyIn	std::string containing the key name. Variable passed by value.
valIn	Vector of strings containing the key values. Variable passed by value.

Returns

Returns status of assigning key. Returned value is an integer, passed by value. See list of return codes below: 0: Key definition found. Success. 1: No key found. / General error message. 2: Key is invalid within current active object. 99: Function virtual definition only. Not currently defined.

Reimplemented from osea::Dictionary.

Definition at line 66 of file dictcontrol.cpp.

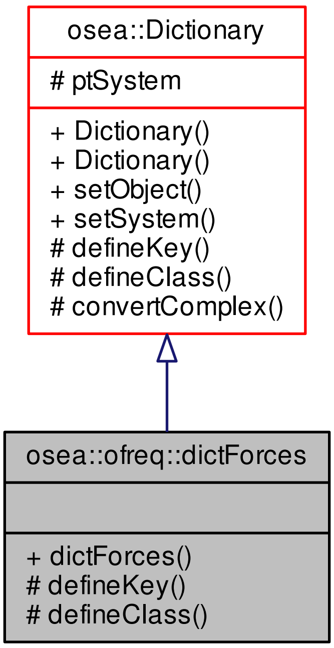
The documentation for this class was generated from the following files:

- bin/ofreq/file_reader/dictcontrol.h
- bin/ofreq/file_reader/dictcontrol.cpp

15.5 osea::ofreq::dictForces Class Reference

```
#include <dictforces.h>
```

Inheritance diagram for osea::ofreq::dictForces:



Public Member Functions

- dictForces ()

Protected Member Functions

- int defineKey (std::string keyIn, std::vector< std::string > valIn)

Function that defines how to interpret the key values. Contains a list of key names and corresponding actions to take for interpretting each key value.

- int defineClass (std::string nameIn)

  Function that defines how to interpret the class name. Class name implies declaration of a new object of the class named by the class name. This is a separate set of definitions to handle class declarations.

Additional Inherited Members

15.5.1 Detailed Description

The dictForces class defines the key-word value pairs associated with the Forces.in input file. Just as a normal dictionary defines the meaning of words, the dictForces class works in the same way. The dictForces class takes individual pairs of keywords and values. It has a definition for each of these keywords. The definition is whatever actions are necessary to process the value of key-pair and apply it to the program. This may include variable type conversions. It will also use slots and signals to retrieve pointers to any appropriate objects that the dictForces object needs to interact with. It will use the properties of those objects to apply the values it finds in the key-value pair. Any objects created in the dictForces class can be safely deleted once all file reading is done.

Note: The code for the dictForces object always references the last object in the list. This assumes that no other commands get issued in the input file between the creation of an object and the definition of key-value pairs associated with that object. Currently, I can not imagine any situation where this assumption would be violated. But do consider this when planning error recovery methods.

See Also

Dictionary
FileReader

Definition at line 97 of file dictforces.h.

15.5.2 Constructor & Destructor Documentation

15.5.2.1 dictForces::dictForces ( )

Definition at line 57 of file dictforces.cpp.

15.5.3 Member Function Documentation

15.5.3.1 int dictForces::defineClass (std::string *nameIn*) [protected], [virtual]

Function that defines how to interpret the class name. Class name implies declaration of a new object of the class named by the class name. This is a separate set of definitions to handle class declarations.

Parameters

nameIn	std::string, variable passed by value. The name of the class name.

Returns

Returns status of assigning key. Returned value is an integer, passed by value. See list of return codes below: 0: Key definition found. Success. 1: No key found. / General error message. 2: Key is invalid within current active object. 99: Function virtual definition only. Not currently defined.

Reimplemented from osea::Dictionary.

Definition at line 203 of file dictforces.cpp.

15.5.3.2 int dictForces::defineKey (std::string *keyIn*, std::vector< std::string > *valIn*) [protected], [virtual]

Function that defines how to interpret the key values. Contains a list of key names and corresponding actions to take for interpretting each key value.

Parameters

keyIn	std::string containing the key name. Variable passed by value.
valIn	Vector of strings containing the key values. Variable passed by value.

Returns

Returns status of assigning key. Returned value is an integer, passed by value. See list of return codes below: 0: Key definition found. Success. 1: No key found. / General error message. 2: Key is invalid within current active object. 99: Function virtual definition only. Not currently defined.

Reimplemented from osea::Dictionary.

Definition at line 71 of file dictforces.cpp.

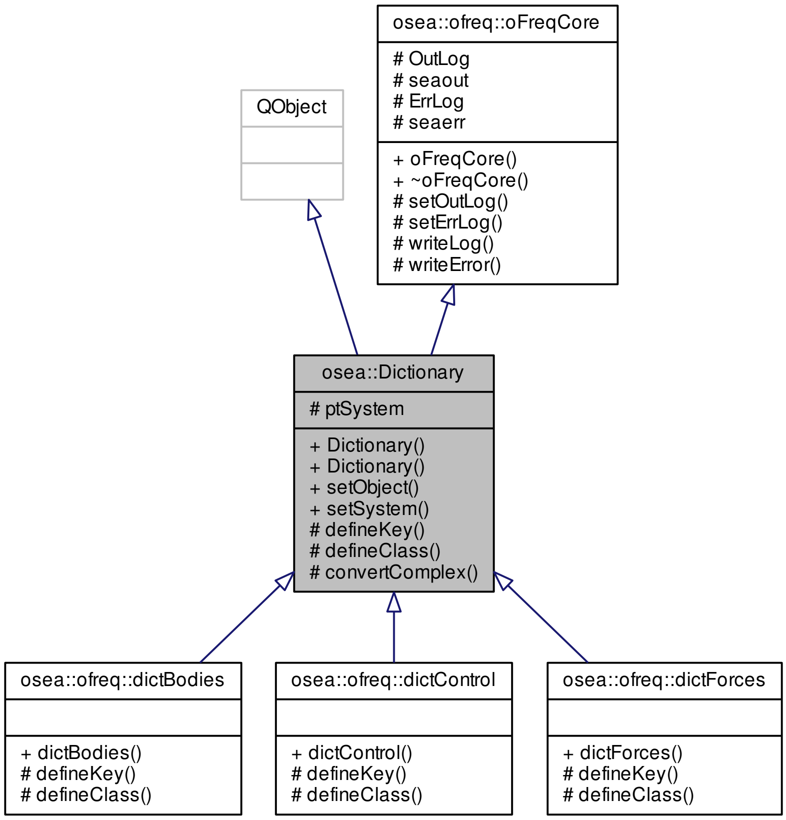
The documentation for this class was generated from the following files:

- bin/ofreq/file_reader/dictforces.h
- bin/ofreq/file_reader/dictforces.cpp

15.6 osea::Dictionary Class Reference

`#include <dictionary.h>`

Inheritance diagram for osea::Dictionary:



Public Slots

- virtual void setObject (ObjectGroup input)

  Public signal for the ObjectGroup object that is sent to the Dictionary object for procesing.
- virtual void setSystem (ofreq::System ∗ptInput)

  Sets the system object for the dictionary to reference.

Public Member Functions

- Dictionary (QObject ∗parent)
- Dictionary ()

Protected Member Functions

- virtual int defineKey (std::string keyIn, std::vector< std::string > valIn)

  Pure virtual function that defines how to interpret the key values. Contains a list of key names and corresponding actions to take for interpretting each key value.
- virtual int defineClass (std::string nameIn)

  Pure virtual function that defines how to interpret the class name. Class name implies declaration of a new object of the class named by the class name. This is a separate set of definitions to handle class declarations.
- std::complex< double > convertComplex (std::string input)

Converts a std::string of a complex number into a complex object (double base type) i.e. std::complex<double>.

Protected Attributes

- ofreq::System * ptSystem

  Pointer to the System object. Used to reference any important variables in the System object.

Additional Inherited Members

15.6.1 Detailed Description

This is a virtual class definition, inheritted by each fileDictionary object. Contains the basic functions for how to recursively progress through the definitions for an ObjectGroup object that is fed in.

See Also

ObjectGroup

Definition at line 84 of file dictionary.h.

15.6.2 Constructor & Destructor Documentation

15.6.2.1 Dictionary::Dictionary (QObject * *parent*) `[explicit]`

Definition at line 36 of file dictionary.cpp.

15.6.2.2 Dictionary::Dictionary ( )

Definition at line 42 of file dictionary.cpp.

15.6.3 Member Function Documentation

15.6.3.1 complex< double > Dictionary::convertComplex (std::string *input*) `[protected]`

Converts a std::string of a complex number into a complex object (double base type) i.e. std::complex<double>.

Parameters

input	The std::string which holds the complex number. Valid input formats are: 1.00+1.00i 1.00-1.00i 1.00+i1.00 1.00-i1.00 1.414<0.785398 (angle must be in radians) 1.414<-0.785398 (angle must be in radians)

Returns

Returns a std::complex<double> object. Variable passed by value.

Definition at line 105 of file dictionary.cpp.

15.6.3.2 int Dictionary::defineClass (std::string *nameIn*) `[protected]`, `[virtual]`

Pure virtual function that defines how to interpret the class name. Class name implies declaration of a new object of the class named by the class name. This is a separate set of definitions to handle class declarations.

Parameters

nameIn	std::string, variable passed by value. The name of the class name.

Returns

Returns status of assigning key. Returned value is an integer, passed by value. See list of return codes below: 0: Key definition found. Success. 1: No key found. / General error message. 2: Key is invalid within current active object. 99: Function virtual definition only. Not currently defined.

Reimplemented in osea::ofreq::dictBodies, osea::ofreq::dictForces, and osea::ofreq::dictControl.

Definition at line 97 of file dictionary.cpp.

15.6.3.3 int Dictionary::defineKey (std::string *keyIn*, std::vector< std::string > *valIn*) [protected], [virtual]

Pure virtual function that defines how to interpret the key values. Contains a list of key names and corresponding actions to take for interpretting each key value.

Parameters

keyIn	std::string containing the key name. Variable passed by value.
valIn	Vector of strings containing the key values. Variable passed by value.

Returns

Returns status of assigning key. Returned value is an integer, passed by value. See list of return codes below: 0: Key definition found. Success. 1: No key found. / General error message. 2: Key is invalid within current active object. 99: Function virtual definition only. Not currently defined.

Reimplemented in osea::ofreq::dictBodies, osea::ofreq::dictForces, and osea::ofreq::dictControl.

Definition at line 89 of file dictionary.cpp.

15.6.3.4 void Dictionary::setObject (ObjectGroup *input*) [virtual], [slot]

Public signal for the ObjectGroup object that is sent to the Dictionary object for procesing.

Parameters

input	The ObjectGroup object that contains the class definitions. Variable passed by value.

Definition at line 54 of file dictionary.cpp.

15.6.3.5 void Dictionary::setSystem (ofreq::System * *ptInput*) [virtual], [slot]

Sets the system object for the dictionary to reference.

Parameters

ptSystem	Pointer to the System object. Variable passed by value.

Definition at line 80 of file dictionary.cpp.

15.6.4 Member Data Documentation

15.6.4.1 ofreq::System* osea::Dictionary::ptSystem [protected]

Pointer to the System object. Used to reference any important variables in the System object.

Definition at line 166 of file dictionary.h.

The documentation for this class was generated from the following files:

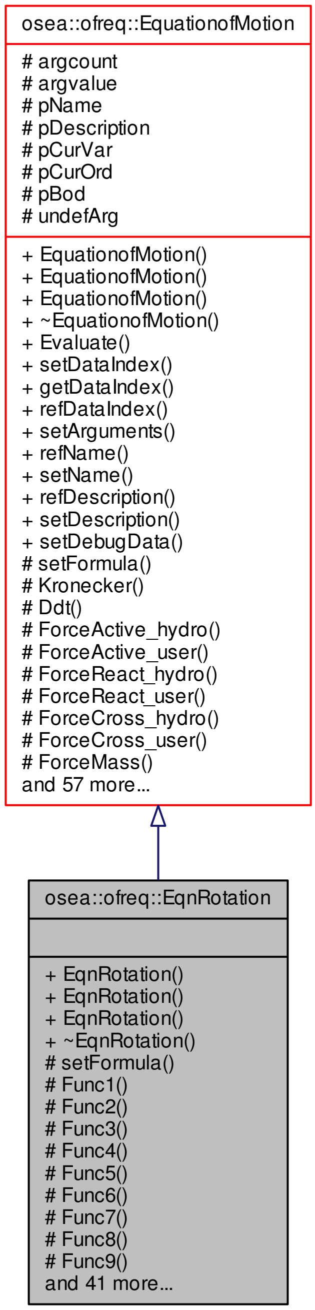
- bin/ofreq/file_reader/dictionary.h
- bin/ofreq/file_reader/dictionary.cpp

15.7 osea::ofreq::EqnRotation Class Reference

The EqnRotation class.

`#include <eqnrotation.h>`

Inheritance diagram for osea::ofreq::EqnRotation:



Public Member Functions

- EqnRotation (MotionModel *modelIn)

    Default contrustor. Contains a reference to the motion model class which constructs it.

- EqnRotation (MotionModel *modelIn, std::string NameIn)

    Contrustor with name. Contains a reference to the motion model class which constructs it.

- EqnRotation (MotionModel *modelIn, std::string NameIn, int IndexIn)

Contrustor with name and index. Contains a reference to the motion model class which constructs it.
- ∼EqnRotation ()

  Default destructor.

Protected Member Functions

- std::complex< double > setFormula ()

  The formula used by the equation of motion.
- std::complex< double > Func1 ()

  Func1 through Func50 provide user custom defined functions.
- std::complex< double > Func2 ()
- std::complex< double > Func3 ()
- std::complex< double > Func4 ()
- std::complex< double > Func5 ()
- std::complex< double > Func6 ()
- std::complex< double > Func7 ()
- std::complex< double > Func8 ()
- std::complex< double > Func9 ()
- std::complex< double > Func10 ()
- std::complex< double > Func11 ()
- std::complex< double > Func12 ()
- std::complex< double > Func13 ()
- std::complex< double > Func14 ()
- std::complex< double > Func15 ()
- std::complex< double > Func16 ()
- std::complex< double > Func17 ()
- std::complex< double > Func18 ()
- std::complex< double > Func19 ()
- std::complex< double > Func20 ()
- std::complex< double > Func21 ()
- std::complex< double > Func22 ()
- std::complex< double > Func23 ()
- std::complex< double > Func24 ()
- std::complex< double > Func25 ()
- std::complex< double > Func26 ()
- std::complex< double > Func27 ()
- std::complex< double > Func28 ()
- std::complex< double > Func29 ()
- std::complex< double > Func30 ()
- std::complex< double > Func31 ()
- std::complex< double > Func32 ()
- std::complex< double > Func33 ()
- std::complex< double > Func34 ()
- std::complex< double > Func35 ()
- std::complex< double > Func36 ()
- std::complex< double > Func37 ()
- std::complex< double > Func38 ()
- std::complex< double > Func39 ()
- std::complex< double > Func40 ()
- std::complex< double > Func41 ()
- std::complex< double > Func42 ()
- std::complex< double > Func43 ()
- std::complex< double > Func44 ()
- std::complex< double > Func45 ()

- std::complex< double > Func46 ()
- std::complex< double > Func47 ()
- std::complex< double > Func48 ()
- std::complex< double > Func49 ()
- std::complex< double > Func50 ()

Additional Inherited Members

15.7.1 Detailed Description

The EqnRotation class.

This class defines motion for rotation in one of the three principle axes. (X-axis, Y-axis, Z-axis). This is specific to rigid body motion and intended for applicaiton in a six-degree of freedom (6DOF) motion model.

This class is a derived equation of motion. It inherits all the programming needed to define an equation of motion from the class EquationofMotion.

Custom equation classes serve a specific purpose. All the programming necessary to define an equation of motion within the program ofreq is already contained within the class EquationofMotion. All that programming gets inherited from the class. The only thing the custom class needs to define is the specific equation for this purpose. That equation is contained inside the function setFormula(). Everything else is already defined, as far as programming.

There are specific requirements for how to define the custom EquationofMotion class. These instructions assume the user has some basic experience with C++ programming. If not, first try a few tutorials. There are numerous tutorials on the internet. You should specifically familiarize yourself with concepts such as namespaces, classes, objects, and functions. Every effort was made to isolate the user from the complexity of the ofreq program, but some basic understand of program operation is needed.

Assuming you have a basic knowledge of C++ programming, you can create your custom equation of motion by editing the .cpp file associated with this class. Follow these basic rules:

1.) Always create a backup of every file before you edit it.

2.) Only edit the .cpp file. Leave the header file alone (the .h file). The header file creates the linking necessary to make everything work correctly.

3.) Only define your equation within the function setFormula().

4.) Do not add function definitions to the header file unless you are familiar with C++ class programming and know how to safely add the function into the class definition. If you need to define custom functions for your equstions, the safest thing is to define them strictly within your source code file (.cpp file). Just remember that all functions must be fully defined before they get used within the code.

5.) There are several functions inheritted from the EquationofMotion class. You can use these to refer to the different forces when developing your own equation of motion. oFreq recognizes seven (7) basic force types shown below, with the function name to reference them in the EquationofMotion. (Arguments for each function are not shown, for sake of clarity. 5.1) ForceMass(...) = The forces associated with the mass of an object. This includes direct mass for straight linear motion, and moment of inertia for rotational motion. 5.2) ForceActive_hydro(...) = The forces which are independant of body motions. The hydro subcategory refers to active forces that specifically come from hydrodynamic forces. This includes the forces from incident waves. Sometimes call the Froude-Krylov forces. 5.3) ForceActive_user(...) = The forces which are independant of body motions. The user subcategory refers to active forces specifically defined by the user in the ofreq run file. These may be some external force such as an active control system. Regardless, it is customed defined by the user. 5.4) ForceReact_hydro(...) = The forces which are reactive and dependant on body motions. This includes derivatives of body motions. The hydro subcategory refers to reactive forces hydrodynamic in origin. This would include body hydrostatic properties, added damping, and added mass. 5.5) ForceReact_user(...) = The forces which are reactive and dependant on body motions. This includes derivatives of body motions. The user subcategory refers to reactive forces defined by the user. This might include external forces such as a mooring line or dynamic positioning system. In any case, these are reactive forces defined at run time in the ofreq input files. 5.6) ForceCross_hydro(...) = The forces which are reactive and dependant on the body motions of another body. This is only applicable to multi-body systems. Examples might be two vessels near each other. The program can accept equations that use the cross-body forces but are only applied to a single body problem. The hydro subcategory refers to reactive forces hydrodynamic in origin. This would include body hydrostatic properties, added damping, and added mass, except that these forces would be dependant on the motions of another body. 5.7) ForceCross_user(...) = The forces which are reactive and dependant on the body motions of another body. This is only applicable to multi-body systems. Examples might be two vessels near each other. The program can accept equations that use the cross-body forces but are only applied to a single body problem. The user subcategory refers to reactive forces defined by the user. This might include external forces such as a mooring line or dynamic positioning system. In any case, these are reactive forces defined at run time in the ofreq input files.

6.) Use of the Sum() Function. There are three possible implementations of the Sum() function. The input syntax determines which function to use. 6.1) Sum a finite value: This implementation occurrs when a variable is provided as the argument for for the summation. The variable must be of data type complex<double>. The variable will not change during the summation. Variable is passed by value. 6.2) Sum a function contained within the class: This is the most common implementation of the Sum() function. The class has 50 functions provided for your use. They are named Func1 through Func50. You may enter any code within these functions. But the functions do not accept any inputs. This is a limitation of program. The functions will update with each iteration of the Sum() function. Anyhthing that you wish to change during summation must be captured within one of the custom functions. This also includes references to any other class functions. To implement the custom function, you simply type in the function name as a string input. Example: Sum("Func1()", "body", 0, 1)

And then the function definition for Func1 would be: Func1() { return ForceReact_hydro(ord(), var()) * Ddt(var(), ord()); }

This was just one example. Any combination may be used within the custom function. 6.3) Sum a function not contained within the class. This is mostly used for debugging when you wish to test a custom equation of motion, isolated from the main program. The returned data type from the function must be complex<double>. To use your external function within the Sum() function, you must enter as a function pointer. The Sum function expects a pointer to a function. You would enter it as follows (all capitals are the terms you change for your specific function):

output = Sum(&FUNCTION_NAME, index, from, to);

Two key points to notice: The function name was preceded with a reference symbol (&); and I only stated the function name. I did not include the brackets to explicitely state that it's a function. Don't include the brackets. You will get a compiler error if you do.

See Also

EquationofMotion
MotionModel

Definition at line 185 of file eqnrotation.h.

15.7.2 Constructor & Destructor Documentation

15.7.2.1 EqnRotation::EqnRotation (MotionModel * *modelIn*)

Default contrustor. Contains a reference to the motion model class which constructs it.

Default contrustor. Contains a reference to the motion model class which constructs it. The constructing class is necessary because several functions in the EquationOfMotion class use data in the constructing class, the motion model class.

Parameters

modelIn	A pointer to the motion model object that created the equation of motion.

Definition at line 37 of file eqnrotation.cpp.

15.7.2.2 EqnRotation::EqnRotation (MotionModel ∗ *modelIn*, std::string *NameIn*)

Contrustor with name. Contains a reference to the motion model class which constructs it.

Default contrustor. Contains a reference to the motion model class which constructs it. The constructing class is necessary because several functions in the EquationOfMotion class use data in the constructing class, the motion model class.

Parameters

modelIn	A pointer to the motion model object that created the equation of motion.
NameIn	A name for what physical property the equation solves for. Used for user output. Not critical to program execution.

Definition at line 44 of file eqnrotation.cpp.

15.7.2.3 EqnRotation::EqnRotation (MotionModel ∗ *modelIn*, std::string *NameIn*, int *IndexIn*)

Contrustor with name and index. Contains a reference to the motion model class which constructs it.

Default contrustor. Contains a reference to the motion model class which constructs it. The constructing class is necessary because several functions in the EquationOfMotion class use data in the constructing class, the motion model class.

Parameters

modelIn	A pointer to the motion model object that created the equation of motion.
NameIn	A name for what physical property the equation solves for. Used for user output. Not critical to program execution.
IndexIn	Sets the index for the Equation of Motion. The index is how the equation determines which numbers to access on the data. The following indices are used. Any higher indices can extend beyond this range, and the program easily adapts. But the following three are reserved. Unused indices are not transferred to the matrices when solved. So unused indices to not negatively impact calculation performance. However, using excessively large indices (say 500 when you only have 3 equations) will result in large matrices and unecessary memory requirements. THe following index reservations apply. 1: Translation in x-direction. Specific to rigid body motion. 2: Translation in y-direction. Specific to rigid body motion. 3: Translation in z-direction. Specific to rigid body motion. 4: Rotation about x-direction. Specific to rigid body motion. 5: Rotation about y-direction. Specific to rigid body motion. 6: Rotation about z-direction. Specific to rigid body motion.

Definition at line 51 of file eqnrotation.cpp.

15.7.2.4 EqnRotation::∼EqnRotation ( )

Default destructor.

Definition at line 58 of file eqnrotation.cpp.

15.7.3 Member Function Documentation

15.7.3.1 std::complex< double > EqnRotation::Func1 ( ) [protected], [virtual]

Func1 through Func50 provide user custom defined functions.

These are custom functions that the user may need to create to define their equations of motion. The only restriction is that the functions can not take any arguments. Any arguments required must be supplied through a set of global variables. Sorry, that's just a restriction of how the code is written and the use of the C++ language.

Returns

Returns a complex<double> variable. Returned variabled passed by value.

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 111 of file eqnrotation.cpp.

15.7.3.2 std::complex< double > EqnRotation::Func10 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 176 of file eqnrotation.cpp.

15.7.3.3 std::complex< double > EqnRotation::Func11 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 183 of file eqnrotation.cpp.

15.7.3.4 std::complex< double > EqnRotation::Func12 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 190 of file eqnrotation.cpp.

15.7.3.5 std::complex< double > EqnRotation::Func13 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 196 of file eqnrotation.cpp.

15.7.3.6 std::complex< double > EqnRotation::Func14 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 202 of file eqnrotation.cpp.

15.7.3.7 std::complex< double > EqnRotation::Func15 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 208 of file eqnrotation.cpp.

15.7.3.8 std::complex< double > EqnRotation::Func16 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 214 of file eqnrotation.cpp.

15.7.3.9 std::complex< double > EqnRotation::Func17 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 220 of file eqnrotation.cpp.

15.7.3.10 std::complex< double > EqnRotation::Func18 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 226 of file eqnrotation.cpp.

15.7.3.11 std::complex< double > EqnRotation::Func19 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 232 of file eqnrotation.cpp.

15.7.3.12 std::complex< double > EqnRotation::Func2 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 118 of file eqnrotation.cpp.

15.7.3.13 std::complex< double > EqnRotation::Func20 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 238 of file eqnrotation.cpp.

15.7.3.14 std::complex< double > EqnRotation::Func21 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 244 of file eqnrotation.cpp.

15.7.3.15 std::complex< double > EqnRotation::Func22 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 250 of file eqnrotation.cpp.

15.7.3.16 std::complex< double > EqnRotation::Func23 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 256 of file eqnrotation.cpp.

15.7.3.17 std::complex< double > EqnRotation::Func24 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 262 of file eqnrotation.cpp.

15.7.3.18 std::complex< double > EqnRotation::Func25 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 268 of file eqnrotation.cpp.

15.7.3.19 std::complex< double > EqnRotation::Func26 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 274 of file eqnrotation.cpp.

15.7.3.20 std::complex< double > EqnRotation::Func27 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 280 of file eqnrotation.cpp.

15.7.3.21 std::complex< double > EqnRotation::Func28 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 286 of file eqnrotation.cpp.

15.7.3.22 std::complex< double > EqnRotation::Func29 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 292 of file eqnrotation.cpp.

15.7.3.23 std::complex< double > EqnRotation::Func3 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 125 of file eqnrotation.cpp.

15.7.3.24 std::complex< double > EqnRotation::Func30 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 298 of file eqnrotation.cpp.

15.7.3.25 std::complex< double > EqnRotation::Func31 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 304 of file eqnrotation.cpp.

15.7.3.26 std::complex< double > EqnRotation::Func32 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 310 of file eqnrotation.cpp.

15.7.3.27 std::complex< double > EqnRotation::Func33 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 316 of file eqnrotation.cpp.

15.7.3.28 std::complex< double > EqnRotation::Func34 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 322 of file eqnrotation.cpp.

15.7.3.29 std::complex< double > EqnRotation::Func35 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 328 of file eqnrotation.cpp.

15.7.3.30 std::complex< double > EqnRotation::Func36 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 334 of file eqnrotation.cpp.

15.7.3.31 std::complex< double > EqnRotation::Func37 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 340 of file eqnrotation.cpp.

15.7.3.32 std::complex< double > EqnRotation::Func38 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 346 of file eqnrotation.cpp.

15.7.3.33 std::complex< double > EqnRotation::Func39 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 352 of file eqnrotation.cpp.

15.7.3.34 std::complex< double > EqnRotation::Func4 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 132 of file eqnrotation.cpp.

15.7.3.35 std::complex< double > EqnRotation::Func40 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 358 of file eqnrotation.cpp.

15.7.3.36 std::complex< double > EqnRotation::Func41 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 364 of file eqnrotation.cpp.

15.7.3.37 std::complex< double > EqnRotation::Func42 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 370 of file eqnrotation.cpp.

15.7.3.38 std::complex< double > EqnRotation::Func43 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 376 of file eqnrotation.cpp.

15.7.3.39 std::complex< double > EqnRotation::Func44 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 382 of file eqnrotation.cpp.

15.7.3.40 std::complex< double > EqnRotation::Func45 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 388 of file eqnrotation.cpp.

15.7.3.41 std::complex< double > EqnRotation::Func46 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 394 of file eqnrotation.cpp.

15.7.3.42 std::complex< double > EqnRotation::Func47 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 400 of file eqnrotation.cpp.

15.7.3.43 std::complex< double > EqnRotation::Func48 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 406 of file eqnrotation.cpp.

15.7.3.44 std::complex< double > EqnRotation::Func49 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 412 of file eqnrotation.cpp.

15.7.3.45 std::complex< double > EqnRotation::Func5 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 139 of file eqnrotation.cpp.

15.7.3.46 std::complex< double > EqnRotation::Func50 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 418 of file eqnrotation.cpp.

15.7.3.47 std::complex< double > EqnRotation::Func6 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 146 of file eqnrotation.cpp.

15.7.3.48 std::complex< double > EqnRotation::Func7 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 154 of file eqnrotation.cpp.

15.7.3.49 std::complex< double > EqnRotation::Func8 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 161 of file eqnrotation.cpp.

15.7.3.50 std::complex< double > EqnRotation::Func9 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 168 of file eqnrotation.cpp.

15.7.3.51 std::complex< double > EqnRotation::setFormula ( ) [protected], [virtual]

The formula used by the equation of motion.

The formula used by the equation of motion. The formula gets rewritten in a unique form. Rearrange any equations so that they have zero on the right hand size.

Example: If the formula were Ax + By = F, it must be rearranged to: Ax + By - F = 0

The formula can also make use of several math functions provided by the equation of motion object.

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 83 of file eqnrotation.cpp.

The documentation for this class was generated from the following files:

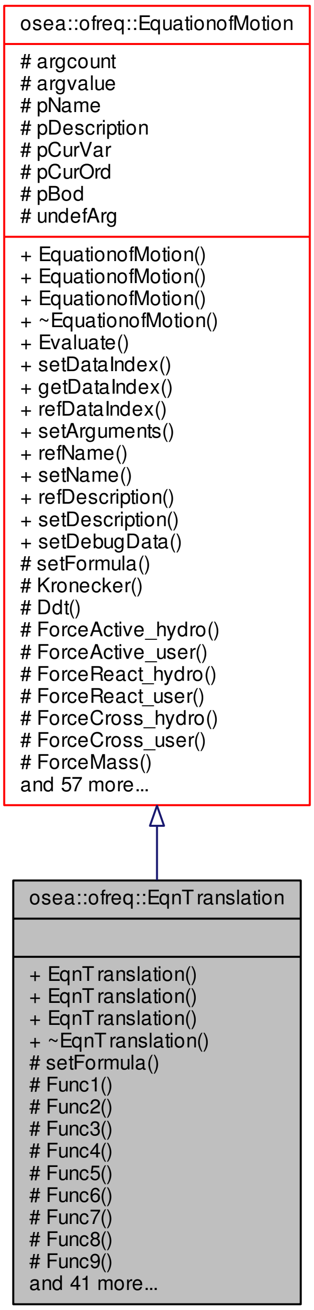
- bin/ofreq/motion_model/eqnrotation.h
- bin/ofreq/motion_model/eqnrotation.cpp

15.8 osea::ofreq::EqnTranslation Class Reference

The EqnTranslation class.

`#include <eqntranslation.h>`

Inheritance diagram for osea::ofreq::EqnTranslation:



Public Member Functions

- EqnTranslation (MotionModel *modelIn)

  Default contrustor. Contains a reference to the motion model class which constructs it.

- EqnTranslation (MotionModel *modelIn, std::string NameIn)

  Contrustor with name. Contains a reference to the motion model class which constructs it.

- EqnTranslation (MotionModel *modelIn, std::string NameIn, int IndexIn)

Contrustor with name and index. Contains a reference to the motion model class which constructs it.
- ∼EqnTranslation ()

  Default destructor.

Protected Member Functions

- std::complex< double > setFormula ()

  The formula used by the equation of motion.
- std::complex< double > Func1 ()

  Func1 through Func50 provide user custom defined functions.
- std::complex< double > Func2 ()
- std::complex< double > Func3 ()
- std::complex< double > Func4 ()
- std::complex< double > Func5 ()
- std::complex< double > Func6 ()
- std::complex< double > Func7 ()
- std::complex< double > Func8 ()
- std::complex< double > Func9 ()
- std::complex< double > Func10 ()
- std::complex< double > Func11 ()
- std::complex< double > Func12 ()
- std::complex< double > Func13 ()
- std::complex< double > Func14 ()
- std::complex< double > Func15 ()
- std::complex< double > Func16 ()
- std::complex< double > Func17 ()
- std::complex< double > Func18 ()
- std::complex< double > Func19 ()
- std::complex< double > Func20 ()
- std::complex< double > Func21 ()
- std::complex< double > Func22 ()
- std::complex< double > Func23 ()
- std::complex< double > Func24 ()
- std::complex< double > Func25 ()
- std::complex< double > Func26 ()
- std::complex< double > Func27 ()
- std::complex< double > Func28 ()
- std::complex< double > Func29 ()
- std::complex< double > Func30 ()
- std::complex< double > Func31 ()
- std::complex< double > Func32 ()
- std::complex< double > Func33 ()
- std::complex< double > Func34 ()
- std::complex< double > Func35 ()
- std::complex< double > Func36 ()
- std::complex< double > Func37 ()
- std::complex< double > Func38 ()
- std::complex< double > Func39 ()
- std::complex< double > Func40 ()
- std::complex< double > Func41 ()
- std::complex< double > Func42 ()
- std::complex< double > Func43 ()
- std::complex< double > Func44 ()
- std::complex< double > Func45 ()

- std::complex< double > Func46 ()
- std::complex< double > Func47 ()
- std::complex< double > Func48 ()
- std::complex< double > Func49 ()
- std::complex< double > Func50 ()

Additional Inherited Members

15.8.1 Detailed Description

The EqnTranslation class.

This class defines motion for linear translation in one of the three principle axes. (X-axis, Y-axis, Z-axis). This is specific to rigid body motion and intended for applicaiton in a six-degree of freedom (6DOF) motion model.

This class is a derived equation of motion. It inherits all the programming needed to define an equation of motion from the class EquationofMotion.

Custom equation classes serve a specific purpose. All the programming necessary to define an equation of motion within the program ofreq is already contained within the class EquationofMotion. All that programming gets inherited from the class. The only thing the custom class needs to define is the specific equation for this purpose. That equation is contained inside the function setFormula(). Everything else is already defined, as far as programming.

There are specific requirements for how to define the custom EquationofMotion class. These instructions assume the user has some basic experience with C++ programming. If not, first try a few tutorials. There are numerous tutorials on the internet. You should specifically familiarize yourself with concepts such as namespaces, classes, objects, and functions. Every effort was made to isolate the user from the complexity of the ofreq program, but some basic understand of program operation is needed.

Assuming you have a basic knowledge of C++ programming, you can create your custom equation of motion by editing the .cpp file associated with this class. Follow these basic rules:

1.) Always create a backup of every file before you edit it.

2.) Only edit the .cpp file. Leave the header file alone (the .h file). The header file creates the linking necessary to make everything work correctly.

3.) Only define your equation within the function setFormula().

4.) Do not add function definitions to the header file unless you are familiar with C++ class programming and know how to safely add the function into the class definition. If you need to define custom functions for your equstions, the safest thing is to define them strictly within your source code file (.cpp file). Just remember that all functions must be fully defined before they get used within the code.

5.) There are several functions inheritted from the EquationofMotion class. You can use these to refer to the different forces when developing your own equation of motion. oFreq recognizes seven (7) basic force types shown below, with the function name to reference them in the EquationofMotion. (Arguments for each function are not shown, for sake of clarity. 5.1) ForceMass() = The forces associated with the mass of an object. This includes direct mass for straight linear motion, and moment of inertia for rotational motion. 5.2) ForceActive_hydro() = The forces which are independant of body motions. The hydro subcategory refers to active forces that specifically come from hydrodynamic forces. This includes the forces from incident waves. Sometimes call the Froude-Krylov forces. 5.3) ForceActive_user() = The forces which are independant of body motions. The user subcategory refers to active forces specifically defined by the user in the ofreq run file. These may be some external force such as an active control system. Regardless, it is customed defined by the user. 5.4) ForceReact_hydro() = The forces which are reactive and dependant on body motions. This includes derivatives of body motions. The hydro subcategory refers to reactive forces hydrodynamic in origin. This would include body hydrostatic properties, added damping, and added mass. 5.5) ForceReact_user() = The forces which are reactive and dependant on body motions. This includes derivatives of body motions. The user subcategory refers to reactive forces defined by the user. This might include external forces such as a mooring line or dynamic positioning system. In any case, these are reactive forces defined at run time in the ofreq input files. 5.6) ForceCross_hydro() = The forces which are reactive and dependant on the body motions of another body. This is only applicable to multi-body systems. Examples might be two vessels near each other. The program can accept equations that use the cross-body forces but are only applied to a single body problem. The hydro subcategory refers to reactive forces hydrodynamic in origin. This would include body hydrostatic properties, added damping, and added mass, except that these forces would be dependant on the motions of another body. 5.7) ForceCross_user() = The forces which are reactive and dependant on the body motions of another body. This is only applicable to multi-body systems. Examples might be two vessels near each other. The program can accept equations that use the cross-body forces but are only applied to a single body problem. The user subcategory refers to reactive forces defined by the user. This might include external forces such as a mooring line or dynamic positioning system. In any case, these are reactive forces defined at run time in the ofreq input files.

6.) Use of the Sum() Function. There are three possible implementations of the Sum() function. The input syntax determines which function to use. 6.1) Sum a finite value: This implementation occurrs when a variable is provided as the argument for for the summation. The variable must be of data type complex<double>. The variable will not change during the summation. Variable is passed by value. 6.2) Sum a function contained within the class: This is the most common implementation of the Sum() function. The class has 50 functions provided for your use. They are named Func1 through Func50. You may enter any code within these functions. But the functions do not accept any inputs. This is a limitation of program. The functions will update with each iteration of the Sum() function. Anyhthing that you wish to change during summation must be captured within one of the custom functions. This also includes references to any other class functions. To implement the custom function, you simply type in the function name as a string input. Example: Sum("Func1()", "body", 0, 1)

And then the function definition for Func1 would be: Func1() { return ForceReact_hydro(ord(), var()) ∗ Ddt(var(), ord()); }

This was just one example. Any combination may be used within the custom function. 6.3) Sum a function not contained within the class. This is mostly used for debugging when you wish to test a custom equation of motion, isolated from the main program. The returned data type from the function must be complex<double>. To use your external function within the Sum() function, you must enter as a function pointer. The Sum function expects a pointer to a function. You would enter it as follows (all capitals are the terms you change for your specific function):

output = Sum(&FUNCTION_NAME, index, from, to);

Two key points to notice: The function name was preceded with a reference symbol (&); and I only stated the function name. I did not include the brackets to explicitely state that it's a function. Don't include the brackets. You will get a compiler error if you do.

See Also

EquationofMotion
MotionModel

Definition at line 186 of file eqntranslation.h.

15.8.2 Constructor & Destructor Documentation

15.8.2.1 EqnTranslation::EqnTranslation (**MotionModel** ∗ *modelIn*)

Default contrustor. Contains a reference to the motion model class which constructs it.

Default contrustor. Contains a reference to the motion model class which constructs it. The constructing class is necessary because several functions in the EquationOfMotion class use data in the constructing class, the motion model class.

Parameters

modelIn	A pointer to the motion model object that created the equation of motion.

Definition at line 38 of file eqntranslation.cpp.

15.8.2.2 EqnTranslation::EqnTranslation (MotionModel * *modelIn*, std::string *NameIn*)

Contrustor with name. Contains a reference to the motion model class which constructs it.

Default contrustor. Contains a reference to the motion model class which constructs it. The constructing class is necessary because several functions in the EquationOfMotion class use data in the constructing class, the motion model class.

Parameters

modelIn	A pointer to the motion model object that created the equation of motion.
NameIn	A name for what physical property the equation solves for. Used for user output. Not critical to program execution.

Definition at line 45 of file eqntranslation.cpp.

15.8.2.3 EqnTranslation::EqnTranslation (MotionModel * *modelIn*, std::string *NameIn*, int *IndexIn*)

Contrustor with name and index. Contains a reference to the motion model class which constructs it.

Default contrustor. Contains a reference to the motion model class which constructs it. The constructing class is necessary because several functions in the EquationOfMotion class use data in the constructing class, the motion model class.

Parameters

modelIn	A pointer to the motion model object that created the equation of motion.
NameIn	A name for what physical property the equation solves for. Used for user output. Not critical to program execution.
IndexIn	Sets the index for the Equation of Motion. The index is how the equation determines which numbers to access on the data. The following indices are used. Any higher indices can extend beyond this range, and the program easily adapts. But the following three are reserved. Unused indices are not transferred to the matrices when solved. So unused indices to not negatively impact calculation performance. However, using excessively large indices (say 500 when you only have 3 equations) will result in large matrices and unecessary memory requirements. THe following index reservations apply. 1: Translation in x-direction. Specific to rigid body motion. 2: Translation in y-direction. Specific to rigid body motion. 3: Translation in z-direction. Specific to rigid body motion. 4: Rotation about x-direction. Specific to rigid body motion. 5: Rotation about y-direction. Specific to rigid body motion. 6: Rotation about z-direction. Specific to rigid body motion.

Definition at line 52 of file eqntranslation.cpp.

15.8.2.4 EqnTranslation::∼EqnTranslation ( )

Default destructor.

Definition at line 59 of file eqntranslation.cpp.

15.8.3 Member Function Documentation

15.8.3.1 std::complex< double > EqnTranslation::Func1 ( ) [protected], [virtual]

Func1 through Func50 provide user custom defined functions.

These are custom functions that the user may need to create to define their equations of motion. The only restriction is that the functions can not take any arguments. Any arguments required must be supplied through a set of global variables. Sorry, that's just a restriction of how the code is written and the use of the C++ language.

Returns

Returns a complex<double> variable. Returned variabled passed by value.

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 139 of file eqntranslation.cpp.

15.8.3.2 std::complex< double > EqnTranslation::Func10 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 204 of file eqntranslation.cpp.

15.8.3.3 std::complex< double > EqnTranslation::Func11 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 211 of file eqntranslation.cpp.

15.8.3.4 std::complex< double > EqnTranslation::Func12 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 217 of file eqntranslation.cpp.

15.8.3.5 std::complex< double > EqnTranslation::Func13 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 223 of file eqntranslation.cpp.

15.8.3.6 std::complex< double > EqnTranslation::Func14 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 229 of file eqntranslation.cpp.

15.8.3.7 std::complex< double > EqnTranslation::Func15 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 235 of file eqntranslation.cpp.

15.8.3.8 std::complex< double > EqnTranslation::Func16 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 241 of file eqntranslation.cpp.

15.8.3.9 std::complex< double > EqnTranslation::Func17 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 247 of file eqntranslation.cpp.

15.8.3.10 std::complex< double > EqnTranslation::Func18 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 253 of file eqntranslation.cpp.

15.8.3.11 std::complex< double > EqnTranslation::Func19 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 259 of file eqntranslation.cpp.

15.8.3.12 std::complex< double > EqnTranslation::Func2 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 146 of file eqntranslation.cpp.

15.8.3.13 std::complex< double > EqnTranslation::Func20 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 265 of file eqntranslation.cpp.

15.8.3.14 std::complex< double > EqnTranslation::Func21 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 271 of file eqntranslation.cpp.

15.8.3.15 std::complex< double > EqnTranslation::Func22 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 277 of file eqntranslation.cpp.

15.8.3.16 std::complex< double > EqnTranslation::Func23 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 283 of file eqntranslation.cpp.

15.8.3.17 std::complex< double > EqnTranslation::Func24 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 289 of file eqntranslation.cpp.

15.8.3.18 std::complex< double > EqnTranslation::Func25 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 295 of file eqntranslation.cpp.

15.8.3.19 std::complex< double > EqnTranslation::Func26 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 301 of file eqntranslation.cpp.

15.8.3.20 std::complex< double > EqnTranslation::Func27 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 307 of file eqntranslation.cpp.

15.8.3.21 std::complex< double > EqnTranslation::Func28 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 313 of file eqntranslation.cpp.

15.8.3.22 std::complex< double > EqnTranslation::Func29 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 319 of file eqntranslation.cpp.

15.8.3.23 std::complex< double > EqnTranslation::Func3 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 153 of file eqntranslation.cpp.

15.8.3.24 std::complex< double > EqnTranslation::Func30 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 325 of file eqntranslation.cpp.

15.8.3.25 std::complex< double > EqnTranslation::Func31 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 331 of file eqntranslation.cpp.

15.8.3.26 std::complex< double > EqnTranslation::Func32 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 337 of file eqntranslation.cpp.

15.8.3.27 std::complex< double > EqnTranslation::Func33 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 343 of file eqntranslation.cpp.

15.8.3.28 std::complex< double > EqnTranslation::Func34 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 349 of file eqntranslation.cpp.

15.8.3.29 std::complex< double > EqnTranslation::Func35 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 355 of file eqntranslation.cpp.

15.8.3.30 std::complex< double > EqnTranslation::Func36 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 361 of file eqntranslation.cpp.

15.8.3.31 std::complex< double > EqnTranslation::Func37 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 367 of file eqntranslation.cpp.

15.8.3.32 std::complex< double > EqnTranslation::Func38 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 373 of file eqntranslation.cpp.

15.8.3.33 std::complex< double > EqnTranslation::Func39 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 379 of file eqntranslation.cpp.

15.8.3.34 std::complex< double > EqnTranslation::Func4 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 160 of file eqntranslation.cpp.

15.8.3.35 std::complex< double > EqnTranslation::Func40 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 385 of file eqntranslation.cpp.

15.8.3.36 std::complex< double > EqnTranslation::Func41 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 391 of file eqntranslation.cpp.

15.8.3.37 std::complex< double > EqnTranslation::Func42 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 397 of file eqntranslation.cpp.

15.8.3.38 std::complex< double > EqnTranslation::Func43 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 403 of file eqntranslation.cpp.

15.8.3.39 std::complex< double > EqnTranslation::Func44 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 409 of file eqntranslation.cpp.

15.8.3.40 std::complex< double > EqnTranslation::Func45 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 415 of file eqntranslation.cpp.

15.8.3.41 std::complex< double > EqnTranslation::Func46 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 421 of file eqntranslation.cpp.

15.8.3.42 std::complex< double > EqnTranslation::Func47 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 427 of file eqntranslation.cpp.

15.8.3.43 std::complex< double > EqnTranslation::Func48 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 433 of file eqntranslation.cpp.

15.8.3.44 std::complex< double > EqnTranslation::Func49 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 439 of file eqntranslation.cpp.

15.8.3.45 std::complex< double > EqnTranslation::Func5 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 167 of file eqntranslation.cpp.

15.8.3.46 std::complex< double > EqnTranslation::Func50 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 445 of file eqntranslation.cpp.

15.8.3.47 std::complex< double > EqnTranslation::Func6 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 174 of file eqntranslation.cpp.

15.8.3.48 std::complex< double > EqnTranslation::Func7 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 182 of file eqntranslation.cpp.

15.8.3.49 std::complex< double > EqnTranslation::Func8 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 189 of file eqntranslation.cpp.

15.8.3.50 std::complex< double > EqnTranslation::Func9 ( ) [protected], [virtual]

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 197 of file eqntranslation.cpp.

15.8.3.51 complex< double > EqnTranslation::setFormula ( ) [protected], [virtual]

The formula used by the equation of motion.

The formula used by the equation of motion. The formula gets rewritten in a unique form. Rearrange any equations so that they have zero on the right hand size.

Example: If the formula were Ax + By = F, it must be rearranged to: Ax + By - F = 0

The formula can also make use of several math functions provided by the equation of motion object.

Reimplemented from osea::ofreq::EquationofMotion.

Definition at line 84 of file eqntranslation.cpp.

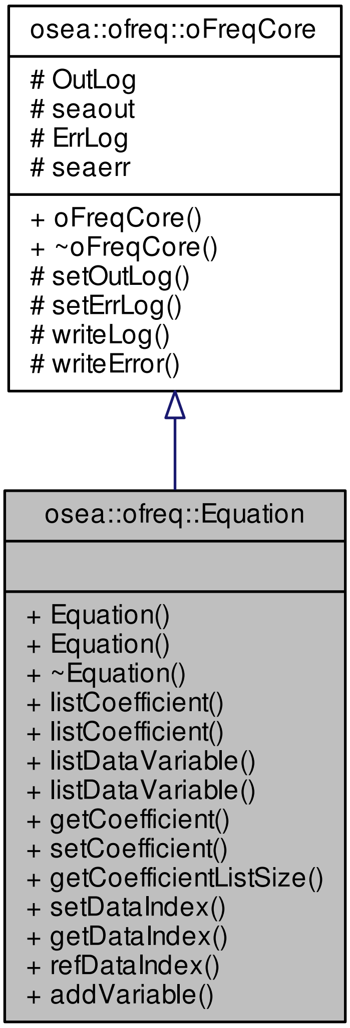
The documentation for this class was generated from the following files:

- bin/ofreq/motion_model/eqntranslation.h
- bin/ofreq/motion_model/eqntranslation.cpp

15.9 osea::ofreq::Equation Class Reference

```
#include <equation.h>
```

Inheritance diagram for osea::ofreq::Equation:



Public Member Functions

- Equation ()
- Equation (int IndexIn)

  Constructor with setting the data index.
- ∼Equation ()
- std::vector< double > & listCoefficient ()

  Direct access to the list of coefficients.
- double & listCoefficient (unsigned int index)

  Provides direct access to a coefficient from the list of coefficients.
- std::vector< double > & listDataVariable ()

  Provides direct access to the list of coefficients.
- double & listDataVariable (int DataIndex)

Provides direct access to a coefficient from the list of coefficients.
- double getCoefficient (int number)

  Get the coefficient at the specified number.
- void setCoefficient (int number, double coeffIn)

  Set the coefficient value at the specified index number.
- unsigned int getCoefficientListSize ()
- void setDataIndex (int index)

  Set the index number of any equation data that should be retrieved.
- int getDataIndex ()

  Get the index number of any equation data that should be retrieved.
- int & refDataIndex ()

  Exposed access to the data access variable.
- void addVariable (double CoeffIn, int VarDataIn=-1)

  This function adds a data variable to the list of variables contained in the equation.

Additional Inherited Members

15.9.1 Detailed Description

This class holds data for an equation.

Definition at line 83 of file equation.h.

15.9.2 Constructor & Destructor Documentation

15.9.2.1 Equation::Equation ( )

This default constructor.

Definition at line 36 of file equation.cpp.

15.9.2.2 Equation::Equation (int *IndexIn*)

Constructor with setting the data index.

Constructor with setting the data index.

Parameters

IndexIn	The integer specifying the data index number.

Definition at line 45 of file equation.cpp.

15.9.2.3 Equation::∼Equation ( )

The default destructor, nothing happens here.

Definition at line 53 of file equation.cpp.

15.9.3 Member Function Documentation

15.9.3.1 void Equation::addVariable (double *CoeffIn*, int *VarDataIn* = -1)

This function adds a data variable to the list of variables contained in the equation.

Two parameters are necessary to add data variable to the list. The first parameter records the actual coefficient used in the equation. The second parameter records the data index for the new entry.

Parameters

CoeffIn	Double. The actual coefficient used in the equation. Variable passed by value.
VarDataIn	Integer. The data index of the new variable passed to the equation. Variable passed by value. If no value is provided, the function defaults to using the coefficients index in the containin vector.

Definition at line 143 of file equation.cpp.

15.9.3.2 double Equation::getCoefficient (int *number*)

Get the coefficient at the specified number.

Parameters

number	The index number of the coefficient to retrieve.

Returns

Returns a double precision floating point number of the coefficient at the index specified by number.

Definition at line 99 of file equation.cpp.

15.9.3.3 unsigned int Equation::getCoefficientListSize ( )

Retrieve the size of the coefficient list.

Returns

The size of the coefficient list.

Definition at line 119 of file equation.cpp.

15.9.3.4 int Equation::getDataIndex ( )

Get the index number of any equation data that should be retrieved.

Get the index number of any equation data that should be retrieved. Because the first six values in the index are reserved for 6DOF, it is necessary that equation objects should be able to specify their index as something other than their place in a containing vector. The default initialization value for this is -1, which indicates the index is not set. Any number less than zero indicates the index is not set.

Parameters

index	Integer. The index number that should be retrieved. Any number less than zero indicates the index is not set.

Definition at line 131 of file equation.cpp.

15.9.3.5 vector< double > & Equation::listCoefficient ( )

Direct access to the list of coefficients.

Returns

The list of coefficients. Returned variable passed by reference.

Definition at line 58 of file equation.cpp.

15.9.3.6 double & Equation::listCoefficient (unsigned int *index*)

Provides direct access to a coefficient from the list of coefficients.

Returns a value from the list of coefficents. Which value to return is specified by the input index.

Parameters

index	Unsigned integer. Specifies which value to return from the list of coefficients.

Returns

Returns a double. Returned variable is a value from the list of coefficients. Returned variable is passed by reference.

Definition at line 65 of file equation.cpp.

15.9.3.7 std::vector< double > & Equation::listDataVariable ( )

Provides direct access to the list of coefficients.

Returns

The list of coefficients. Returned variable passed by reference.

See Also

listCoefficient()

Definition at line 71 of file equation.cpp.

15.9.3.8 double & Equation::listDataVariable (int *DataIndex*)

Provides direct access to a coefficient from the list of coefficients.

Returns a value from the list of coefficents. Which value to return is specified by the data index. This is like the listCoefficient method. But that method returned values based on the index of occurrence in the vector. This method returns values based on the specified data index property.

Parameters

DataIndex	Integer. Specifies which value to return from the list of coefficients. Specification is by the data index of each variable.

Returns

Returns a double. Returned variable is a value from the list of coefficients. Returned variable is passed by reference.

See Also

listCoefficient(index)

Definition at line 77 of file equation.cpp.

15.9.3.9 int & Equation::refDataIndex ( )

Exposed access to the data access variable.

Returns

Returns the data access variable. Return passed by reference.

Definition at line 137 of file equation.cpp.

15.9.3.10 void Equation::setCoefficient (int *number*, double *coeffIn*)

Set the coefficient value at the specified index number.

Set the coefficient value at the specified index number.

Parameters

number	Integer. The index number of the coefficient to set.
coeffIn	Double precision floating number. The value of the coefficient to set at the specified index.

Definition at line 106 of file equation.cpp.

15.9.3.11 void Equation::setDataIndex (int *index*)

Set the index number of any equation data that should be retrieved.

Set the index number of any equation data that should be retrieved. Because the first six values in the index are reserved for 6DOF, it is necessary that equation objects should be able to specify their index as something other than their place in a containing vector. The default initialization value for this is -1, which indicates the index is not set. Any number less than zero indicates the index is not set.

Parameters

index	The index number that should be set. Any number less than zero indicates the index is not set.

Definition at line 125 of file equation.cpp.

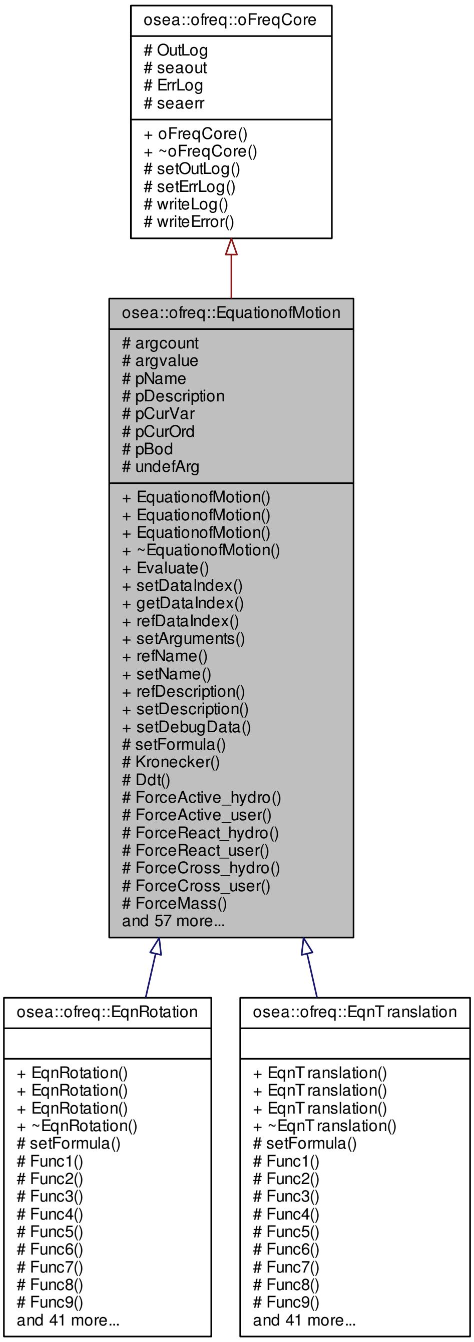
The documentation for this class was generated from the following files:

- bin/ofreq/global_objects/equation.h
- bin/ofreq/global_objects/equation.cpp

15.10 osea::ofreq::EquationofMotion Class Reference

```
#include <equationofmotion.h>
```

Inheritance diagram for osea::ofreq::EquationofMotion:



Public Member Functions

- EquationofMotion (MotionModel ∗modelIn)

  Default contrustor. Contains a reference to the motion model class which constructs it.
- EquationofMotion (MotionModel ∗modelIn, std::string NameIn)

  Contrustor with name. Contains a reference to the motion model class which constructs it.
- EquationofMotion (MotionModel ∗modelIn, std::string NameIn, int IndexIn)

Contrustor with name and index. Contains a reference to the motion model class which constructs it.

- virtual ∼EquationofMotion ()

  Default destructor.
- virtual std::complex< double > Evaluate ()

  Triggers evaluation of the equation of motion object.
- void setDataIndex (int DataIn)

  Sets the index for the equation of motion.
- int getDataIndex ()

  Gets the index for the equation of motion.
- int & refDataIndex ()

  Gets the index for the equation of motion.
- void setArguments (int argn, std::vector< double > argv)

  Sets any values for arguments that may be used by the equation of motion.
- std::string & refName ()

  The name for the equation object.
- void setName (std::string nameIn)

  The name for the equation object.
- std::string & refDescription ()

  The description for the equation object.
- void setDescription (std::string descIn)

  The description for the equation object.
- void setDebugData (double freqIn, std::complex< double > solnIn, bool coeffIn=false)

  Sets debugging data to use when creating fictional inputs purely for debugging this function. Allows the programmer to debug the function independent of the other functions which depend on it.

Protected Member Functions

- virtual std::complex< double > setFormula ()

  The formula used by the equation of motion.
- std::complex< double > Kronecker (int var1, int var2, bool anti=false)

  The mathematical Kronecker delta function.
- std::complex< double > Ddt (int var, int ord, int bodIn=-1)

  Time differential function.
- std::complex< double > ForceActive_hydro ()

  A reference to the data set of the ForceActive_hydro.
- std::complex< double > ForceActive_user ()

  A reference to the data set of the ForceActive_user.
- std::complex< double > ForceReact_hydro (unsigned int ordIn, unsigned int varIn)

  A reference to the data set of the ForceReact_hydro.
- std::complex< double > ForceReact_user (unsigned int ordIn, unsigned int varIn)

  A reference to the data set of the ForceReact_user.
- std::complex< double > ForceCross_hydro (unsigned int bodIn, unsigned int ordIn, unsigned int varIn)

  A reference to the data set of the ForceCross_hydro.
- std::complex< double > ForceCross_user (unsigned int bodIn, unsigned int ordIn, unsigned int varIn)

  A reference to the data set of the ForceCross_user.
- std::complex< double > ForceMass (int varIn)

  A reference to the data set of the ForceMass.
- int var ()

  Returns the index integer for iteration on variable.
- int ord ()

  Returns the index integer for iteration on order of derviative.

- int body ()

  Retuns the index integer for the body in the list of bodies.
- int curbody ()

  Returns the index integer for the current body used by the motion model that created this equation of motion.
- std::complex< double > Sum (std::string FuncName, std::string index, int from=-1, int to=-1)

  Sums across a variable.
- std::complex< double > Sum (std::complex< double >(∗force)(void), int from, int to)

  Sums a function multiple times.
- std::complex< double > Sum (std::complex< double > force, int from, int to)

  Sums a constant value multiple times.
- virtual std::complex< double > Func1 ()

  Func1 through Func50 provide user custom defined functions.
- virtual std::complex< double > Func2 ()
- virtual std::complex< double > Func3 ()
- virtual std::complex< double > Func4 ()
- virtual std::complex< double > Func5 ()
- virtual std::complex< double > Func6 ()
- virtual std::complex< double > Func7 ()
- virtual std::complex< double > Func8 ()
- virtual std::complex< double > Func9 ()
- virtual std::complex< double > Func10 ()
- virtual std::complex< double > Func11 ()
- virtual std::complex< double > Func12 ()
- virtual std::complex< double > Func13 ()
- virtual std::complex< double > Func14 ()
- virtual std::complex< double > Func15 ()
- virtual std::complex< double > Func16 ()
- virtual std::complex< double > Func17 ()
- virtual std::complex< double > Func18 ()
- virtual std::complex< double > Func19 ()
- virtual std::complex< double > Func20 ()
- virtual std::complex< double > Func21 ()
- virtual std::complex< double > Func22 ()
- virtual std::complex< double > Func23 ()
- virtual std::complex< double > Func24 ()
- virtual std::complex< double > Func25 ()
- virtual std::complex< double > Func26 ()
- virtual std::complex< double > Func27 ()
- virtual std::complex< double > Func28 ()
- virtual std::complex< double > Func29 ()
- virtual std::complex< double > Func30 ()
- virtual std::complex< double > Func31 ()
- virtual std::complex< double > Func32 ()
- virtual std::complex< double > Func33 ()
- virtual std::complex< double > Func34 ()
- virtual std::complex< double > Func35 ()
- virtual std::complex< double > Func36 ()
- virtual std::complex< double > Func37 ()
- virtual std::complex< double > Func38 ()
- virtual std::complex< double > Func39 ()
- virtual std::complex< double > Func40 ()
- virtual std::complex< double > Func41 ()
- virtual std::complex< double > Func42 ()
- virtual std::complex< double > Func43 ()

- virtual std::complex< double > Func44 ()
- virtual std::complex< double > Func45 ()
- virtual std::complex< double > Func46 ()
- virtual std::complex< double > Func47 ()
- virtual std::complex< double > Func48 ()
- virtual std::complex< double > Func49 ()
- virtual std::complex< double > Func50 ()

Protected Attributes

- int argcount

  Used to supply arguments to the equation of motion. Recods the number of arguments.
- std::vector< double > argvalue

  Used to supply arguments to the equation of motion. Uknown, arbitrary double precision values. A vector of uknown size.
- std::string pName

  The name for the equation object.
- std::string pDescription

  The description for the equation object.
- unsigned int pCurVar

  The integer of the current value of var() index. Used for iteration and summation functions.
- unsigned int pCurOrd

  The integer of the current value of ord() index. Used for iteration and summation functions.
- unsigned int pBod

  The integer of the current body. Used for iteration and summation functions.

Static Protected Attributes

- static int undefArg = -1

Additional Inherited Members

15.10.1 Detailed Description

The Equation of motion class defines a single equation of motion. Each object of the class represents a new instance. This is the base class, which gets inheritted by any custom class. The only major definition added to any inherrited class is the actual formula definition for the equation. It may be that the equation are repetitions of the same sequence, just with a different equation index. In that case, multiple instances of the same class can be created and the equation index changed. This can save on typing. Or, if the equations are truly different for each equation, you can create a separate equation of motion class for each equation, and initiate with just one object from each class.

In addition to the regular object entries, the class also has provision for a list of arbitrary arguments.

Definition at line 101 of file equationofmotion.h.

15.10.2 Constructor & Destructor Documentation

15.10.2.1 EquationofMotion::EquationofMotion (**MotionModel** * *modelIn*)

Default contrustor. Contains a reference to the motion model class which constructs it.

Default contrustor. Contains a reference to the motion model class which constructs it. The constructing class is necessary because several functions in the EquationOfMotion class use data in the constructing class, the motion model class.

Parameters

modelIn	A pointer to the motion model object that created the equation of motion.

Definition at line 46 of file equationofmotion.cpp.

15.10.2.2 osea::ofreq::EquationofMotion::EquationofMotion (MotionModel * *modelIn*, std::string *NameIn*)

Contrustor with name. Contains a reference to the motion model class which constructs it.

Default contrustor. Contains a reference to the motion model class which constructs it. The constructing class is necessary because several functions in the EquationOfMotion class use data in the constructing class, the motion model class.

Parameters

modelIn	A pointer to the motion model object that created the equation of motion.
NameIn	A name for what physical property the equation solves for. Used for user output. Not critical to program execution.

15.10.2.3 osea::ofreq::EquationofMotion::EquationofMotion (MotionModel * *modelIn*, std::string *NameIn*, int *IndexIn*)

Contrustor with name and index. Contains a reference to the motion model class which constructs it.

Default contrustor. Contains a reference to the motion model class which constructs it. The constructing class is necessary because several functions in the EquationOfMotion class use data in the constructing class, the motion model class.

Parameters

modelIn	A pointer to the motion model object that created the equation of motion.
NameIn	A name for what physical property the equation solves for. Used for user output. Not critical to program execution.
IndexIn	Sets the index for the Equation of Motion. The index is how the equation determines which numbers to access on the data. The following indices are used. Any higher indices can extend beyond this range, and the program easily adapts. But the following three are reserved. Unused indices are not transferred to the matrices when solved. So unused indices to not negatively impact calculation performance. However, using excessively large indices (say 500 when you only have 3 equations) will result in large matrices and unecessary memory requirements. THe following index reservations apply. 1: Translation in x-direction. Specific to rigid body motion. 2: Translation in y-direction. Specific to rigid body motion. 3: Translation in z-direction. Specific to rigid body motion. 4: Rotation about x-direction. Specific to rigid body motion. 5: Rotation about y-direction. Specific to rigid body motion. 6: Rotation about z-direction. Specific to rigid body motion.

15.10.2.4 EquationofMotion::∼EquationofMotion ( ) [virtual]

Default destructor.

Definition at line 81 of file equationofmotion.cpp.

15.10.3 Member Function Documentation

15.10.3.1 int EquationofMotion::body ( ) [protected]

Retuns the index integer for the body in the list of bodies.

This is used for summation functions when iterating through each body in the list of bodies. This index cannot be modified through this function. It is purely meant for access to the variable.

Returns

Returns the index integer for each body in the list of bodies.

Definition at line 717 of file equationofmotion.cpp.

15.10.3.2 int EquationofMotion::curbody ( ) [protected]

Returns the index integer for the current body used by the motion model that created this equation of motion.

This index cannot be modified through this function. It is purely meant for access of the variable.

Returns

Returns the index integer for the current body used by the motion model that created this equation of motions.

Definition at line 724 of file equationofmotion.cpp.

15.10.3.3 complex< double > EquationofMotion::Ddt (int *var*, int *ord*, int *bodIn* = -1) [protected]

Time differential function.

Time differential function. Used to calculate the time derivative of a reponse. Can convert from response amplitude to velocity to acceleration, and further. Used to calculated amplitude of response.

Parameters

var	Index of the variable to use for the time differential. If included with the function var(), the index is automatically determined by the summation functions that you include Ddt() into.
ord	Integer. The order of the differential. If the function ord() is used, the order is automatically determined by the summation function that you include Ddt() into.
bodIn	The body to retrieve variable data for.

Returns

Returns a complex value that is the time differential, transposed into a frequency domain. If absolute values of response were desired, the function will include the effects of response amplitude.

Definition at line 219 of file equationofmotion.cpp.

15.10.3.4 complex< double > EquationofMotion::Evaluate ( ) [virtual]

Triggers evaluation of the equation of motion object.

Returns

Returns a complex number that is the result of evaluating the equation of motion object.

Definition at line 87 of file equationofmotion.cpp.

15.10.3.5 complex< double > EquationofMotion::ForceActive_hydro ( ) [protected]

A reference to the data set of the ForceActive_hydro.

Returns

Returns the data set for the ForceActive_hydro. Indices can be specified to access individual elements.

Definition at line 282 of file equationofmotion.cpp.

15.10.3.6 complex< double > EquationofMotion::ForceActive_user ( ) [protected]

A reference to the data set of the ForceActive_user.

Returns

Returns the data set for the ForceActive_user. Indices can be specified to access individual elements.

Definition at line 328 of file equationofmotion.cpp.

15.10.3.7 complex< double > EquationofMotion::ForceCross_hydro (unsigned int *bodIn*, unsigned int *ordIn*, unsigned int *varIn*) [protected]

A reference to the data set of the ForceCross_hydro.

Parameters

bodIn	Integer. Represents the input variable for the body that the cross body force is linked to.
ordIn	Integer. Represents the input variable for the order of derivative.
varIn	Integer. Represents the input varaible for the variable.

Returns

Returns the data set for the ForceCross_hydro. Indices can be specified to access individual elements.

Definition at line 492 of file equationofmotion.cpp.

15.10.3.8 complex< double > EquationofMotion::ForceCross_user (unsigned int *bodIn*, unsigned int *ordIn*, unsigned int *varIn*) [protected]

A reference to the data set of the ForceCross_user.

Parameters

bodIn	Integer. Represents the input variable for the body that the cross body force is linked to.
ordIn	Integer. Represents the input variable for the order of derivative.
varIn	Integer. Represents the input varaible for the variable.

Returns

Returns the data set for the ForceCross_user. Indices can be specified to access individual elements.

Definition at line 556 of file equationofmotion.cpp.

15.10.3.9 complex< double > EquationofMotion::ForceMass (int *varIn*) [protected]

A reference to the data set of the ForceMass.

Parameters

varIn	Integer. Represents the input index for the variable.

Returns

Returns the data set for the ForceMass. Indices can be specified to access individual elements.

Definition at line 619 of file equationofmotion.cpp.

15.10.3.10 complex< double > EquationofMotion::ForceReact_hydro (unsigned int *ordIn,* unsigned int *varIn*) [protected]

A reference to the data set of the ForceReact_hydro.

Parameters

ordIn	Integer. Represents the input variable for the order of derivative.
varIn	Integer. Represents the input varaible for the variable.

Returns

Returns the data set for the ForceReact_hydro. Indices can be specified to access individual elements.

Definition at line 378 of file equationofmotion.cpp.

15.10.3.11 complex< double > EquationofMotion::ForceReact_user (unsigned int *ordIn,* unsigned int *varIn*) [protected]

A reference to the data set of the ForceReact_user.

Parameters

ordIn	Integer. Represents the input variable for the order of derivative.
varIn	Integer. Represents the input varaible for the variable.

Returns

Returns the data set for the ForceReact_user. Indices can be specified to access individual elements.

Definition at line 435 of file equationofmotion.cpp.

15.10.3.12 std::complex< double > EquationofMotion::Func1 ( ) [protected], [virtual]

Func1 through Func50 provide user custom defined functions.

These are custom functions that the user may need to create to define their equations of motion. The only restriction is that the functions can not take any arguments. Any arguments required must be supplied through a set of global variables. Sorry, that's just a restriction of how the code is written and the use of the C++ language.

Returns

Returns a complex<double> variable. Returned variabled passed by value.

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 884 of file equationofmotion.cpp.

15.10.3.13 std::complex< double > EquationofMotion::Func10 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 938 of file equationofmotion.cpp.

15.10.3.14 std::complex< double > EquationofMotion::Func11 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 944 of file equationofmotion.cpp.

15.10.3.15 std::complex< double > EquationofMotion::Func12 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 950 of file equationofmotion.cpp.

15.10.3.16 std::complex< double > EquationofMotion::Func13 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 956 of file equationofmotion.cpp.

15.10.3.17 std::complex< double > EquationofMotion::Func14 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 962 of file equationofmotion.cpp.

15.10.3.18 std::complex< double > EquationofMotion::Func15 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 968 of file equationofmotion.cpp.

15.10.3.19 std::complex< double > EquationofMotion::Func16 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 974 of file equationofmotion.cpp.

15.10.3.20 std::complex< double > EquationofMotion::Func17 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 980 of file equationofmotion.cpp.

15.10.3.21 std::complex< double > EquationofMotion::Func18 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 986 of file equationofmotion.cpp.

15.10.3.22 std::complex< double > EquationofMotion::Func19 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 992 of file equationofmotion.cpp.

15.10.3.23 std::complex< double > EquationofMotion::Func2 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 890 of file equationofmotion.cpp.

15.10.3.24 std::complex< double > EquationofMotion::Func20 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 998 of file equationofmotion.cpp.

15.10.3.25 std::complex< double > EquationofMotion::Func21 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1004 of file equationofmotion.cpp.

15.10.3.26 std::complex< double > EquationofMotion::Func22 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1010 of file equationofmotion.cpp.

15.10.3.27 std::complex< double > EquationofMotion::Func23 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1016 of file equationofmotion.cpp.

15.10.3.28 std::complex< double > EquationofMotion::Func24 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1022 of file equationofmotion.cpp.

15.10.3.29 std::complex< double > EquationofMotion::Func25 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1028 of file equationofmotion.cpp.

15.10.3.30 std::complex< double > EquationofMotion::Func26 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1034 of file equationofmotion.cpp.

15.10.3.31 std::complex< double > EquationofMotion::Func27 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1040 of file equationofmotion.cpp.

15.10.3.32 std::complex< double > EquationofMotion::Func28 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1046 of file equationofmotion.cpp.

15.10.3.33 std::complex< double > EquationofMotion::Func29 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1052 of file equationofmotion.cpp.

15.10.3.34 std::complex< double > EquationofMotion::Func3 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 896 of file equationofmotion.cpp.

15.10.3.35 std::complex< double > EquationofMotion::Func30 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1058 of file equationofmotion.cpp.

15.10.3.36 std::complex< double > EquationofMotion::Func31 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1064 of file equationofmotion.cpp.

15.10.3.37 std::complex< double > EquationofMotion::Func32 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1070 of file equationofmotion.cpp.

15.10.3.38 std::complex< double > EquationofMotion::Func33 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1076 of file equationofmotion.cpp.

15.10.3.39 std::complex< double > EquationofMotion::Func34 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1082 of file equationofmotion.cpp.

15.10.3.40 std::complex< double > EquationofMotion::Func35 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1088 of file equationofmotion.cpp.

15.10.3.41 std::complex< double > EquationofMotion::Func36 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1094 of file equationofmotion.cpp.

15.10.3.42 std::complex< double > EquationofMotion::Func37 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1100 of file equationofmotion.cpp.

15.10.3.43 std::complex< double > EquationofMotion::Func38 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1106 of file equationofmotion.cpp.

15.10.3.44 std::complex< double > EquationofMotion::Func39 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1112 of file equationofmotion.cpp.

15.10.3.45 std::complex< double > EquationofMotion::Func4 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 902 of file equationofmotion.cpp.

15.10.3.46 std::complex< double > EquationofMotion::Func40 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1118 of file equationofmotion.cpp.

15.10.3.47 std::complex< double > EquationofMotion::Func41 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1124 of file equationofmotion.cpp.

15.10.3.48 std::complex< double > EquationofMotion::Func42 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1130 of file equationofmotion.cpp.

15.10.3.49 std::complex< double > EquationofMotion::Func43 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1136 of file equationofmotion.cpp.

15.10.3.50 std::complex< double > EquationofMotion::Func44 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1142 of file equationofmotion.cpp.

15.10.3.51 std::complex< double > EquationofMotion::Func45 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1148 of file equationofmotion.cpp.

15.10.3.52 std::complex< double > EquationofMotion::Func46 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1154 of file equationofmotion.cpp.

15.10.3.53 std::complex< double > EquationofMotion::Func47 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1160 of file equationofmotion.cpp.

15.10.3.54 std::complex< double > EquationofMotion::Func48 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1166 of file equationofmotion.cpp.

15.10.3.55 std::complex< double > EquationofMotion::Func49 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1172 of file equationofmotion.cpp.

15.10.3.56 std::complex< double > EquationofMotion::Func5 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 908 of file equationofmotion.cpp.

15.10.3.57 std::complex< double > EquationofMotion::Func50 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 1178 of file equationofmotion.cpp.

15.10.3.58 std::complex< double > EquationofMotion::Func6 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 914 of file equationofmotion.cpp.

15.10.3.59 std::complex< double > EquationofMotion::Func7 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 920 of file equationofmotion.cpp.

15.10.3.60 std::complex< double > EquationofMotion::Func8 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 926 of file equationofmotion.cpp.

15.10.3.61 std::complex< double > EquationofMotion::Func9 ( ) [protected], [virtual]

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 932 of file equationofmotion.cpp.

15.10.3.62 int EquationofMotion::getDataIndex ( )

Gets the index for the equation of motion.

Gets the index for the equation of motion. The index is how the equation determines which numbers to access on the data. The following indices are used. Any higher indices can extend beyond this range, and the program easily adapts. But the following three are reserved. Unused indices are not transferred to the matrices when solved. So unused indices to not negatively impact calculation performance. However, using excessively large indices (say 500 when you only have 3 equations) will result in large matrices and unecessary memory requirements. THe following index reservations apply. 1: Translation in x-direction. Specific to rigid body motion. 2: Translation in y-direction. Specific to rigid body motion. 3: Translation in z-direction. Specific to rigid body motion. 4: Rotation about x-direction. Specific to rigid body motion. 5: Rotation about y-direction. Specific to rigid body motion. 6: Rotation about z-direction. Specific to rigid body motion.

Returns

Returns an integer number representing the data index used by the equation.

Definition at line 109 of file equationofmotion.cpp.

15.10.3.63 complex< double > EquationofMotion::Kronecker (int *var1*, int *var2*, bool *anti* = false) [protected]

The mathematical Kronecker delta function.

The mathematical Kronecker-delta function. Used to filter out terms when doing a double summation between two indices. The function evaluates to one when the two indices are equal, and evaluates to zero any other time. Multiplying a term by the Kronecker delta ensures that the results will be filtered to only have terms of equal indices. If this relates back to a matrix, the kronecked delta filters the deta to only include diagonal terms.

Parameters

var1	Integer variable. The first index that is being summed across.
var2	Integer variable. The second index that is being summed across.

anti	Boolean variable. Sometimes the researcher may be interested in the off diagonal terms. Cases when var1 does not equatl var2. In those cases, the Kronecker delta function should work in reverse and filter out the diagonal terms in a matrix. The anti variable is a trigger for the Kronecker delta function to work in reverse of its normal method. The default setting for this variable is false. By default, the Kronecker delta function evaluates with one when var1 = var2.

Returns

Complex number. Evaluates to either zero (0 + 0j), or one (1 + 0j).

Definition at line 192 of file equationofmotion.cpp.

15.10.3.64 int EquationofMotion::ord ( ) [protected]

Returns the index integer for iteration on order of derviative.

Returned variable is expressed in human numbering. But in this case, the order of 0 is a valid number. So numbering starts from zero (0).

Returns

Returns the index integer for iteration on order of derviative.

Definition at line 709 of file equationofmotion.cpp.

15.10.3.65 int & EquationofMotion::refDataIndex ( )

Gets the index for the equation of motion.

Gets the index for the equation of motion. The index is how the equation determines which numbers to access on the data. The following indices are used. Any higher indices can extend beyond this range, and the program easily adapts. But the following three are reserved. Unused indices are not transferred to the matrices when solved. So unused indices to not negatively impact calculation performance. However, using excessively large indices (say 500 when you only have 3 equations) will result in large matrices and unecessary memory requirements. THe following index reservations apply. 1: Translation in x-direction. Specific to rigid body motion. 2: Translation in y-direction. Specific to rigid body motion. 3: Translation in z-direction. Specific to rigid body motion. 4: Rotation about x-direction. Specific to rigid body motion. 5: Rotation about y-direction. Specific to rigid body motion. 6: Rotation about z-direction. Specific to rigid body motion.

Returns

Returns a reference to the protected data index variable contained in the class.

Definition at line 117 of file equationofmotion.cpp.

15.10.3.66 string & EquationofMotion::refDescription ( )

The description for the equation object.

The description for the equation object. This is an expanded version of the name. Again, purely for user identification of the EquationofMotion object. Brief names go under the Name property. More extensive descriptions go under this property. These would be useful to the user for describing the physical meaning behind the equation of motion.

Returns

Returns reference to the protected pDescription variable.

Definition at line 157 of file equationofmotion.cpp.

15.10.3.67 string & EquationofMotion::refName ( )

The name for the equation object.

The name for the equation object. This is the short name that user will use to identify the meaning of the equation.

Returns

Returns reference to the protected pName variable.

Definition at line 144 of file equationofmotion.cpp.

15.10.3.68 void EquationofMotion::setArguments (int *argn*, std::vector< double > *argv*)

Sets any values for arguments that may be used by the equation of motion.

Sets any values for arguments that may be used by the equation of motion. These can be any numerical value as needed by the equation of motion.

Parameters

argn	The number of arguments to expect.
argv	The vector containing the argument values.

Definition at line 136 of file equationofmotion.cpp.

15.10.3.69 void EquationofMotion::setDataIndex (int *DataIn*)

Sets the index for the equation of motion.

Sets the index for the equation of motion. The index is how the equation determines which numbers to access on the data. The following indices are used. Any higher indices can extend beyond this range, and the program easily adapts. But the following three are reserved. Unused indices are not transferred to the matrices when solved. So unused indices to not negatively impact calculation performance. However, using excessively large indices (say 500 when you only have 3 equations) will result in large matrices and unecessary memory requirements. THe following index reservations apply. 1: Translation in x-direction. Specific to rigid body motion. 2: Translation in y-direction. Specific to rigid body motion. 3: Translation in z-direction. Specific to rigid body motion. 4: Rotation about x-direction. Specific to rigid body motion. 5: Rotation about y-direction. Specific to rigid body motion. 6: Rotation about z-direction. Specific to rigid body motion.

Parameters

DataIn	The integer of the data index to use.

Definition at line 102 of file equationofmotion.cpp.

15.10.3.70 void EquationofMotion::setDebugData (double *freqIn*, std::complex< double > *solnIn*, bool *coeffIn* = `false`)

Sets debugging data to use when creating fictional inputs purely for debugging this function. Allows the programmer to debug the function independent of the other functions which depend on it.

Parameters

freqIn	Wave Frequency. Double. Variable passed by value.
solnIn	Solution of motion. Complex, double variable. Variable passed by value.
coeffIn	Boolean to describe if Ddt should calculate coefficients only. False by default.

Definition at line 170 of file equationofmotion.cpp.

15.10.3.71 void EquationofMotion::setDescription (std::string *descIn*)

The description for the equation object.

This is an expanded version of the name. Again, purely for user identification of the EquationofMotion object. Brief names go under the Name property. More extensive descriptions go under this property. These would be useful to the user for describing the physical meaning behind the equation of motion.

Parameters

descIn	String. The variable used to specify the description for the equation of motion. Variable passed by value.

Definition at line 164 of file equationofmotion.cpp.

15.10.3.72 complex< double > EquationofMotion::setFormula ( ) [protected], [virtual]

The formula used by the equation of motion.

The formula used by the equation of motion. The formula gets rewritten in a unique form. Rearrange any equations so that they have zero on the right hand size.

Example: If the formula were Ax + By = F, it must be rearranged to: Ax + By - F = 0

The formula can also make use of several math functions provided by the equation of motion object.

Reimplemented in osea::ofreq::EqnTranslation, and osea::ofreq::EqnRotation.

Definition at line 181 of file equationofmotion.cpp.

15.10.3.73 void EquationofMotion::setName (std::string *nameIn*)

The name for the equation object.

This is the short name that user will use to identify the meaning of the equation.

Parameters

nameIn	String. The variable which specifies the short name for the equation of motion. Variable passed by value.

Definition at line 151 of file equationofmotion.cpp.

15.10.3.74 std::complex< double > EquationofMotion::Sum (std::string *FuncName*, std::string *index*, int *from* = -1, int *to* = -1) [protected]

Sums across a variable.

Sums across a variable. The index limits can be specified. Or the keyword functions can be used to automatically Sum across the entire index range. This implementation accepts the name which specifies one of 50 available functions. The functions are not defined. The user must define the function and then specify the function name to use that function in the Sum function. Sum functions can be nested within other function definitions.

Parameters

FuncName	String which specifies the name of the function you wish to use as input to the summation. Example: Sum("Func1()", ...). The specified function name must be one of the available functions. ("Func1(), "Func2()", ... "Func50()") None of the functions can accept input parameters. But you can use the input parameters already defined within the class. Output for any function definition must always be data type of std::complex<double>.
index	std::string specifying which variable should be summed on. This may be any one of these options: Order of derivative = "ord" Variable = "var" Body = "bod"

from	Integer for the beginning value of the summation. Default value of negative one (-1) indicates that the summation will happen at the lowest value of the variable index specified.
to	Integer for the ending value of the summation. Default value of negative one (-1) indicates that the summation will happen at the highest value of the variable index specified.

Returns

Returns a complex value that is the summation of the index and limits specified.

Definition at line 750 of file equationofmotion.cpp.

15.10.3.75 std::complex< double > EquationofMotion::Sum (std::complex< double >(*)(void) *force*, int *from*, int *to*) [protected]

Sums a function multiple times.

The index limits can be specified. Or the keyword functions can be used to automatically Sum across the entire index range. This implementation accepts a function pointer with no parameters.

Parameters

force	Input to specify which items the results should Sum across. Typically, this is one of the built-in force functions. However, it can be any function, any item, any calculation. The only catch is that the input value must be a std::complex<double> data type. Input format is a function pointer. This allows the Sum function to update as it performs iterations. The only catch is that you can not combine multiple values into one. You must define a single function for each input argument you want.
from	Integer for the beginning value of the summation.
to	Integer for the ending value of the summation.

Returns

Returns a complex value that is the summation of the index and limits specified.

Definition at line 854 of file equationofmotion.cpp.

15.10.3.76 std::complex< double > EquationofMotion::Sum (std::complex< double > *force*, int *from*, int *to*) [protected]

Sums a constant value multiple times.

The index limits can be specified. Or the keyword functions can be used to automatically Sum across the entire index range.

Parameters

force	Input to specify constant value that Sum should add. The only catch is that the input value must be a std::complex<double> data type. Input format is a variabled passed by value.
from	Integer for the beginning value of the summation.
to	Integer for the ending value of the summation.

Returns

Returns a complex value that is the summation of the index and limits specified.

Definition at line 869 of file equationofmotion.cpp.

15.10.3.77 int EquationofMotion::var ( ) [protected]

Returns the index integer for iteration on variable.

Returned value is expressed in human numbering. So numbering starts from 1.

Returns

Returns the index integer for iteration on variable.

Definition at line 669 of file equationofmotion.cpp.

15.10.4 Member Data Documentation

15.10.4.1 int osea::ofreq::EquationofMotion::argcount [protected]

Used to supply arguments to the equation of motion. Recods the number of arguments.

Used to supply arguments to the equation of motion. Recods the number of arguments. Not required for use of the equation of motion object.

Definition at line 411 of file equationofmotion.h.

15.10.4.2 std::vector<double> osea::ofreq::EquationofMotion::argvalue [protected]

Used to supply arguments to the equation of motion. Uknown, arbitrary double precision values. A vector of uknown size.

Used to supply arguments to the equation of motion. Uknown, arbitrary double precision values. A vector of uknown size. Not required for use fo the equation of motion object.

Definition at line 421 of file equationofmotion.h.

15.10.4.3 unsigned int osea::ofreq::EquationofMotion::pBod [protected]

The integer of the current body. Used for iteration and summation functions.

Definition at line 557 of file equationofmotion.h.

15.10.4.4 unsigned int osea::ofreq::EquationofMotion::pCurOrd [protected]

The integer of the current value of ord() index. Used for iteration and summation functions.

Definition at line 551 of file equationofmotion.h.

15.10.4.5 unsigned int osea::ofreq::EquationofMotion::pCurVar [protected]

The integer of the current value of var() index. Used for iteration and summation functions.

Definition at line 545 of file equationofmotion.h.

15.10.4.6 std::string osea::ofreq::EquationofMotion::pDescription [protected]

The description for the equation object.

The description for the equation object. This is an expanded version of the name. Again, purely for user identification of the EquationofMotion object. Brief names go under the Name property. More extensive descriptions go under this property. These would be useful to the user for describing the physical meaning behind the equation of motion.

Definition at line 481 of file equationofmotion.h.

15.10.4.7 std::string osea::ofreq::EquationofMotion::pName [protected]

The name for the equation object.

The name for the equation object. This is the short name that user will use to identify the meaning of the equation.

Definition at line 470 of file equationofmotion.h.

15.10.4.8 int EquationofMotion::undefArg = -1 [static], [protected]

Integer value for undefined argument in the summation function.

Definition at line 561 of file equationofmotion.h.

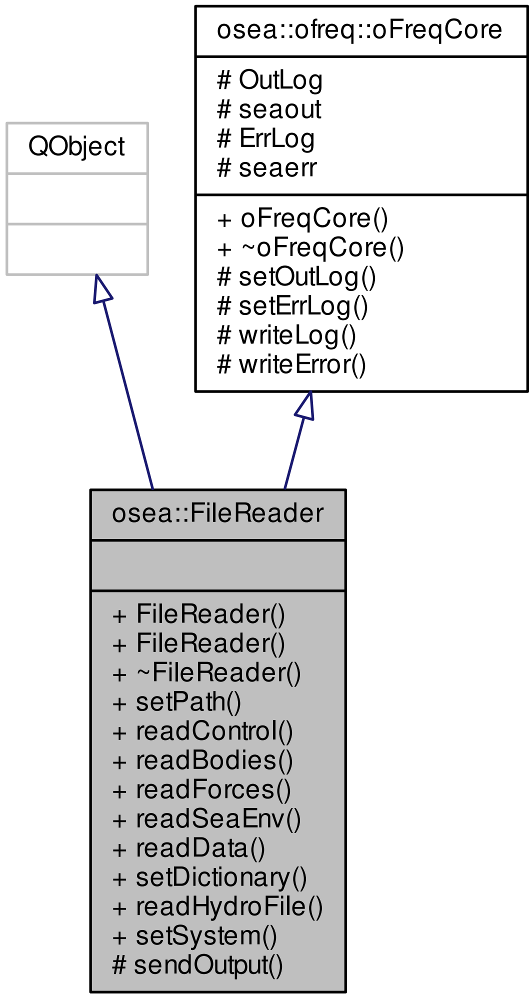
The documentation for this class was generated from the following files:

- bin/ofreq/motion_model/equationofmotion.h
- bin/ofreq/motion_model/equationofmotion.cpp

15.11 osea::FileReader Class Reference

```
#include <filereader.h>
```

Inheritance diagram for osea::FileReader:



Public Slots

- int readHydroFile (std::string path)

    Reads hydrodynamic input files.
- void setSystem (ofreq::System ∗ptInput)

    Sets the system object for the dictionary to reference.

Signals

- void outputControlFile (ObjectGroup output)

    Sends output of objects discovered when reading the Control file. Top level objects can include:
- void outputBodiesFile (ObjectGroup output)

    Sends output of objects discovered when reading the Bodies file. Top level objects can include:
- void outputDataFile (ObjectGroup output)

Sends output of objects discovered when reading the Data file. Top level objects can include:
- void outputForcesFile (ObjectGroup output)

  Sends output of objects discovered when reading the Forces file. Top level objects can include:
- void outputSeaEnvFile (ObjectGroup output)

  Sends output of objects discovered when reading the SeaEnv file. Top level objects can include:

Public Member Functions

- FileReader ()

  Default constructor.
- FileReader (std::string Path)

  Constructor with input for file path that holds input files.
- ∼FileReader ()
- void setPath (std::string input)

  Sets the path to the working directory that all control files are located in.
- int readControl ()

  Reads in the control file and parses its inputs.
- int readBodies ()

  Reads in the Bodies file and parses its inputs.
- int readForces ()

  Reads in the Forces file and parses its inputs.
- int readSeaEnv ()

  Reads in the Sea Environment input file and parses its inputs.
- int readData ()

  Reads in the Data input file and parses its inputs.
- void setDictionary (osea::Dictionary &dictIn)

  Sets the dictionary object to be used for processing any data read from the input files.

Protected Member Functions

- void sendOutput (int index)

  Sends the results of parsing the input file onto the dictionary object. Use for processing the input file.

Additional Inherited Members

15.11.1 Detailed Description

FileReader is the next generation of superseded class: ReadInputFile. FileReader simply reads the text file and parses it into keword value pairs. FileReader reads in the input files. It then passes those input files to the Parser object. Parser then segments the file in a series of object groupings with a vector list of keyword value pairs. The file reader interprets a limited number of those keywords to recognize new object declarations. It creates the new objects. It then parses the information in that object into a series of keyword-value pairs. Each pair is sent to a Dictionary object that interprets the information. Information is sent using Qt Slots and Signals.

To use this class, the following sequence must be followed. 1.) Create FileReader object. 2.) Create Dictionary objects for each file type you will read. 3.) Satisfy any follow on dependencies for Dictionary objects. 4.) Qt Connect FileReader to Dictionary objects. 5.) Qt Connect FileReader to System object. 6.) Feed in the target path to the FileReader object. 7.) Read each file type in turn. The FileReader object will automatically find the file with the correct filename, read it, parse it, and send the resulting keyword-value pairs to the appropriate Dictionary object.

Definition at line 105 of file filereader.h.

15.11.2 Constructor & Destructor Documentation

15.11.2.1 FileReader::FileReader ( )

Default constructor.

Definition at line 71 of file filereader.cpp.

15.11.2.2 osea::FileReader::FileReader (std::string *Path*)

Constructor with input for file path that holds input files.

Parameters

Path	The full path to the working directory. Variable passed by value.

15.11.2.3 FileReader::∼FileReader ( )

Default destructor. Nothing happens here.

Definition at line 84 of file filereader.cpp.

15.11.3 Member Function Documentation

15.11.3.1 void osea::FileReader::outputBodiesFile (ObjectGroup *output*) [signal]

Sends output of objects discovered when reading the Bodies file. Top level objects can include:

1. Body object.

   See Also

   Body

   Parameters

output	The ObjectGroup object parsed out of the file. Variable passed by value.

15.11.3.2 void osea::FileReader::outputControlFile (ObjectGroup *output*) [signal]

Sends output of objects discovered when reading the Control file. Top level objects can include:

1. System object.

   See Also

   System

   Parameters

output	The ObjectGroup object parsed out of the file. Variable passed by value.

15.11.3.3 void osea::FileReader::outputDataFile (ObjectGroup *output*) [signal]

Sends output of objects discovered when reading the Data file. Top level objects can include:

1. hydrofiles

Parameters

output	The ObjectGroup object parsed out of the file. Variable passed by value.

15.11.3.4 void osea::FileReader::outputForcesFile (ObjectGroup *output*) [signal]

Sends output of objects discovered when reading the Forces file. Top level objects can include:

1. ForceActive
2. ForceReactive
3. ForceCrossBody

See Also

ForceActive
ForceCross
ForceReact

Parameters

output	The ObjectGroup object parsed out of the file. Variable passed by value.

15.11.3.5 void osea::FileReader::outputSeaEnvFile (ObjectGroup *output*) [signal]

Sends output of objects discovered when reading the SeaEnv file. Top level objects can include:

1. Wave_Custom
2. Sea_Custom

Parameters

output	The ObjectGroup object parsed out of the file. Variable passed by value.

15.11.3.6 int FileReader::readBodies ( )

Reads in the Bodies file and parses its inputs.

Returns

Returns integer to report success or failure of file parsing. Returns 0 for success. Returns 1 for file does not exist.

Definition at line 124 of file filereader.cpp.

15.11.3.7 int FileReader::readControl ( )

Reads in the control file and parses its inputs.

Returns

Returns integer to report success or failure of file parsing. Returns 0 for success. Returns 1 for file does not exist.

Definition at line 104 of file filereader.cpp.

15.11.3.8 int FileReader::readData ( )

Reads in the Data input file and parses its inputs.

Returns

Returns integer to report success or failure of file parsing. Returns 0 for success. Returns 1 for file does not exist.

Definition at line 190 of file filereader.cpp.

15.11.3.9 int FileReader::readForces ( )

Reads in the Forces file and parses its inputs.

Returns

Returns integer to report success or failure of file parsing. Returns 0 for success. Returns 1 for file does not exist.

Definition at line 150 of file filereader.cpp.

15.11.3.10 int FileReader::readHydroFile (std::string *path*) [slot]

Reads hydrodynamic input files.

Parameters

path	The full path to the hydrodynamic input file to read.

Returns

Returns integer to report success or failure of file parsing. Returns 0 for success. Returns 1 for file does not exist.

Definition at line 222 of file filereader.cpp.

15.11.3.11 int FileReader::readSeaEnv ( )

Reads in the Sea Environment input file and parses its inputs.

Returns

Returns integer to report success or failure of file parsing. Returns 0 for success. Returns 1 for file does not exist.

Definition at line 170 of file filereader.cpp.

15.11.3.12 void FileReader::sendOutput (int *index*) [protected]

Sends the results of parsing the input file onto the dictionary object. Use for processing the input file.

Parameters

index	Integer. The index which specifies which object in the list of recognized objects to use. Variable passed by value.

Definition at line 238 of file filereader.cpp.

15.11.3.13 void FileReader::setDictionary (osea::Dictionary & *dictIn*)

Sets the dictionary object to be used for processing any data read from the input files.

Parameters

dictIn	The dictionary object that you want to use for processing the file. This can change between reading individual files. Variable is passed by reference and stored as a pointer in the object.

Definition at line 210 of file filereader.cpp.

15.11.3.14 void FileReader::setPath (std::string *input*)

Sets the path to the working directory that all control files are located in.

Parameters

input	The full path to the working directory. Do not include directory separator (SLASH, "/") at end of std::string. Variable passed by value.

Definition at line 90 of file filereader.cpp.

15.11.3.15 void FileReader::setSystem (ofreq::System ∗ *ptInput*) [slot]

Sets the system object for the dictionary to reference.

Parameters

ptSystem	Pointer to the System object. Variable passed by value.

Definition at line 229 of file filereader.cpp.

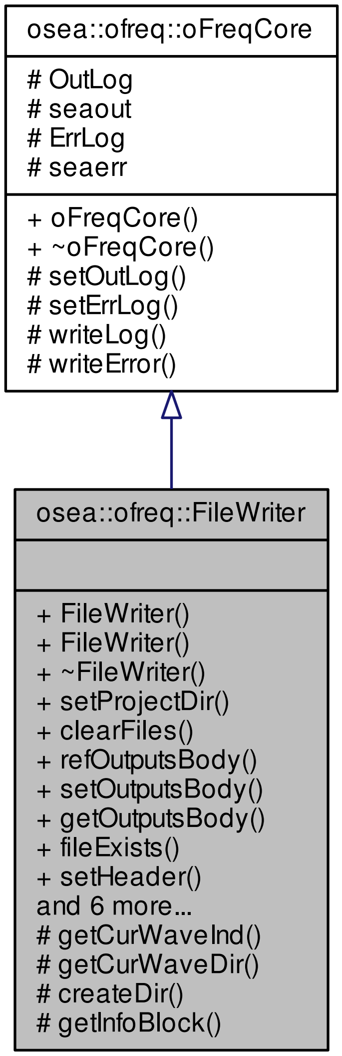
The documentation for this class was generated from the following files:

- bin/ofreq/file_reader/filereader.h
- bin/ofreq/file_reader/filereader.cpp

15.12 osea::ofreq::FileWriter Class Reference

`#include <filewriter.h>`

Inheritance diagram for osea::ofreq::FileWriter:



Public Member Functions

- FileWriter ()

  The default constructor.
- FileWriter (std::string rootPath, OutputsBody &BodyIn)

  Constructor that includes the two important properties that must be declared for FileWriter to work correctly. The root path for the project must be declared. And the pointer to the OutputsBody object must be declared.
- ∼FileWriter ()
- void setProjectDir (std::string dirIn)

  Sets the path to the project directory. Assumes the std::string specifying the path does not end with a slash mark. The class will automatically add the slash mark. If a slash mark is present, the function will automatically remove it.
- bool clearFiles ()
- OutputsBody & refOutputsBody ()

Provides direct access to the OutputsBody object. The OutputsBody object must be set for the fileWriter to work. All file data comes from the FileWriter object.

- void setOutputsBody (OutputsBody ∗input)

  Sets the outputs body object, pointer input.
- OutputsBody ∗ getOutputsBody ()

  Returns a pointer to the Outputs Body object.
- bool fileExists (std::string filename)

  Test is a file exists. Function automatically assumes that the file is located in the directory associated with the getCurWaveInd() function. Returns true if the file exists and is valid. Returns false if the file does not exist or the directory does not exist.
- void setHeader (std::string filePathIn)

  Reads in from input file the header to be used in all files. This is a basic header text that should be at the top of all OpenSEA output files. Simple identification of the program. Nothing specific for output.
- int writeWaveDirection ()

  Writes the directions list to file.
- int writeFrequency ()
- int writeGlobalMotion ()

  Writes the output file of global motions. If file exists, appends to the file, assuming the appended file is a new body object.
- int writeGlobalVelocity ()

  Writes the output file of global velocities. If file exists, appends to the file, assuming the appended file is a new body object.
- int writeGlobalAcceleration ()

  Writes the output file of global accelerations. If file exists, appends to the file, assuming the appended file is a new body object.
- int writeGlobalSolution ()

  Writes the output file of global solutions. If file exists, appends to the file, assuming the appended file is a new body object.

Protected Member Functions

- int getCurWaveInd ()

  Gets the index of the current wave direction. The index is used to specify which directory to write the file into.
- std::string getCurWaveDir ()

  Returns the std::string containing the folder path for the current wave direction. Path name includes the slash mark. For example, if using wave index 0, the std::string output would be: "d0/".
- bool createDir (std::string path)

  Creates the directory specified by the std::string path. Assumes any specified directory is under the root project directory.
- std::string getInfoBlock (std::string nameIn)

Additional Inherited Members

15.12.1 Detailed Description

This class write all outputs to files. All output data is based on the attached OutputsBody object. To use the filewriter object, follow this sequence of steps: 1.) Create object. 2.) Set OutputsBody object associated with the file. 3.) Set the filesystem path to the root directory for the current run of ofreq. 4.) Run the clearFiles() function, which will clear out any pre-existing files. 5.) Run the writeFile function for the specified file that you want to write out.

Note that the OutputsBody object also provides the information on the current wave direction. And the FileWriter changes its directory to write to depending on the current wave direction. So the OutputsBody object must be updated before writing a new wave direction.

Definition at line 103 of file filewriter.h.

15.12.2 Constructor & Destructor Documentation

15.12.2.1 FileWriter::FileWriter ( )

The default constructor.

Definition at line 89 of file filewriter.cpp.

15.12.2.2 osea::ofreq::FileWriter::FileWriter (std::string *rootPath*, OutputsBody & *BodyIn*)

Constructor that includes the two important properties that must be declared for FileWriter to work correctly. The root path for the project must be declared. And the pointer to the OutputsBody object must be declared.

Parameters

rootPath	The full fule system path to the root directory of the currently running oFreq project. Not the path to the oFreq executable files. This is the path to the directory containing input and output files.
BodyIn	The OutputsBody object that will be used to write out data for the FileWriter. The OutputsBody object supplies the data, and the FileWriter writes that data to the ASCII text file. The Outputs-Body object also provides the information on the current wave direction. So the OutputsBody object must be updated before writing a new wave direction.

See Also

OutputsBody

15.12.2.3 FileWriter::∼FileWriter ( )

The default destructor, nothing happens here.

Definition at line 104 of file filewriter.cpp.

15.12.3 Member Function Documentation

15.12.3.1 bool FileWriter::clearFiles ( )

Remove all old directiories & files written by oFreq previous run.

Returns

Return true if all files & directories were successfully deleted.

Definition at line 124 of file filewriter.cpp.

15.12.3.2 bool FileWriter::createDir (std::string *path*) [protected]

Creates the directory specified by the std::string path. Assumes any specified directory is under the root project directory.

Parameters

path	std::string. The path of the directory to create.

Returns

Returns true if creation sucessful.

Definition at line 858 of file filewriter.cpp.

15.12.3.3 bool FileWriter::fileExists (std::string *filename*)

Test is a file exists. Function automatically assumes that the file is located in the directory associated with the getCurWaveInd() function. Returns true if the file exists and is valid. Returns false if the file does not exist or the directory does not exist.

Parameters

filename	std::string. Specifies the filename to search for. Only needs to specify local filename. Directory information is already inferred from previous settings with the OutputsBody object.

Returns

Returns boolean variable. True if the file exists. False if the file or any required directories do not exist.

See Also

FileWriter::getCurWaveInd()

Definition at line 173 of file filewriter.cpp.

15.12.3.4 string FileWriter::getCurWaveDir ( ) [protected]

Returns the std::string containing the folder path for the current wave direction. Path name includes the slash mark. For example, if using wave index 0, the std::string output would be: "d0/".

Returns

std::string output. Has the path name for the current wave directory.

Definition at line 848 of file filewriter.cpp.

15.12.3.5 int FileWriter::getCurWaveInd ( ) [protected]

Gets the index of the current wave direction. The index is used to specify which directory to write the file into.

Returns

Returns integer which specifies the index of the current wave direction. Index specifies the wave direction in the list of wave directions. Valid values are any integer from 0 or greater.

Definition at line 842 of file filewriter.cpp.

15.12.3.6 string FileWriter::getInfoBlock (std::string *nameIn*) [protected]

Set information about the file to be written after header and above data, included in the seafile block.

Parameters

nameIn	The name of the object.

Returns

Returns std::string. std::string contains the file info for the output file. Everything written into the seafile block. Variable passed by value.

Definition at line 873 of file filewriter.cpp.

15.12.3.7 OutputsBody ∗ FileWriter::getOutputsBody ( )

Returns a pointer to the Outputs Body object.

Returns

Returns a pointer to the Outputs Body object. Returned pointer passed by value.

Definition at line 167 of file filewriter.cpp.

15.12.3.8 OutputsBody & FileWriter::refOutputsBody ( )

Provides direct access to the OutputsBody object. The OutputsBody object must be set for the fileWriter to work. All file data comes from the FileWriter object.

Returns

Returns reference to the OutputsBody object. Variable passed by reference.

Definition at line 155 of file filewriter.cpp.

15.12.3.9 void FileWriter::setHeader (std::string *filePathIn*)

Reads in from input file the header to be used in all files. This is a basic header text that should be at the top of all OpenSEA output files. Simple identification of the program. Nothing specific for output.

Parameters

filePathIn	String variable specifying the full location of the folder which has the text for the header file. Header file must be a simple ASCII text file.

Definition at line 193 of file filewriter.cpp.

15.12.3.10 void FileWriter::setOutputsBody (OutputsBody ∗ *input*)

Sets the outputs body object, pointer input.

Takes a pointer to the Outputs body object as input and stores that pointer for future use.

Parameters

input	Pointer to the Outputs Body object. Pointer variable passed by value.

Definition at line 161 of file filewriter.cpp.

15.12.3.11 void FileWriter::setProjectDir (std::string *dirIn*)

Sets the path to the project directory. Assumes the std::string specifying the path does not end with a slash mark. The class will automatically add the slash mark. If a slash mark is present, the function will automatically remove it.

Parameters

dirIn	std::string specifying the path to the project directory. Variable passed by value.

Definition at line 110 of file filewriter.cpp.

15.12.3.12 int FileWriter::writeFrequency ( )

Writes the frequencies list to file.

Returns

Integer reports status of file writing. Returns of succesful. Otherwise returns a non-zero value that is the error code. Returned variable passed by value.

Definition at line 286 of file filewriter.cpp.

15.12.3.13 int FileWriter::writeGlobalAcceleration ( )

Writes the output file of global accelerations. If file exists, appends to the file, assuming the appended file is a new body object.

Returns

Integer reports status of file writing. Returns of succesful. Otherwise returns a non-zero value that is the error code. Returned variable passed by value.

Definition at line 589 of file filewriter.cpp.

15.12.3.14 int FileWriter::writeGlobalMotion ( )

Writes the output file of global motions. If file exists, appends to the file, assuming the appended file is a new body object.

Returns

Integer reports status of file writing. Returns of succesful. Otherwise returns a non-zero value that is the error code. Returned variable passed by value.

Definition at line 336 of file filewriter.cpp.

15.12.3.15 int FileWriter::writeGlobalSolution ( )

Writes the output file of global solutions. If file exists, appends to the file, assuming the appended file is a new body object.

Returns

Integer reports status of file writing. Returns of succesful. Otherwise returns a non-zero value that is the error code. Returned variable passed by value.

Definition at line 714 of file filewriter.cpp.

15.12.3.16 int FileWriter::writeGlobalVelocity ( )

Writes the output file of global velocities. If file exists, appends to the file, assuming the appended file is a new body object.

Returns

Integer reports status of file writing. Returns of succesful. Otherwise returns a non-zero value that is the error code. Returned variable passed by value.

Definition at line 464 of file filewriter.cpp.

15.12.3.17 int FileWriter::writeWaveDirection ( )

Writes the directions list to file.

Returns

Integer reports status of file writing. Returns of succesful. Otherwise returns a non-zero value that is the error code. Returned variable passed by value.

Definition at line 236 of file filewriter.cpp.

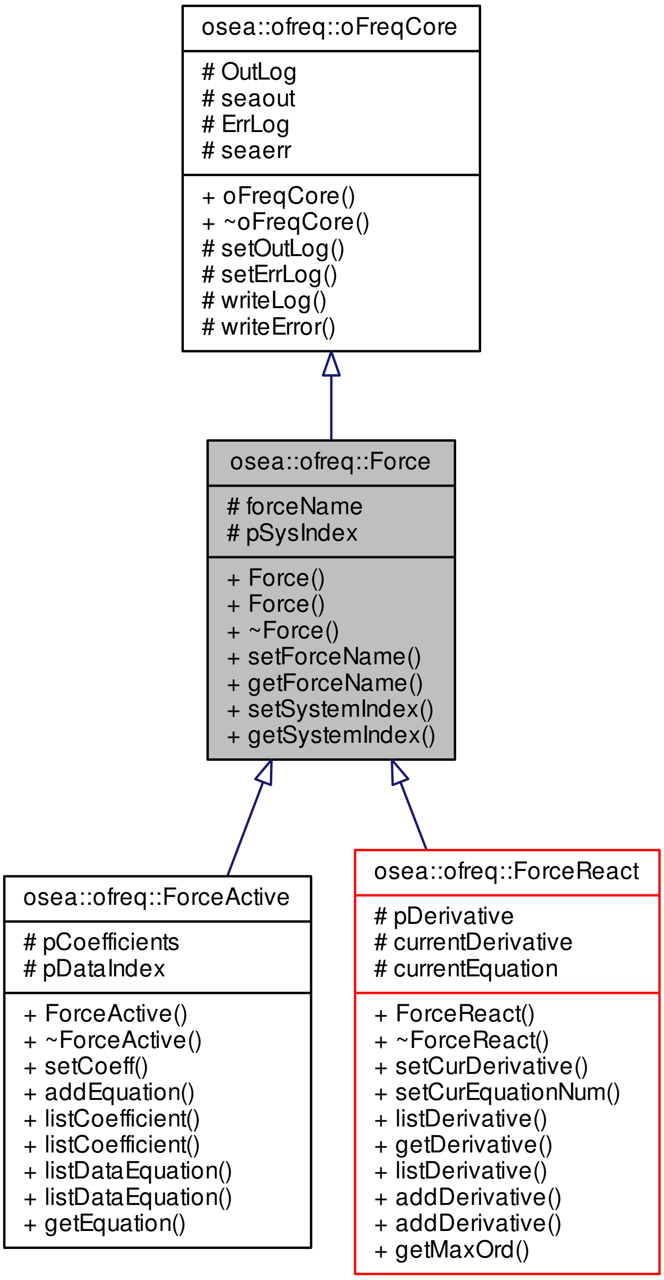
The documentation for this class was generated from the following files:

- bin/ofreq/file_writer/filewriter.h
- bin/ofreq/file_writer/filewriter.cpp

15.13 osea::ofreq::Force Class Reference

```
#include <force.h>
```

Inheritance diagram for osea::ofreq::Force:



Public Member Functions

- Force ()
- Force (int indexIn)

  Overloaded constructor for Force object. Allows the System object to pass the index of the newly created Force object in the list of other similar Force objects. Used during object creation for proper association with Body object.
- virtual ~Force ()

- void setForceName (std::string)
- std::string getForceName ()
- void setSystemIndex (int indexIn)

  Sets the system index. The index of the Force object as it exists in the list of other similar Force objects contained under the System object.
- int getSystemIndex ()

  Gets the system index. The index of the Force object as it exists in the list of other similar Force objects contained under the System object.

Protected Attributes

- std::string forceName
- int pSysIndex

  The index of the Force object as it exists in the list of other similar Force objects contained under the System object.

Additional Inherited Members

15.13.1 Detailed Description

This (base) class holds data for a force object.

Definition at line 83 of file force.h.

15.13.2 Constructor & Destructor Documentation

15.13.2.1 Force::Force ( )

The default constructor.

Definition at line 33 of file force.cpp.

15.13.2.2 Force::Force (int *indexIn*)

Overloaded constructor for Force object. Allows the System object to pass the index of the newly created Force object in the list of other similar Force objects. Used during object creation for proper association with Body object.

Parameters

indexIn	Integer. The index of the Force object as it exists in the list of other similar Force objects contained under the System object. Variable passed by value.

See Also

dictBodies
System
Body

Definition at line 38 of file force.cpp.

15.13.2.3 Force::∼Force ( ) [virtual]

The default destructor, nothing happens here.

Definition at line 45 of file force.cpp.

15.13.3 Member Function Documentation

15.13.3.1 string Force::getForceName ( )

Retrieve the name of the force.

Returns

newName The name of the force.

Definition at line 56 of file force.cpp.

15.13.3.2 int Force::getSystemIndex ( )

Gets the system index. The index of the Force object as it exists in the list of other similar Force objects contained under the System object.

Returns

Integer. The index of the Force object as it exists in the list of other similar Force objects contained under the System object. Variable passed by value.

Definition at line 68 of file force.cpp.

15.13.3.3 void Force::setForceName (std::string)

Sets the name of the force.

Parameters

newName	The name of the force.

Definition at line 50 of file force.cpp.

15.13.3.4 void Force::setSystemIndex (int *indexIn*)

Sets the system index. The index of the Force object as it exists in the list of other similar Force objects contained under the System object.

Parameters

indexIn	Integer. The index of the Force object as it exists in the list of other similar Force objects contained under the System object. Variable passed by value.

Definition at line 62 of file force.cpp.

15.13.4 Member Data Documentation

15.13.4.1 std::string osea::ofreq::Force::forceName [protected]

The force name.

Definition at line 141 of file force.h.

15.13.4.2 int osea::ofreq::Force::pSysIndex [protected]

The index of the Force object as it exists in the list of other similar Force objects contained under the System object.

Definition at line 148 of file force.h.

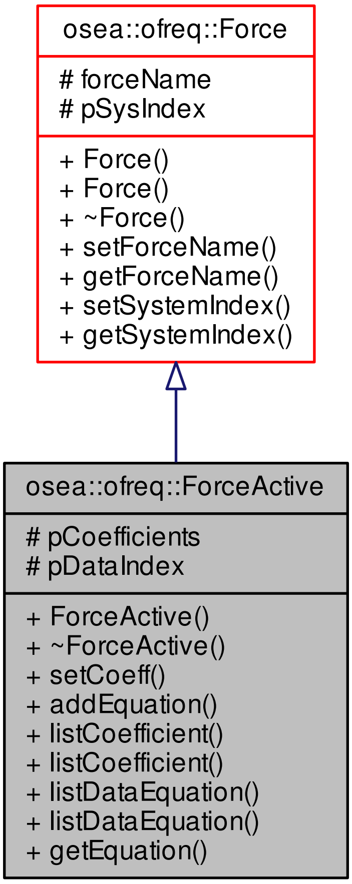
The documentation for this class was generated from the following files:

- bin/ofreq/global_objects/force.h
- bin/ofreq/global_objects/force.cpp

15.14 osea::ofreq::ForceActive Class Reference

`#include <forceactive.h>`

Inheritance diagram for osea::ofreq::ForceActive:



Public Member Functions

- ForceActive ()
- virtual ∼ForceActive ()

- void setCoeff (std::complex< double > coeffIn, unsigned int index=-1)

  Sets the coefficient for the specified index.
- void addEquation (std::complex< double > eqIn, int dataIndex=-1)

  Adds an equation to the list of equations / coefficients stored in the forceActive object.
- std::vector< complexDouble > & listCoefficient ()
- complexDouble & listCoefficient (unsigned int index)

  Provides direct access to a coefficient from the list of coefficients.
- std::vector< complexDouble > & listDataEquation ()

  Another implementation of function listCoefficient. But this one uses the Data index.
- complexDouble & listDataEquation (int index)

  Another implementation of function listCoefficient(index).
- complexDouble getEquation (int number)

  Get a specific number from the list of coefficients.

Protected Attributes

- std::vector< complexDouble > pCoefficients
- std::vector< int > pDataIndex

  The list of data indices for each coefficient. The listDataEquation() method will lookup items by their data index, instead of their occurance index in the container vector. This vector contains those data indices used for that search by dataIndex.

Additional Inherited Members

15.14.1 Detailed Description

This class holds data for an active force.

Definition at line 86 of file forceactive.h.

15.14.2 Constructor & Destructor Documentation

15.14.2.1 ForceActive::ForceActive ( )

The default constructor.

Definition at line 36 of file forceactive.cpp.

15.14.2.2 ForceActive::∼ForceActive ( ) [virtual]

The default destructor, nothing happens here.

Definition at line 41 of file forceactive.cpp.

15.14.3 Member Function Documentation

15.14.3.1 void ForceActive::addEquation (std::complex< double > *eqIn,* int *dataIndex* = -1)

Adds an equation to the list of equations / coefficients stored in the forceActive object.

Expands the current list of equations / coefficients by adding the inputs to the end of the list. Adds both the coefficient specified, and the corresponding equation data index.

Parameters

eqIn	Complex double input. This is the coefficient value stored for the specified equation. Variable passed by value.
dataIndex	Integer input. This is the data index specified. When objects are retrieved by their data index, they will lookup this value.

Definition at line 60 of file forceactive.cpp.

15.14.3.2 complexDouble ForceActive::getEquation (int *number*)

Get a specific number from the list of coefficients.

Get a specific number from the list of coefficients. Similar to getCoefficients(), only instead of returning the entire vector of coefficients, this only returns a single value in the list.

Parameters

number	Integer specifying which number should be retrieved from the list.

Returns

Complex Double which is the input coefficient for the active force on the equation specified by number. Returns by value, not by reference.

Definition at line 116 of file forceactive.cpp.

15.14.3.3 vector< complexDouble > & ForceActive::listCoefficient ( )

Retrieve the list of coefficients.

Returns

The list of coefficients.

Definition at line 77 of file forceactive.cpp.

15.14.3.4 complexDouble & ForceActive::listCoefficient (unsigned int *index*)

Provides direct access to a coefficient from the list of coefficients.

Returns a value from the list of coefficents. Which value to return is specified by the input index.

Parameters

index	Unsigned integer. Specifies which value to return from the list of coefficients.

Returns

Returns a complex double. Returned variable is a value from the list of coefficients. Returned variable is passed by reference.

Definition at line 83 of file forceactive.cpp.

15.14.3.5 vector< complexDouble > & ForceActive::listDataEquation ( )

Another implementation of function listCoefficient. But this one uses the Data index.

Returns

Vector containing the list of coefficients. Argument passed by reference.

Definition at line 89 of file forceactive.cpp.

15.14.3.6 complexDouble & ForceActive::listDataEquation (int *index*)

Another implementation of function listCoefficient(index).

Provides direct access to items in the list of equations. Returns a single variable from the list of coefficients. Variable is accessed by data index, and not by vector occurrence index.

Parameters

index	Integer. Specifies which value to return from the list of coefficients.

Returns

Returns a complex double. Returned variable is a value from the list of coefficients. Returned variable is passed by reference.

Definition at line 95 of file forceactive.cpp.

15.14.3.7 void ForceActive::setCoeff (std::complex< double > *coeffIn*, unsigned int *index* = -1)

Sets the coefficient for the specified index.

Parameters

coeffIn	The value of the coefficient to specify. Added as a complex number. Variable passed by value.
index	The equation index of the coefficient to specify.

Definition at line 46 of file forceactive.cpp.

15.14.4 Member Data Documentation

15.14.4.1 std::vector<complexDouble> osea::ofreq::ForceActive::pCoefficients [protected]

The list of force coeffients.

Definition at line 169 of file forceactive.h.

15.14.4.2 std::vector<int> osea::ofreq::ForceActive::pDataIndex [protected]

The list of data indices for each coefficient. The listDataEquation() method will lookup items by their data index, instead of their occurance index in the container vector. This vector contains those data indices used for that search by dataIndex.

Definition at line 177 of file forceactive.h.

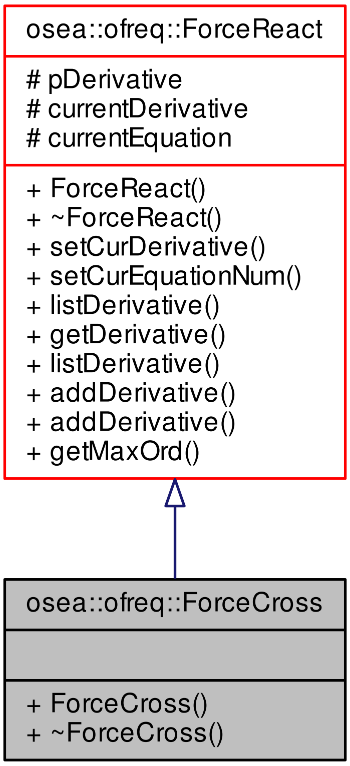
The documentation for this class was generated from the following files:

- bin/ofreq/global_objects/forceactive.h
- bin/ofreq/global_objects/forceactive.cpp

15.15 osea::ofreq::ForceCross Class Reference

```
#include <forcecross.h>
```

Inheritance diagram for osea::ofreq::ForceCross:



Public Member Functions

- ForceCross ()
- ∼ForceCross ()

Additional Inherited Members

15.15.1 Detailed Description

This class holds data for a cross body force. A cross body force is very closely related to a reactive force object and they behave almost exactly the same. The only difference is that the force within a reactive force object which is owned by Body A, they are dependant on the motions of that Body. But for a cross-body force object: The forces from the Cross-body force owned by Body A are dependant on the motions of another body.

Definition at line 85 of file forcecross.h.

15.15.2 Constructor & Destructor Documentation

15.15.2.1 ForceCross::ForceCross ( )

This default constructor creates a Body object.

Definition at line 32 of file forcecross.cpp.

15.15.2.2 ForceCross::∼ForceCross ( )

The default destructor, nothing happens here.

Definition at line 37 of file forcecross.cpp.

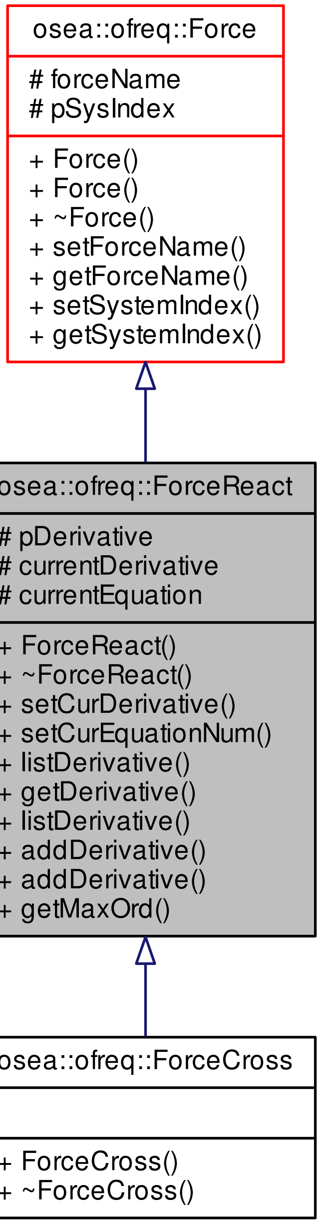
The documentation for this class was generated from the following files:

- bin/ofreq/global_objects/forcecross.h
- bin/ofreq/global_objects/forcecross.cpp

15.16 osea::ofreq::ForceReact Class Reference

```
#include <forcereact.h>
```

Inheritance diagram for osea::ofreq::ForceReact:



Public Member Functions

- ForceReact ()
- ∼ForceReact ()
- void setCurDerivative (int)
- void setCurEquationNum (int)
- std::vector< Derivative > & listDerivative ()

  Another implementation of getDerivatives, under a different name.

- Derivative getDerivative (unsigned int num)

  Retrieve the derivative object specified by the index number.
- Derivative & listDerivative (unsigned int num)

  Retrieve the derivative object specified by the index number.
- void addDerivative ()

  Adds a Derivative object to the list of derivatives. Creates a blank derivative object. Assumed to be the latest order of derivative in the list.
- void addDerivative (Derivative derivIn, unsigned int ordIn)

  Adds a Derivative object in the list of derivatives. Sets the new objects as the input for the new derivative object. Uses the properties of the Derivative object to set the correct index.
- int getMaxOrd ()

  Returns the maximum order of the derivatives included in the force object.

Protected Attributes

- std::vector< Derivative > pDerivative
- int currentDerivative
- int currentEquation

Additional Inherited Members

15.16.1 Detailed Description

This class holds all of the data for a reactive force.

Definition at line 83 of file forcereact.h.

15.16.2 Constructor & Destructor Documentation

15.16.2.1 ForceReact::ForceReact ( )

This default constructor.

Definition at line 33 of file forcereact.cpp.

15.16.2.2 ForceReact::∼ForceReact ( )

The default destructor, nothing happens here.

Definition at line 37 of file forcereact.cpp.

15.16.3 Member Function Documentation

15.16.3.1 void ForceReact::addDerivative ( )

Adds a Derivative object to the list of derivatives. Creates a blank derivative object. Assumed to be the latest order of derivative in the list.

Definition at line 92 of file forcereact.cpp.

15.16.3.2 void ForceReact::addDerivative (Derivative *derivIn*, unsigned int *ordIn*)

Adds a Derivative object in the list of derivatives. Sets the new objects as the input for the new derivative object. Uses the properties of the Derivative object to set the correct index.

Parameters

derivIn	The derivative object to add to the list of derivatives.
ordIn	The order of the derivative.

Definition at line 99 of file forcereact.cpp.

15.16.3.3 Derivative ForceReact::getDerivative (unsigned int *num*)

Retrieve the derivative object specified by the index number.

Retrieve the derivative object specified by the index number.

Parameters

num	The index number of the derivative object.

Returns

Returns the derivative object specified by integer num. Returned value is by value.

Definition at line 59 of file forcereact.cpp.

15.16.3.4 int ForceReact::getMaxOrd ( )

Returns the maximum order of the derivatives included in the force object.

Returns the maximum order of the derivatives included in the force object.

Returns

Returns integer. Returns the maximum order of the derivatives included in the force object. Returned result passed by value.

Definition at line 115 of file forcereact.cpp.

15.16.3.5 vector< Derivative > & ForceReact::listDerivative ( )

Another implementation of getDerivatives, under a different name.

Returns

Returns the vector of derviative objects. Returned object is by reference.

Definition at line 53 of file forcereact.cpp.

15.16.3.6 Derivative & ForceReact::listDerivative (unsigned int *num*)

Retrieve the derivative object specified by the index number.

Retrieve the derivative object specified by the index number. Retrieves a pointer to the derivative object.

Parameters

num	The index number of the derivative object.

Returns

Returns a pointer to the derivative object specified by integer num. Returned value is by reference.

Definition at line 75 of file forcereact.cpp.

15.16.3.7 void ForceReact::setCurDerivative (int *newOrder*)

Sets the current derivative.

Parameters

neworder	The order of derivative.

Definition at line 41 of file forcereact.cpp.

15.16.3.8 void ForceReact::setCurEquationNum (int *newEquationNum*)

Sets the current number of the equation.

Parameters

newEquation-Num	The number of the equation.

Definition at line 47 of file forcereact.cpp.

15.16.4 Member Data Documentation

15.16.4.1 int osea::ofreq::ForceReact::currentDerivative [protected]

The current order derivative.

Definition at line 165 of file forcereact.h.

15.16.4.2 int osea::ofreq::ForceReact::currentEquation [protected]

This current equation number.

Definition at line 166 of file forcereact.h.

15.16.4.3 std::vector<Derivative> osea::ofreq::ForceReact::pDerivative [protected]

This list of derivatives.

Definition at line 164 of file forcereact.h.

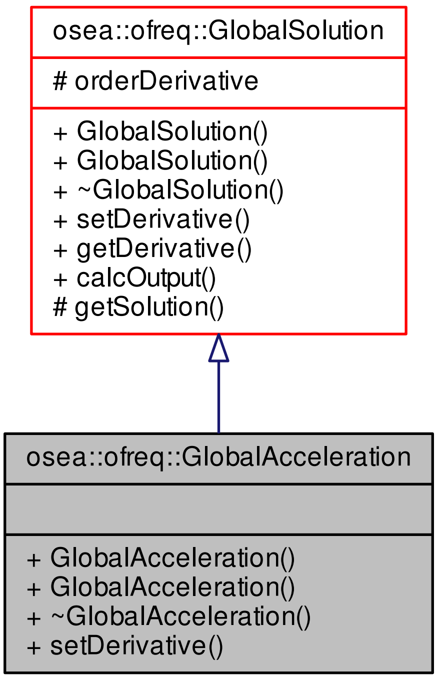
The documentation for this class was generated from the following files:

- bin/ofreq/global_objects/forcereact.h
- bin/ofreq/global_objects/forcereact.cpp

15.17 osea::ofreq::GlobalAcceleration Class Reference

`#include <globalacceleration.h>`

Inheritance diagram for osea::ofreq::GlobalAcceleration:



Public Member Functions

- GlobalAcceleration ()
- GlobalAcceleration (OutputsBody ∗input)

  Constructor that also sets the pointer to the OutputsBody object which contains the OutputDerived object.
- ∼GlobalAcceleration ()
- void setDerivative (int ord)

  Sets the order of the derivative for the object.

Additional Inherited Members

15.17.1 Detailed Description

This class represents the Global Acceleraion Solution.

Definition at line 86 of file globalacceleration.h.

15.17.2 Constructor & Destructor Documentation

15.17.2.1 GlobalAcceleration::GlobalAcceleration ( )

This default constructor creates a Global Acceleration object.

Definition at line 42 of file globalacceleration.cpp.

15.17.2.2 GlobalAcceleration::GlobalAcceleration (OutputsBody ∗ *input*)

Constructor that also sets the pointer to the OutputsBody object which contains the OutputDerived object.

Parameters

input	Pointer to the OutputsBody objec that contains this OutputDerived object. Pointer passed by value.

See Also

setOutputsBody()

Definition at line 51 of file globalacceleration.cpp.

15.17.2.3 GlobalAcceleration::∼GlobalAcceleration ( )

The default destructor, nothing happens here.

Definition at line 60 of file globalacceleration.cpp.

15.17.3 Member Function Documentation

15.17.3.1 void GlobalAcceleration::setDerivative (int *ord*) [virtual]

Sets the order of the derivative for the object.

Parameters

ord	Integer input that specifies the order of the derivative. Value can be anything from 0 or larger. Variable is passed by value.

Reimplemented from osea::ofreq::GlobalSolution.

Definition at line 65 of file globalacceleration.cpp.

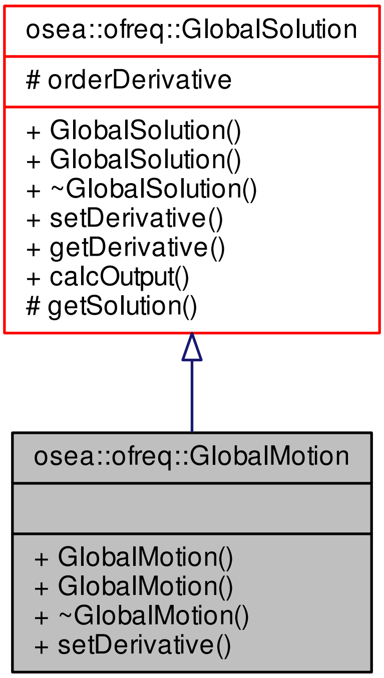
The documentation for this class was generated from the following files:

- bin/ofreq/derived_outputs/globalacceleration.h
- bin/ofreq/derived_outputs/globalacceleration.cpp

15.18 osea::ofreq::GlobalMotion Class Reference

```
#include <globalmotion.h>
```

Inheritance diagram for osea::ofreq::GlobalMotion:



Public Member Functions

- GlobalMotion ()
- GlobalMotion (OutputsBody ∗input)

  Constructor that also sets the pointer to the OutputsBody object which contains the OutputDerived object.
- ∼GlobalMotion ()
- void setDerivative (int ord)

  Sets the order of the derivative for the object.

Additional Inherited Members

15.18.1 Detailed Description

This class represents the Global Motion Solution.

Definition at line 84 of file globalmotion.h.

15.18.2 Constructor & Destructor Documentation

15.18.2.1 GlobalMotion::GlobalMotion ( )

This default constructor creates a Global Motion object.

Definition at line 41 of file globalmotion.cpp.

15.18.2.2 GlobalMotion::GlobalMotion (OutputsBody * *input*)

Constructor that also sets the pointer to the OutputsBody object which contains the OutputDerived object.

Parameters

input	Pointer to the OutputsBody objec that contains this OutputDerived object. Pointer passed by value.

See Also

setOutputsBody()

Definition at line 50 of file globalmotion.cpp.

15.18.2.3 GlobalMotion::∼GlobalMotion ( )

The default destructor, nothing happens here.

Definition at line 59 of file globalmotion.cpp.

15.18.3 Member Function Documentation

15.18.3.1 void GlobalMotion::setDerivative (int *ord*) [virtual]

Sets the order of the derivative for the object.

Parameters

ord	Integer input that specifies the order of the derivative. Value can be anything from 0 or larger. Variable is passed by value.

Reimplemented from osea::ofreq::GlobalSolution.

Definition at line 64 of file globalmotion.cpp.

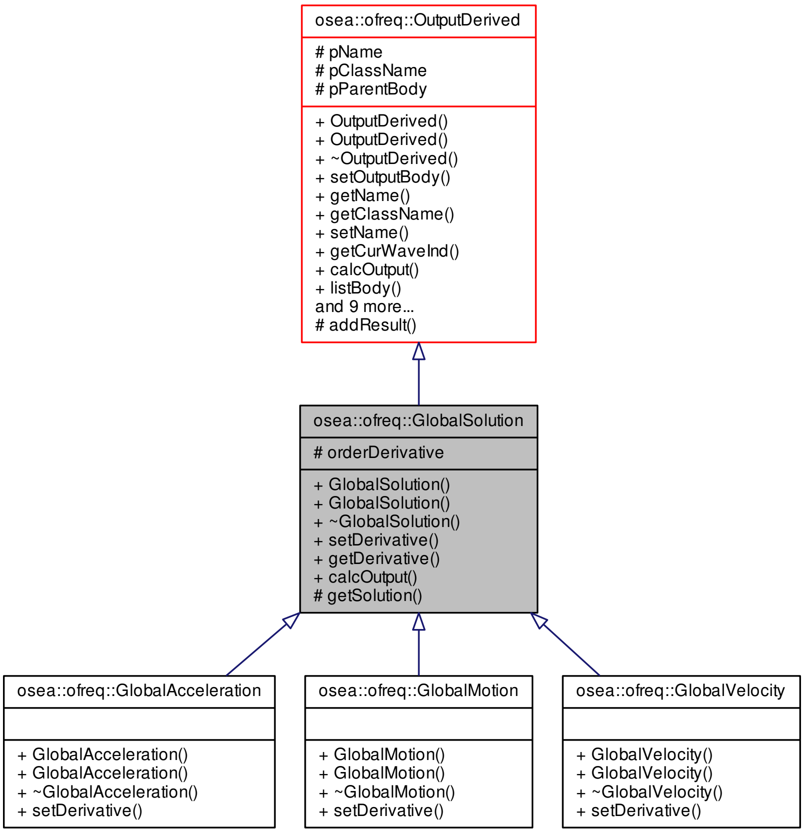
The documentation for this class was generated from the following files:

- bin/ofreq/derived_outputs/globalmotion.h
- bin/ofreq/derived_outputs/globalmotion.cpp

15.19 osea::ofreq::GlobalSolution Class Reference

`#include <globalsolution.h>`

Inheritance diagram for osea::ofreq::GlobalSolution:



Public Member Functions

- GlobalSolution ()
- GlobalSolution (OutputsBody ∗input)

    Constructor that also sets the pointer to the OutputsBody object which contains the OutputDerived object.

- ∼GlobalSolution ()
- virtual void setDerivative (int ord)

    Sets the order of the derivative for the object.

- int getDerivative ()

    Gets the order of derivative set for the object.

- int calcOutput (int freqIn)

    Calculates the GlobalSolution output. GlobalSolution is the direct output of the solution for each body, translated into body coordinates. The output is modified to provide the specified order of derivative for the solution.

Protected Member Functions

- Solution & getSolution (double freqIn)

    Gets the solution matrix to perform operations on. Accesses the solution matrix from the parent body.

Protected Attributes

- int orderDerivative

  The order of derivative to use for calculating output. Set by child classes for specifying specific outputs.

Additional Inherited Members

15.19.1 Detailed Description

This class represents the Global Solution. The Global Solution is the direct output of the solved values for each body. It provides the motions calculated for each body, translated back to body coordinate system. The GlobalSolution can output any desired derivative of the solved motions. Several other child classes are derived from this class. The only difference is that for those other classes, the order of derivative is predefined.

See Also

GlobalMotion
GlobalVelocity
GlobalAcceleration

Definition at line 99 of file globalsolution.h.

15.19.2 Constructor & Destructor Documentation

15.19.2.1 GlobalSolution::GlobalSolution ( )

This default constructor creates a Global Solution object.

Definition at line 38 of file globalsolution.cpp.

15.19.2.2 GlobalSolution::GlobalSolution (OutputsBody ∗ *input*)

Constructor that also sets the pointer to the OutputsBody object which contains the OutputDerived object.

Parameters

input	Pointer to the OutputsBody objec that contains this OutputDerived object. Pointer passed by value.

See Also

setOutputsBody()

Definition at line 46 of file globalsolution.cpp.

15.19.2.3 GlobalSolution::∼GlobalSolution ( )

The default destructor, nothing happens here.

Definition at line 58 of file globalsolution.cpp.

15.19.3 Member Function Documentation

15.19.3.1 int GlobalSolution::calcOutput (int *freqIn*) [virtual]

Calculates the GlobalSolution output. GlobalSolution is the direct output of the solution for each body, translated into body coordinates. The output is modified to provide the specified order of derivative for the solution.

Parameters

freqIn	The wave frequency to use for calculating the OutputDerived object. Most outputs will depend on the wave frequency.

Returns

Returns a complex matrix that is the GlobalSolution object. The complex matrix is a single column matrix. Each rows in the matrix represents a new degree of freedom variable.

Implements osea::ofreq::OutputDerived.

Definition at line 77 of file globalsolution.cpp.

15.19.3.2 int GlobalSolution::getDerivative ( )

Gets the order of derivative set for the object.

Returns

Integer that specifies the order of the derivative. Value can be anything from 0 or larger. Variable is passed by value.

Definition at line 70 of file globalsolution.cpp.

15.19.3.3 Solution & GlobalSolution::getSolution (double *freqIn*) [protected]

Gets the solution matrix to perform operations on. Accesses the solution matrix from the parent body.

Parameters

freqIn	The wave frequency. Used to identify which variable to access in the solution matrix.

Returns

Returns the Solution object to use for calculations. Variable is passed by reference.

Definition at line 114 of file globalsolution.cpp.

15.19.3.4 void GlobalSolution::setDerivative (int *ord*) [virtual]

Sets the order of the derivative for the object.

Parameters

ord	Integer input that specifies the order of the derivative. Value can be anything from 0 or larger. Variable is passed by value.

Reimplemented in osea::ofreq::GlobalAcceleration, osea::ofreq::GlobalMotion, and osea::ofreq::GlobalVelocity.

Definition at line 63 of file globalsolution.cpp.

15.19.4 Member Data Documentation

15.19.4.1 int osea::ofreq::GlobalSolution::orderDerivative [protected]

The order of derivative to use for calculating output. Set by child classes for specifying specific outputs.

Definition at line 154 of file globalsolution.h.

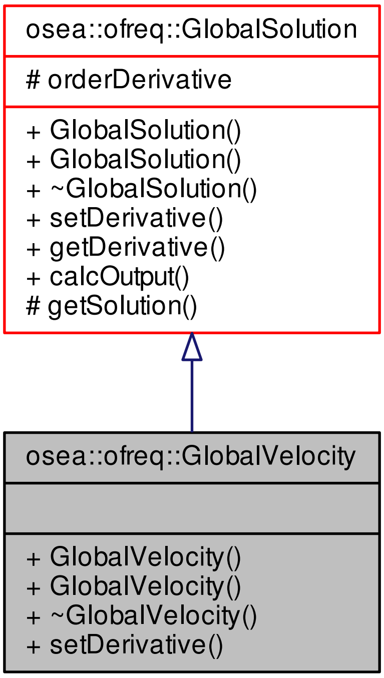
The documentation for this class was generated from the following files:

- bin/ofreq/derived_outputs/globalsolution.h
- bin/ofreq/derived_outputs/globalsolution.cpp

15.20 osea::ofreq::GlobalVelocity Class Reference

`#include <globalvelocity.h>`

Inheritance diagram for osea::ofreq::GlobalVelocity:



Public Member Functions

- GlobalVelocity ()
- GlobalVelocity (OutputsBody ∗input)

    Constructor that also sets the pointer to the OutputsBody object which contains the OutputDerived object.
- ∼GlobalVelocity ()
- void setDerivative (int ord)

    Sets the order of the derivative for the object.

Additional Inherited Members

15.20.1 Detailed Description

This class represents the Global Velocity Solution.

Definition at line 84 of file globalvelocity.h.

15.20.2 Constructor & Destructor Documentation

15.20.2.1 GlobalVelocity::GlobalVelocity ( )

This default constructor creates a Global Velocity object.

Definition at line 41 of file globalvelocity.cpp.

15.20.2.2 GlobalVelocity::GlobalVelocity (**OutputsBody** * *input*)

Constructor that also sets the pointer to the OutputsBody object which contains the OutputDerived object.

Parameters

input	Pointer to the OutputsBody objec that contains this OutputDerived object. Pointer passed by value.

See Also

setOutputsBody()

Definition at line 50 of file globalvelocity.cpp.

15.20.2.3 GlobalVelocity::~GlobalVelocity ( )

The default destructor, nothing happens here.

Definition at line 59 of file globalvelocity.cpp.

15.20.3 Member Function Documentation

15.20.3.1 void GlobalVelocity::setDerivative (int *ord*) [virtual]

Sets the order of the derivative for the object.

Parameters

ord	Integer input that specifies the order of the derivative. Value can be anything from 0 or larger. Variable is passed by value.

Reimplemented from osea::ofreq::GlobalSolution.

Definition at line 64 of file globalvelocity.cpp.

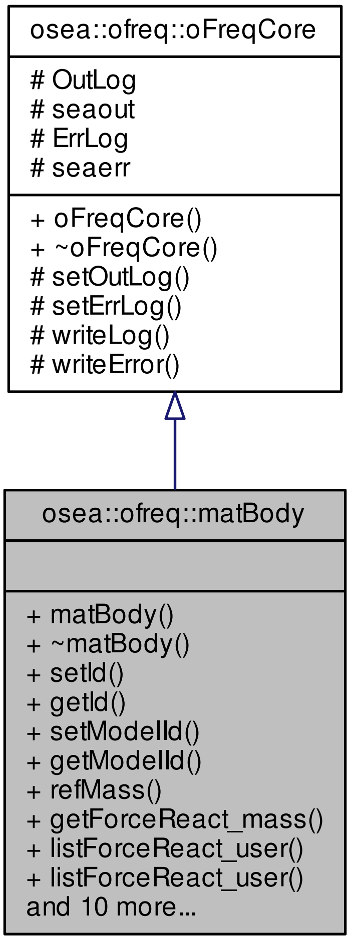
The documentation for this class was generated from the following files:

- bin/ofreq/derived_outputs/globalvelocity.h
- bin/ofreq/derived_outputs/globalvelocity.cpp

15.21 osea::ofreq::matBody Class Reference

```
#include <matbody.h>
```

Inheritance diagram for osea::ofreq::matBody:



Public Member Functions

- matBody ()
- ∼matBody ()
- void setId (int num)

  Sets the force id number for the object.
- int getId ()

  Returns the force id number for the object.
- void setModelId (int num)

  Sets the integer id of the motion model used by the matBody object.
- int getModelId ()

  Gets the integer id of the motion model used by the matBody object.

- arma::cx_mat & refMass ()

    Returns a reference to the mass matrix.
- matForceReact getForceReact_mass ()

    Returns the mass matrix as a reactive force matrix.
- std::vector< matForceReact > & listForceReact_user ()

    Returns a reference to the Reactive Force, user objects.
- matForceReact & listForceReact_user (unsigned int index)

    Returns a reference to the Reactive Force, user object specified by the index.
- std::vector< matForceCross > & listForceCross_user ()

    Returns a reference to the Cross-Body Force, user objects.
- matForceCross & listForceCross_user (unsigned int index)

    Returns a reference to the Cross-Body Force, user object specified by the index.
- std::vector< matForceActive > & listForceActive_user ()

    Returns a reference to the Active Force, user objects.
- matForceActive & listForceActive_user (unsigned int index)

    Returns a reference to the Active Force, user object specified by the index.
- std::vector< matForceReact > & listForceReact_hydro ()

    Returns a reference to the Reactive Force, hydro objects.
- matForceReact & listForceReact_hydro (unsigned int index)

    Returns a reference to the Reactive Force, hydro object specified by the index.
- std::vector< matForceCross > & listForceCross_hydro ()

    Returns a reference to the Cross-Body Force, hydro objects.
- matForceCross & listForceCross_hydro (unsigned int index)

    Returns a reference to the Cross-Body Force, hydro object specified by the index.
- std::vector< matForceActive > & listForceActive_hydro ()

    Returns a reference to the Active Force, hydro objects.
- matForceActive & listForceActive_hydro (unsigned int index)

    Returns a reference to the Active Force, hydro object specified by the index.

Additional Inherited Members

15.21.1 Detailed Description

This class holds all data for a body and related force matrices. The matBody class contains data defined in a pure mathematical context. It is prepared for combination and solution. User interface items, such as relative coordinates and body names are stripped out. Definitions of equations, derivatives, and variables are replaced by sets of matrices. The body still contains force objects, but the objects are defined in terms of matrices. Each force object contains a vector of matrices, denoted by the derivative property. Each matrix within that vector represents a derivative, starting with order 0 (index 0 in the vector.) Each matrix is organized so that rows = equations, and columns = variables.

The class definition also includes a number and size property. The number denotes the body's position within a vector of bodies. The size denotes the number of equations used. This in turn notes how big the matrices must be to accomodate any forces from the body.

Definition at line 108 of file matbody.h.

15.21.2 Constructor & Destructor Documentation

15.21.2.1 matBody::matBody ( )

The default constructor.

Definition at line 37 of file matbody.cpp.

15.21.2.2 matBody::∼matBody ( )

The default destructor, nothing happens here.

Definition at line 42 of file matbody.cpp.

15.21.3 Member Function Documentation

15.21.3.1 matForceReact matBody::getForceReact_mass ( )

Returns the mass matrix as a reactive force matrix.

This allows the motion solver to easily add the body mass in with the other reactive force matrices.

Returns

Returns a matrix reactive force object. Object contains the mass matrix for the body. Returned variabled passed by value.

Definition at line 77 of file matbody.cpp.

15.21.3.2 int matBody::getId ( )

Returns the force id number for the object.

This is similar to the name parameter in other force objects. It is an identifier. In this case, a numerical identifier. Normally correlates to the objects index in a vector of other objects of the same class.

Returns

Returns the force id number, integer data type.

Definition at line 53 of file matbody.cpp.

15.21.3.3 int matBody::getModelId ( )

Gets the integer id of the motion model used by the matBody object.

Gets the integer id of the motion model used by the matBody object.

Returns

Returns the integer id of the motion model used by the matBody object. Variable is passed by value.

Definition at line 65 of file matbody.cpp.

15.21.3.4 vector< matForceActive > & matBody::listForceActive_hydro ( )

Returns a reference to the Active Force, hydro objects.

Returns a reference to the Active Force, hydro objects. This is a vector list of the Active Force objects. Provides direct access to the variable and all the member functions of the vector class.

Returns

This is a vector list of the Active Force objects. Provides direct access to the variable and all the member functions of the vector class. Variable passed by reference.

Definition at line 150 of file matbody.cpp.

15.21.3.5 matForceActive & matBody::listForceActive_hydro (unsigned int *index*)

Returns a reference to the Active Force, hydro object specified by the index.

This is a single item from the vector list of the Active Force objects. Provides direct access to the variable.

Parameters

index	Unsigned integer. Index to specify which variable retrieve from the vector.

Returns

Returns matForceActive object specified by index. Returned variable passed by reference.

See Also

matForceActive

Definition at line 156 of file matbody.cpp.

15.21.3.6 vector< matForceActive > & matBody::listForceActive_user ( )

Returns a reference to the Active Force, user objects.

Returns a reference to the Active Force, user objects. This is a vector list of the Active Force objects. Provides direct access to the variable and all the member functions of the vector class.

Returns

This is a vector list of the Active Force objects. Provides direct access to the variable and all the member functions of the vector class. Variable passed by reference.

Definition at line 114 of file matbody.cpp.

15.21.3.7 matForceActive & matBody::listForceActive_user (unsigned int *index*)

Returns a reference to the Active Force, user object specified by the index.

This is a single item from the vector list of the Active Force objects. Provides direct access to the variable.

Parameters

index	Unsigned integer. Index to specify which variable retrieve from the vector.

Returns

Returns matForceActive object specified by index. Returned variable passed by reference.

See Also

matForceActive

Definition at line 120 of file matbody.cpp.

15.21.3.8 vector< matForceCross > & matBody::listForceCross_hydro ( )

Returns a reference to the Cross-Body Force, hydro objects.

Returns a reference to the Cross-Body Force, hydro objects. This is a vector list of the Cross-Body Force objects. Provides direct access to the variable and all the member functions of the vector class.

Returns

This is a vector list of the Cross-Body Force objects. Provides direct access to the variable and all the member functions of the vector class. Variable passed by reference.

Definition at line 138 of file matbody.cpp.

15.21.3.9 matForceCross & matBody::listForceCross_hydro (unsigned int *index*)

Returns a reference to the Cross-Body Force, hydro object specified by the index.

This is a single item from the vector list of the Cross-Body Force objects. Provides direct access to the variable.

Parameters

index	Unsigned integer. Index to specify which variable retrieve from the vector.

Returns

Returns matForceCross object specified by index. Returned variable passed by reference.

See Also

matForceCross

Definition at line 144 of file matbody.cpp.

15.21.3.10 vector< matForceCross > & matBody::listForceCross_user ( )

Returns a reference to the Cross-Body Force, user objects.

Returns a reference to the Cross-Body Force, user objects. This is a vector list of the Cross-Body Force objects. Provides direct access to the variable and all the member functions of the vector class.

Returns

This is a vector list of the Cross-Body Force objects. Provides direct access to the variable and all the member functions of the vector class. Variable passed by reference.

Definition at line 102 of file matbody.cpp.

15.21.3.11 matForceCross & matBody::listForceCross_user (unsigned int *index*)

Returns a reference to the Cross-Body Force, user object specified by the index.

This is a single item from the vector list of the Cross-Body Force objects. Provides direct access to the variable.

Parameters

index	Unsigned integer. Index to specify which variable retrieve from the vector.

Returns

Returns matForceCross object specified by index. Returned variable passed by reference.

See Also

matForceCross

Definition at line 108 of file matbody.cpp.

15.21.3.12 vector< matForceReact > & matBody::listForceReact_hydro ( )

Returns a reference to the Reactive Force, hydro objects.

Returns a reference to the Reactive Force, hydro objects. This is a vector list of the Reactive Force objects. Provides direct access to the variable and all the member functions of the vector class.

Returns

This is a vector list of the Reactive Force objects. Provides direct access to the variable and all the member functions of the vector class. Variable passed by reference.

Definition at line 126 of file matbody.cpp.

15.21.3.13 matForceReact & matBody::listForceReact_hydro (unsigned int *index*)

Returns a reference to the Reactive Force, hydro object specified by the index.

This is a single item from the vector list of the Reactive Force objects. Provides direct access to the variable.

Parameters

index	Unsigned integer. Index to specify which variable retrieve from the vector.

Returns

Returns matForceReact object specified by index. Returned variable passed by reference.

See Also

matForceReact

Definition at line 132 of file matbody.cpp.

15.21.3.14 vector< matForceReact > & matBody::listForceReact_user ( )

Returns a reference to the Reactive Force, user objects.

Returns a reference to the Reactive Force, user objects. This is a vector list of the Reactive Force objects. Provides direct access to the variable and all the member functions of the vector class.

Returns

This is a vector list of the Reactive Force objects. Provides direct access to the variable and all the member functions of the vector class. Variable passed by reference.

Definition at line 90 of file matbody.cpp.

15.21.3.15 matForceReact & matBody::listForceReact_user (unsigned int *index*)

Returns a reference to the Reactive Force, user object specified by the index.

This is a single item from the vector list of the Reactive Force objects. Provides direct access to the variable.

Parameters

index	Unsigned integer. Index to specify which variable retrieve from the vector.

Returns

Returns matForceReact object specified by index. Returned variable passed by reference.

See Also

matForceReact

Definition at line 96 of file matbody.cpp.

15.21.3.16 cx_mat & matBody::refMass ( )

Returns a reference to the mass matrix.

Returns a reference to the mass matrix.

Returns

Returns a reference to the mass matrix. Variable passed by reference.

Definition at line 71 of file matbody.cpp.

15.21.3.17 void matBody::setId (int *num*)

Sets the force id number for the object.

This is similar to the name parameter in other force objects. It is an identifier. In this case, a numerical identifier. Normally correlates to the objects index in a vector of other objects of the same class.

Parameters

num	The integer number to input as the objects integer id.

Definition at line 47 of file matbody.cpp.

15.21.3.18 void matBody::setModelId (int *num*)

Sets the integer id of the motion model used by the matBody object.

Sets the integer id of the motion model used by the matBody object.

Parameters

num	Integer. The integer id of the motion model used by the matBody object. Variable is passed by value.

Definition at line 59 of file matbody.cpp.

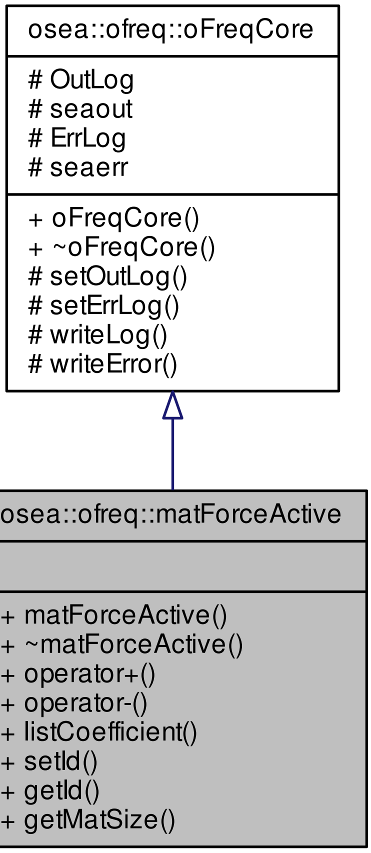
The documentation for this class was generated from the following files:

- bin/ofreq/motion_solver/matbody.h
- bin/ofreq/motion_solver/matbody.cpp

15.22 osea::ofreq::matForceActive Class Reference

`#include <matforceactive.h>`

Inheritance diagram for osea::ofreq::matForceActive:



Public Member Functions

- matForceActive ()
- ∼matForceActive ()
- matForceActive operator+ (matForceActive &forceOther)
- matForceActive operator- (matForceActive &forceOther)
- arma::cx_mat & listCoefficient ()

    Returns the coefficients matrix.
- void setId (int num)

    Sets the force id number for the object.
- int getId ()

    Returns the force id number for the object.
- int getMatSize ()

    Returns the size of the matrix in each order of derivative.

Additional Inherited Members

15.22.1 Detailed Description

This class holds all data for an active force matrix.

Definition at line 85 of file matforceactive.h.

15.22.2 Constructor & Destructor Documentation

15.22.2.1 matForceActive::matForceActive ( )

The default constructor.

Definition at line 34 of file matforceactive.cpp.

15.22.2.2 matForceActive::~matForceActive ( )

The default destructor, nothing happens here.

Definition at line 39 of file matforceactive.cpp.

15.22.3 Member Function Documentation

15.22.3.1 int matForceActive::getId ( )

Returns the force id number for the object.

This is similar to the name parameter in other force objects. It is an identifier. In this case, a numerical identifier. Normally correlates to the objects index in a vector of other objects of the same class.

Returns

Returns the force id number, integer data type.

Definition at line 140 of file matforceactive.cpp.

15.22.3.2 int matForceActive::getMatSize ( )

Returns the size of the matrix in each order of derivative.

Returns the size of the matrix in each order of derivative. Integer output type.

Returns

Returns the size of the matrix in each order of derivative.

Definition at line 146 of file matforceactive.cpp.

15.22.3.3 cx_mat & matForceActive::listCoefficient ( )

Returns the coefficients matrix.

Returns the coefficients matrix.

Returns

Returns the coefficients matrix.

Definition at line 128 of file matforceactive.cpp.

15.22.3.4 matForceActive matForceActive::operator+ (matForceActive & *forceOther*)

Definition at line 44 of file matforceactive.cpp.

15.22.3.5 matForceActive matForceActive::operator- (matForceActive & *forceOther*)

Definition at line 86 of file matforceactive.cpp.

15.22.3.6 void matForceActive::setId (int *num*)

Sets the force id number for the object.

This is similar to the name parameter in other force objects. It is an identifier. In this case, a numerical identifier. Normally correlates to the objects index in a vector of other objects of the same class.

Parameters

num	The integer number to input as the objects integer id.

Definition at line 134 of file matforceactive.cpp.

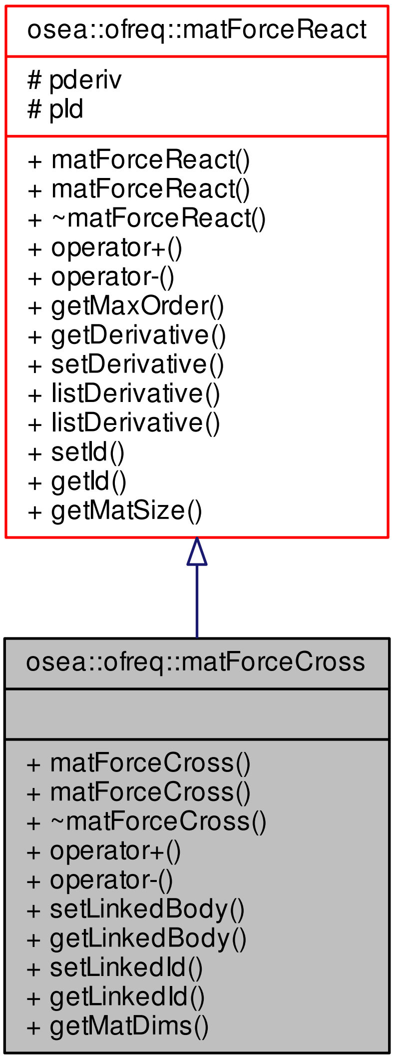
The documentation for this class was generated from the following files:

- bin/ofreq/motion_solver/matforceactive.h
- bin/ofreq/motion_solver/matforceactive.cpp

15.23 osea::ofreq::matForceCross Class Reference

```
#include <matforcecross.h>
```

Inheritance diagram for osea::ofreq::matForceCross:



Public Member Functions

- matForceCross ()
- matForceCross (std::vector< arma::cx_mat > forceIn)

    The constructor. Takes a vector of complex matrices and stores them as derivatives.
- ∼matForceCross ()
- matForceCross operator+ (matForceCross &forceOther)

    Operator overload to add two matForceCross objects together.
- matForceCross operator- (matForceCross &forceOther)

    Operator overload to subtract two matForceCross objects together.
- void setLinkedBody (matBody &BodIn)
- matBody ∗ getLinkedBody ()
- void setLinkedId (int bodId)

Sets the id of the linked body.

- int getLinkedId ()

  Gets the id of the linked body.

- std::vector< int > getMatDims ()

  Returns the size of the matrix in each order of derivative.

Additional Inherited Members

15.23.1 Detailed Description

*This class defines the cross body force matrix. It is an extension of the reactive force matrix class. The main *difference is that this class includes an additional property for the connected body.

Definition at line 93 of file matforcecross.h.

15.23.2 Constructor & Destructor Documentation

15.23.2.1 matForceCross::matForceCross ( )

The default constructor

Definition at line 35 of file matforcecross.cpp.

15.23.2.2 osea::ofreq::matForceCross::matForceCross (std::vector< arma::cx_mat > *forceIn*)

The constructor. Takes a vector of complex matrices and stores them as derivatives.

The constructor. Takes a vector of complex matrices and stores them as derivatives. Assumes that the matrices in the vector are order in sequence of increasing order of derivative. (index 0 = derivative order 0.)

Parameters

forceIn	The list of forces.

15.23.2.3 matForceCross::~matForceCross ( )

The default destructor. Nothing happens here.

Definition at line 55 of file matforcecross.cpp.

15.23.3 Member Function Documentation

15.23.3.1 matBody * matForceCross::getLinkedBody ( )

Return the linked body for the cross body object

Returns

Returns a pointer to the matBody object that this force relates to.

Definition at line 239 of file matforcecross.cpp.

15.23.3.2 int matForceCross::getLinkedId ( )

Gets the id of the linked body.

Gets the id of the linked body. The id is like the body's name. This is normally the index of the body within the vector of other bodies.

Returns

Integer value which is the body's id. This is normally the index of the body within the vector of other bodies.

Definition at line 251 of file matforcecross.cpp.

15.23.3.3 vector< int > matForceCross::getMatDims ( )

Returns the size of the matrix in each order of derivative.

Returns the size of the matrix in each order of derivative. Integer output type. Reports both number of columns and number of rows. Vector of 2 integers output.

Returns

Returns a vector of two integers specifying size of matrix. First output is number of rows. Second output is number of columns.

Definition at line 257 of file matforcecross.cpp.

15.23.3.4 matForceCross matForceCross::operator+ (matForceCross & *forceOther*)

Operator overload to add two matForceCross objects together.

This overloads the + operator to add two matForceCross objects together. Functions are added on a per-derivative basis. The function recognizes the derivative matrices contained within each object. Only derivatives of the same order are added together. The function also checks the linked body parameter. Only objects with the same linked body are added together.

Parameters

forceOther	The other objects of type matForceCross that will be added.

Returns

Returns an object of type matForceCross. The new object will contain the same order of derivatives as the highest derivative of the two added functions.

Definition at line 60 of file matforcecross.cpp.

15.23.3.5 matForceCross matForceCross::operator- (matForceCross & *forceOther*)

Operator overload to subtract two matForceCross objects together.

This overloads the - operator to subtract two matForceCross objects together. Functions are subtracted on a per-derivative basis. The function recognizes the derivative matrices contained within each object. Only derivatives of the same order are subtracted together. Order of operations does matter. The function also checks the linked body parameter. Only objects with the same linked body are added together.

Parameters

forceOther	The other objects of type matForceCross that will be subtracted. forceOther is always subtracted from the calling object.

Returns

Returns an object of type matForceCross. The new object will contain the same order of derivatives as the highest derivative of the two subtracted functions.

Definition at line 145 of file matforcecross.cpp.

15.23.3.6 void matForceCross::setLinkedBody (matBody & *BodIn*)

Set linked body for cross body object.

Parameters

BodIn	pointer to the matBody object that this linked force relates to.

Definition at line 230 of file matforcecross.cpp.

15.23.3.7 void matForceCross::setLinkedId (int *bodId*)

Sets the id of the linked body.

Sets the id of the linked body. The id is like the body's name. This is normally the index of the body within the vector of other bodies.

Parameters

bodId	The integer of the body id. This is normally the index of the body within the vector of other bodies.

Definition at line 245 of file matforcecross.cpp.

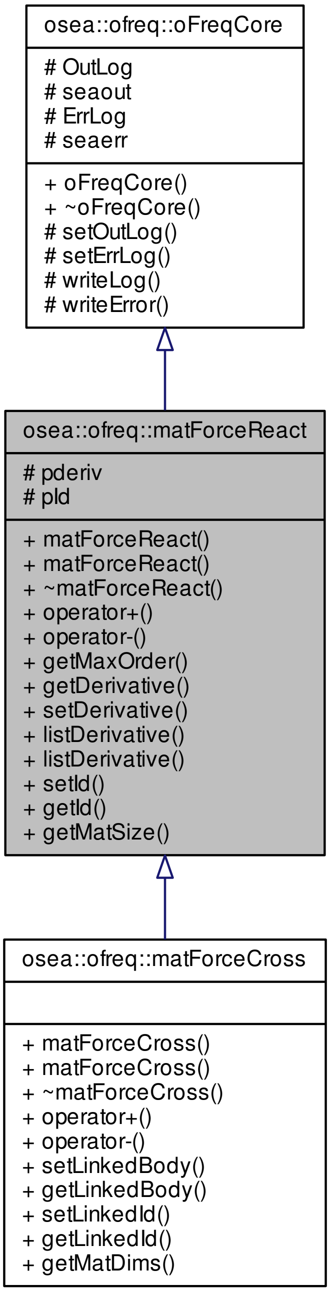
The documentation for this class was generated from the following files:

- bin/ofreq/motion_solver/matforcecross.h
- bin/ofreq/motion_solver/matforcecross.cpp

15.24 osea::ofreq::matForceReact Class Reference

```
#include <matforcereact.h>
```

Inheritance diagram for osea::ofreq::matForceReact:



Public Member Functions

- matForceReact ()
- matForceReact (std::vector< arma::cx_mat > forceIn)

  The constructor. Takes a vector of complex matrices and stores them as derivatives.
- virtual ∼matForceReact ()
- virtual matForceReact operator+ (matForceReact &forceOther)

Operator overload to add two matForceReact objects together.

- virtual matForceReact operator- (matForceReact &forceOther)

    Operator overload to subtract two matForceReact objects together.
- int getMaxOrder ()

    The maximum order of the derivatives.
- arma::cx_mat getDerivative (int order)

    Derivative Returns the complex matrix for only the order of derivative specified.
- void setDerivative (unsigned int order, arma::cx_mat Coeff)

    Inputs a derivative matrix.
- std::vector< arma::cx_mat > & listDerivative ()

    Provides direct access to the vector of derivatives.
- arma::cx_mat & listDerivative (unsigned int index)

    Provides direct access to the derivative specified by the index.
- void setId (int num)

    Sets the force id number for the object.
- int getId ()

    Returns the force id number for the object.
- int getMatSize ()

    Returns the size of the matrix in each order of derivative.

Protected Attributes

- std::vector< arma::cx_mat > pderiv

    Defines the vector of derivatives.
- int pId

    the number of the object in the outside vector that contains it.

Additional Inherited Members

15.24.1 Detailed Description

This class holds data for reactive force matrix whch includes force coefficients.

Definition at line 89 of file matforcereact.h.

15.24.2 Constructor & Destructor Documentation

15.24.2.1 matForceReact::matForceReact ( )

The default constructor.

Definition at line 34 of file matforcereact.cpp.

15.24.2.2 osea::ofreq::matForceReact::matForceReact (std::vector< arma::cx_mat > *forceIn*)

The constructor. Takes a vector of complex matrices and stores them as derivatives.

The constructor. Takes a vector of complex matrices and stores them as derivatives. Assumes that the matrices in the vector are order in sequence of increasing order of derivative. (index 0 = derivative order 0.)

Parameters

forceIn	The list of forces.

15.24.2.3 matForceReact::∼matForceReact ( ) [virtual]

The default destructor, nothing happens here.

Definition at line 39 of file matforcereact.cpp.

15.24.3 Member Function Documentation

15.24.3.1 cx_mat matForceReact::getDerivative (int *order*)

Derivative Returns the complex matrix for only the order of derivative specified.

Derivative Returns the complex matrix for only the order of derivative specified.

Parameters

order	Integer input to specify the order of the derivative.

Returns

Returns a complex matrix that contains the force coefficients for the given order of derivative. Passed as a value.

Definition at line 204 of file matforcereact.cpp.

15.24.3.2 int matForceReact::getId ( )

Returns the force id number for the object.

This is similar to the name parameter in other force objects. It is an identifier. In this case, a numerical identifier. Normally correlates to the objects index in a vector of other objects of the same class.

Returns

Returns the force id number, integer data type.

Definition at line 291 of file matforcereact.cpp.

15.24.3.3 int matForceReact::getMatSize ( )

Returns the size of the matrix in each order of derivative.

Returns the size of the matrix in each order of derivative. Integer output type.

Returns

Returns the size of the matrix in each order of derivative.

Definition at line 297 of file matforcereact.cpp.

15.24.3.4 int matForceReact::getMaxOrder ( )

The maximum order of the derivatives.

The maximum order of the derivatives (Integer). Also the total size of the vector containing the derivatives.

Returns

Returns the maximum order of derivatives in the force.

Definition at line 195 of file matforcereact.cpp.

15.24.3.5 vector< cx_mat > & matForceReact::listDerivative ( )

Provides direct access to the vector of derivatives.

Provides direct access to the vector of derivatives. Allows for use of vector operations on the derivatives object.

Returns

Returns reference to the vector of complex matrices which contain the derivatives. Variable passed by reference.

Definition at line 227 of file matforcereact.cpp.

15.24.3.6 cx_mat & matForceReact::listDerivative (unsigned int *index*)

Provides direct access to the derivative specified by the index.

Allows for direct access to edit the derivative or just retrieve information from. Index is also the order of the derivative.

Parameters

index	Unsigned integer. Specifies the index of which derivative to retrieve from the list.

Returns

Complex matrix returned. Returns the complex matrix for the derivative specified by the index. Returned variable is passed by reference.

Definition at line 233 of file matforcereact.cpp.

15.24.3.7 matForceReact matForceReact::operator+ (matForceReact & *forceOther*) [virtual]

Operator overload to add two matForceReact objects together.

This overloads the + operator to add two matForceReact objects together. Functions are added on a per-derivative basis. The function recognizes the derivative matrices contained within each object. Only derivatives of the same order are added together.

Parameters

forceOther	The other objects of type matForceReact that will be added.

Returns

Returns an object of type matForceReact. The new object will contain the same order of derivatives as the highest derivative of the two added functions.

Definition at line 53 of file matforcereact.cpp.

15.24.3.8 matForceReact matForceReact::operator- (matForceReact & *forceOther*) [virtual]

Operator overload to subtract two matForceReact objects together.

This overloads the - operator to subtract two matForceReact objects together. Functions are subtracted on a per-derivative basis. The function recognizes the derivative matrices contained within each object. Only derivatives of the same order are subtracted together. Order of operations does matter.

Parameters

forceOther	The other objects of type matForceReact that will be subtracted. forceOther is always subtracted from the calling object.

Returns

Returns an object of type matForceReact. The new object will contain the same order of derivatives as the highest derivative of the two subtracted functions.

Definition at line 124 of file matforcereact.cpp.

15.24.3.9 void matForceReact::setDerivative (unsigned int *order,* arma::cx_mat *Coeff*)

Inputs a derivative matrix.

Parameters

order	The order of the derivative matrix. Also is sequence in the vector that contains the matrices.
Coeff	The matrix of complex numbers that contains the force coefficients for the derivative. Passed as a value, not a reference.

Definition at line 210 of file matforcereact.cpp.

15.24.3.10 void matForceReact::setId (int *num*)

Sets the force id number for the object.

This is similar to the name parameter in other force objects. It is an identifier. In this case, a numerical identifier. Normally correlates to the objects index in a vector of other objects of the same class.

Parameters

num	The integer number to input as the objects integer id.

Definition at line 285 of file matforcereact.cpp.

15.24.4 Member Data Documentation

15.24.4.1 std::vector<arma::cx_mat> osea::ofreq::matForceReact::pderiv [protected]

Defines the vector of derivatives.

Defines the vector of derivatives. Each entry in vector represents the order of the derivative.

Definition at line 225 of file matforcereact.h.

15.24.4.2 int osea::ofreq::matForceReact::pId [protected]

the number of the object in the outside vector that contains it.

This is similar to the name parameter in other force objects. It is an identifier. In this case, a numerical identifier. Normally correlates to the objects index in a vector of other objects of the same class.

Definition at line 234 of file matforcereact.h.

The documentation for this class was generated from the following files:

- bin/ofreq/motion_solver/matforcereact.h

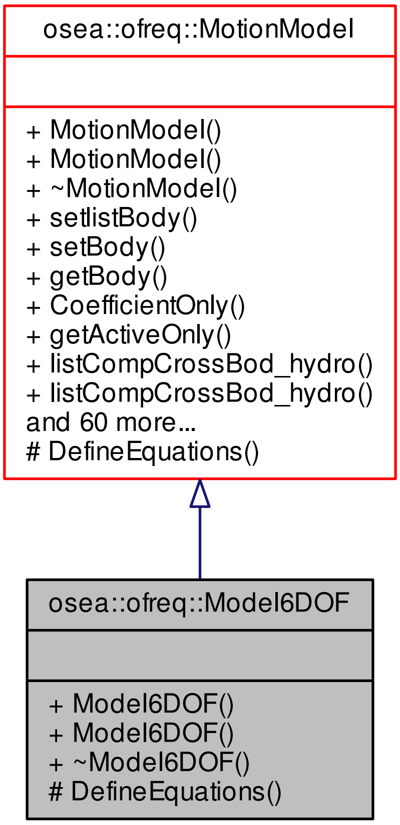
- bin/ofreq/motion_solver/matforcereact.cpp

15.25 osea::ofreq::Model6DOF Class Reference

The motion model for standard six-degree of freedom rigid-body dynamics problems.

`#include <model6dof.h>`

Inheritance diagram for osea::ofreq::Model6DOF:



Public Member Functions

- Model6DOF ()

  The default constructor.
- Model6DOF (std::vector< Body > &listBodIn)

  Constructor. This is the preferred constructor as it supplies the body data.
- ∼Model6DOF ()

Protected Member Functions

- void DefineEquations ()

The function used to define the equation of motion objects.

15.25.1 Detailed Description

The motion model for standard six-degree of freedom rigid-body dynamics problems.

This is where each EquationofMotion object is created. Can also be any object from a class that is derived from the EquationofMotion object. Before adding equations within this motion model, the individual equation must be defined. These will be new objects inheritted from the EquationofMotion object. Once those new equation clases are defined, they can be used in the motion model.

Using an EquationofMotion in the motion model will generally follow the following sequence. All step are executed within the DefineEquations function. 1.) Create a new object from the appropriate class which is derived from the EquationofMotion. When creating the equation of motion, you must include the pointer to the existing motion model. Use the keyword this when creating the new object. 2.) Set the data index for the equation. This is probably the most important step. Regardless of what name you give the equation, the program ofreq only sees the equation as one in a list of equations, and refers to it by its index within that list. Any input data (such as hydrodynamic or user coefficients) is similarly referenced by that index. When you set the data index, you tell ofreq which index in the list of data is has available should correspond to this specific equation. (Set the data index using the function setDataIndex(). 3.) Set the name for the new object. This is just the short name or equation symbol. (Use the function setName() to set it.) 4.) Set the description for the new object. This is the more extensive name for the equation. (Use the function setDescription() to set it.) 5.) Now that you set all the appropriate information, add the equation of motion into the list of equations used by this motion model. (Use the function AddEquation()).

Once you define all the equations, you will also want to define a name for your motion model. This is the name you will use to select the motion model within the input files. It can be any sequence you want and include spaces. One name is already reserved as part of the standard program models. You can not use the following name: "6DOF" - Reserved.∗

See Also

AddEquation()
EquationofMotion

Definition at line 121 of file model6dof.h.

15.25.2 Constructor & Destructor Documentation

15.25.2.1 Model6DOF::Model6DOF ( )

The default constructor.

Definition at line 36 of file model6dof.cpp.

15.25.2.2 Model6DOF::Model6DOF (std::vector< Body > & *listBodIn*)

Constructor. This is the preferred constructor as it supplies the body data.

Parameters

listBodIn	The vector of the body objects to input.

Definition at line 44 of file model6dof.cpp.

15.25.2.3 Model6DOF::∼Model6DOF ( )

Default destructor.

Definition at line 52 of file model6dof.cpp.

15.25.3 Member Function Documentation

15.25.3.1 void Model6DOF::DefineEquations ( ) [protected], [virtual]

The function used to define the equation of motion objects.

This function gets executed when the Motion model is first created. It contains all the statements to add the appropriate equations to the motion model. This is where each EquationofMotion object is created. Can also be any object from a class that is derived from the EquationofMotion object. Before definining equations within this motion model, the individual equation must be defined. These will be new objects inheritted from the EquationofMotion object. Once those new equation clases are defined, they can be used in the motion model.

Using an EquationofMotion in the motion model will generally follow the following sequence. All step are executed within the DefineEquations function. 1.) Create a new object from the appropriate class which is derived from the EquationofMotion. When creating the equation of motion, you must include the pointer to the existing motion model. Use the keyword this when creating the new object. 2.) Set the data index for the equation. This is probably the most important step. Regardless of what name you give the equation, the program ofreq only sees the equation as one in a list of equations, and refers to it by its index within that list. Any input data (such as hydrodynamic or user coefficients) is similarly referenced by that index. When you set the data index, you tell ofreq which index in the list of data is has available should correspond to this specific equation. (Set the data index using the function setDataIndex(). 3.) Set the name for the new object. This is just the short name or equation symbol. (Use the function setName() to set it.) 4.) Set the description for the new object. This is the more extensive name for the equation. (Use the function setDescription() to set it.) 5.) Now that you set all the appropriate information, add the equation of motion into the list of equations used by this motion model. (Use the function AddEquation()).

Once you define all the equations, you will also want to define a name for your motion model. This is the name you will use to select the motion model within the input files. It can be any sequence you want and include spaces. One name is already reserved as part of the standard program models. You can not use the following name: "6DOF" - Reserved.

See Also

AddEquation()
EquationofMotion

Reimplemented from osea::ofreq::MotionModel.

Definition at line 77 of file model6dof.cpp.

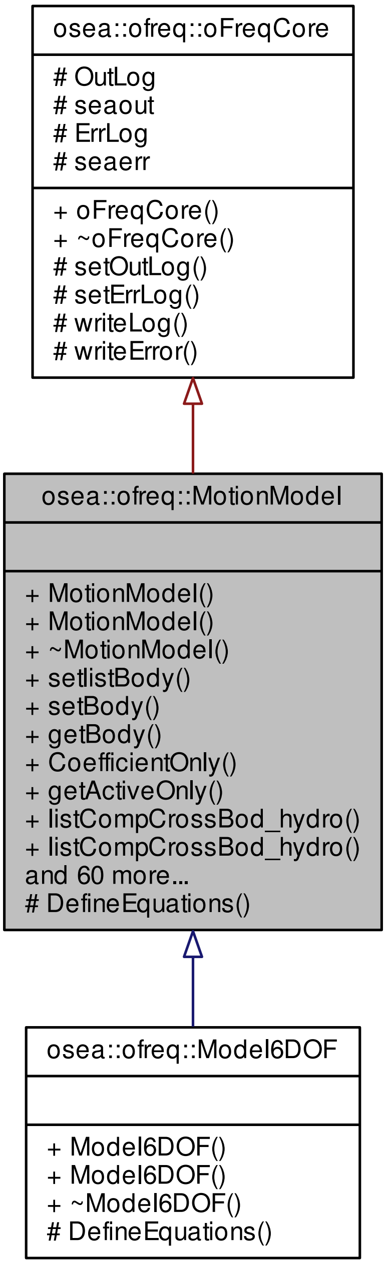
The documentation for this class was generated from the following files:

- bin/ofreq/motion_model/model6dof.h
- bin/ofreq/motion_model/model6dof.cpp

15.26 osea::ofreq::MotionModel Class Reference

```
#include <motionmodel.h>
```

Inheritance diagram for osea::ofreq::MotionModel:



Public Member Functions

- MotionModel ()
- MotionModel (std::vector< Body > &listBodIn)

  Constructor. This is the preferred constructor as it supplies the body data.
- virtual ∼MotionModel ()
- void setlistBody (std::vector< Body > &listBodIn)

Inputs the list of body data.
- void setBody (int bod)

  Sets the index for the body that all calculations are based on.
- int getBody ()

  Gets the index for the body that all calculations are based on.
- bool & CoefficientOnly ()

  Determines whether the class should calculate force coefficients or actual force values. True = Calculate force coefficients only. False = Calculate force values. Default = (False) Calculate force values.
- bool getActiveOnly ()

  Boolean to track whether only the active forces are requested.
- std::vector< int > & listCompCrossBod_hydro ()

  Records the index of the body object referenced by the cross body.
- int & listCompCrossBod_hydro (int crossbodIn)

  Records the index of the body object referenced by the cross body.
- std::vector< int > & listCompCrossBod_user ()

  Records the index of the body object referenced by the cross body.
- int & listCompCrossBod_user (int crossbodIn)

  Records the index of the body object referenced by the cross body.
- void Reset ()

  Resets the class data to have all input coefficients. Any evaluation after a reset will produce a value of zero. Force coefficients will be zero and force values will be zero.
- void setFreq (double freq)

  Sets the current operating frequency for the function. Only necessary when calculating true forces and using derivatives defined in the motion model. Otherwise, you can safely ignore this function.
- double getFreq ()

  Gets the current operating frequency for the function. Only necessary when calculating true forces and using derivatives defined in the motion model. Otherwise, you can safely ignore this function.
- void useForceActive_user (unsigned int force, unsigned int eqn)

  Passes information to the object to use input coefficients from the entry specified.
- void useForceActive_user (unsigned int force)

  Passes information to the object to use input coefficients from the entry specified.
- void useForceActive_user ()

  Passes information to the object to use input coefficients from the entry specified.
- void useForceActive_hydro (unsigned int force, unsigned int eqn)

  Passes information to the object to use input coefficients from the entry specified.
- void useForceActive_hydro (unsigned int force)

  Passes information to the object to use input coefficients from the entry specified.
- void useForceActive_hydro ()

  Passes information to the object to use input coefficients from the entry specified.
- void useForceReact_user (unsigned int force, unsigned int ord, unsigned int eqn, unsigned int var)

  Passes information to the object to use input coefficients from the entry specified.
- void useForceReact_user (unsigned int force, unsigned int ord, unsigned int eqn)

  Passes information to the object to use input coefficients from the entry specified.
- void useForceReact_user (unsigned int force, unsigned int ord)

  Passes information to the object to use input coefficients from the entry specified.
- void useForceReact_user (unsigned int force)

  Passes information to the object to use input coefficients from the entry specified.
- void useForceReact_user ()

  Passes information to the object to use input coefficients from the entry specified.
- void useForceReact_hydro (unsigned int force, unsigned int ord, unsigned int eqn, unsigned int var)

  Passes information to the object to use input coefficients from the entry specified.

- void useForceReact_hydro (unsigned int force, unsigned int ord, unsigned int eqn)
  Passes information to the object to use input coefficients from the entry specified.
- void useForceReact_hydro (unsigned int force, unsigned int ord)
  Passes information to the object to use input coefficients from the entry specified.
- void useForceReact_hydro (unsigned int force)
  Passes information to the object to use input coefficients from the entry specified.
- void useForceReact_hydro ()
  Passes information to the object to use input coefficients from the entry specified.
- void useForceCross_user (unsigned int force, unsigned int ord, unsigned int eqn, unsigned int var)
  Passes information to the object to use input coefficients from the entry specified.
- void useForceCross_user (unsigned int force, unsigned int ord, unsigned int eqn)
  Passes information to the object to use input coefficients from the entry specified.
- void useForceCross_user (unsigned int force, unsigned int ord)
  Passes information to the object to use input coefficients from the entry specified.
- void useForceCross_user (unsigned int force)
  Passes information to the object to use input coefficients from the entry specified.
- void useForceCross_user ()
  Passes information to the object to use input coefficients from the entry specified.
- void useForceCross_hydro (unsigned int force, unsigned int ord, unsigned int eqn, unsigned int var)
  Passes information to the object to use input coefficients from the entry specified.
- void useForceCross_hydro (unsigned int force, unsigned int ord, unsigned int eqn)
  Passes information to the object to use input coefficients from the entry specified.
- void useForceCross_hydro (unsigned int force, unsigned int ord)
  Passes information to the object to use input coefficients from the entry specified.
- void useForceCross_hydro (unsigned int force)
  Passes information to the object to use input coefficients from the entry specified.
- void useForceCross_hydro ()
  Passes information to the object to use input coefficients from the entry specified.
- void useForceMass (unsigned int eqn, unsigned int var)
  Passes information to the object to use input coefficients from the entry specified.
- void useForceMass (unsigned int eqn)
  Passes information to the object to use input coefficients from the entry specified.
- void useForceMass ()
  Passes information to the object to use input coefficients from the entry specified.
- arma::cx_mat getMatForceActive_user (int force)
  Evaluates the motion model for a whole range of equations on the specified force.
- arma::cx_mat getMatForceActive_hydro (int force)
  Evaluates the motion model for a whole range of equations on the specified force.
- arma::cx_mat getMatForceReact_user (int force, int ord)
  Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative.
- arma::cx_mat getMatForceReact_hydro (int force, int ord)
  Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative.
- arma::cx_mat getMatForceCross_user (int force, int ord)
  Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative.
- arma::cx_mat getMatForceCross_hydro (int force, int ord)
  Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative.
- arma::cx_mat getMatForceMass ()
  Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative.
- std::complex< double > Evaluate (int eqn)
  Triggers evaluation of the currently activated set of input coefficients.
- int numEquations ()

Reports the number of equations used in the motion model.

- std::vector< int > & listDataIndex ()

  Returns a vector containing all equation indices. This may be the same as the number of equations. However, if they are custom equations, they must avoid the first six indices, which are reserved for standard 6dof models. This means that the data index may not start at zero, which is why the data index vector is returned. It allows you to see for each entry in the slot, what the index is for that equation.

- int & listDataIndex (unsigned int index)

  Returns an enry from a vector containing all equation indices.

- std::vector< int > getDataIndex ()

  Assembles and gets the vector of equation indices.

- int MaxDataIndex ()

  Returns the maximum number of the data index.

- void setName (std::string nameIn)

  Name for the motion model.

- std::string getName ()

  Name for the motion model.

- void setDescription (std::string DescIn)

  Description for the motion model.

- std::string getDescription ()

  Description for the motion model.

- std::vector< Body > & listBody ()

  Provides direct access to the list of Bodies referenced by the motion model.

- Body & listBody (int bodIn)

  Direct access to an individual Body from the list of Bodies contained in the motion model.

- std::vector< Body > & listData ()

  Provides direct access to the list of Bodies used as data for the motion model.

- Body & listData (int dataIn)

  Direct access to an individual Body from the list of Data contained in the motion model.

- std::vector< EquationofMotion * > & listEquation ()

  Provides direct access to the list of equation of motion objects used in the motion model.

- EquationofMotion & listEquation (int eqIn)

  Direct access to an individual EquationofMotion object from the list of Equations contained in the motion model.

- std::vector< EquationofMotion * > & listDataEquation ()

  Provides direct access to the list of equation of motion objects used in the motion model.

- EquationofMotion & listDataEquation (int eqIn)

  Direct access to an individual EquationofMotion object from the list of Equations contained in the motion model.

- void AddEquation (EquationofMotion *eqIn)

  This adds an equation of motion to the motion model.

Protected Member Functions

- virtual void DefineEquations ()

  The function used to define the equation of motion objects.

Additional Inherited Members

15.26.1 Detailed Description

This class provides the functionality to translate between input coefficients in the body class and the force coefficients in the matBody class. Most important, it acts as an interface for advanced users to enter their own equations of motion. This was devised to create a very generic interface that could allow any sort of definition for equations. The use of functions for the class should use the following sequence. 1.) Create class: constructor 2.) Set body data (if not already done in constructor): setListBodies 3.) Set the current body working with: setBody 4.) Set the current wave frequency working with: setFreq 5.) Set whether calculating coefficients or values (default: Values): calcCoefficient 6.) Reset the forces you wish to use. 7.) Set the new list of forces you wish to use.: useForceAct_usr useForceAct_hydro useForceReact_usr useForceReact_hydro useForceCross_usr useForceCross_hydro useForceMass 7.) Evaluate the motion model to produce a single complex value result.

Definition at line 106 of file motionmodel.h.

15.26.2 Constructor & Destructor Documentation

15.26.2.1 MotionModel::MotionModel ( )

Default constructor.

Definition at line 37 of file motionmodel.cpp.

15.26.2.2 osea::ofreq::MotionModel::MotionModel (std::vector< **Body** > & *listBodIn*)

Constructor. This is the preferred constructor as it supplies the body data.

Parameters

listBodIn	The vector of the body objects to input.

15.26.2.3 MotionModel::∼MotionModel ( ) [virtual]

Default destructor.

Definition at line 53 of file motionmodel.cpp.

15.26.3 Member Function Documentation

15.26.3.1 void MotionModel::AddEquation (**EquationofMotion** ∗ *eqIn*)

This adds an equation of motion to the motion model.

Adds the equation of motion on to the end of the vector of equation of motions. Also works for any objects derived from the EquationofMotion object, which is how this method should really be used.

Parameters

eqIn	EquationfMotion object. The object that you want to add to the list of equations of motion. Also works for any object classes derived from the EquationOfMotion. Variable passed by value, so it will make a copy of the input variable.

See Also

DefineEquations()

Definition at line 2075 of file motionmodel.cpp.

15.26.3.2 bool & MotionModel::CoefficientOnly ( )

Determines whether the class should calculate force coefficients or actual force values. True = Calculate force coefficients only. False = Calculate force values. Default = (False) Calculate force values.

Returns

Boolean to determine whether should calculate coefficients or values.

Definition at line 96 of file motionmodel.cpp.

15.26.3.3 void MotionModel::DefineEquations ( ) [protected], [virtual]

The function used to define the equation of motion objects.

This function gets executed when the Motion model is first created. It contains all the statements to add the appropriate equations to the motion model. This is where each EquationofMotion object is created. Can also be any object from a class that is derived from the EquationofMotion object. Before definining equations within this motion model, the individual equation must be defined. These will be new objects inheritted from the EquationofMotion object. Once those new equation clases are defined, they can be used in the motion model.

Using an EquationofMotion in the motion model will generally follow the following sequence. All step are executed within the DefineEquations function. 1.) Create a new object from the appropriate class which is derived from the EquationofMotion. When creating the equation of motion, you must include the pointer to the existing motion model. Use the keyword this when creating the new object. 2.) Set the data index for the equation. This is probably the most important step. Regardless of what name you give the equation, the program ofreq only sees the equation as one in a list of equations, and refers to it by its index within that list. Any input data (such as hydrodynamic or user coefficients) is similarly referenced by that index. When you set the data index, you tell ofreq which index in the list of data is has available should correspond to this specific equation. (Set the data index using the function setDataIndex(). 3.) Set the name for the new object. This is just the short name or equation symbol. (Use the function setName() to set it.) 4.) Set the description for the new object. This is the more extensive name for the equation. (Use the function setDescription() to set it.) 5.) Now that you set all the appropriate information, add the equation of motion into the list of equations used by this motion model. (Use the function AddEquation()).

See Also

AddEquation()
EquationofMotion

Reimplemented in osea::ofreq::Model6DOF.

Definition at line 2087 of file motionmodel.cpp.

15.26.3.4 complex< double > MotionModel::Evaluate (int *eqn*)

Triggers evaluation of the currently activated set of input coefficients.

Triggers evaluation of the currently activated set of input coefficients. If Calc_Coeff is set to True, then evaluation will only generate the force coefficients from the resulting evaluation. Otherwise, the evaluation will use the currently defined solution data and evaluate for force values.

Parameters

eqn	Integer representing which equation object to evaluate. Integer specifies the Data index of the equation.

Returns

Returns a complex number representing the force under the currently set conditions.

Definition at line 1903 of file motionmodel.cpp.

15.26.3.5 bool MotionModel::getActiveOnly ( )

Boolean to track whether only the active forces are requested.

Boolean to track whether only the active forces are requested. The active forces are included negatively in the equation of motion. They should be on the opposite side of the equation and included as a positive constant. The final matrix body accomplishes this. And when only active forces are requested, they should be sent out as positive values. However, when pulling the information out, the signs must be reversed. The boolean variable triggers to determine if this should happen. If any reactive or cross-body forces are activated as well, this variable is set false.

Returns

Returns boolean variable. Variable passed by value. Returns true if only active forces are used in the equation of motion. Returns false if any reactive or cross-body forces are used in the equation of motion.

Definition at line 103 of file motionmodel.cpp.

15.26.3.6 int MotionModel::getBody ( )

Gets the index for the body that all calculations are based on.

Returns

Returns integer specifying the number of the body currently in use. Integer corresponds to the sequence of bodies in the vector supplied with the body. If no Body is currently set, the function returns -1.

Definition at line 87 of file motionmodel.cpp.

15.26.3.7 std::vector< int > MotionModel::getDataIndex ( )

Assembles and gets the vector of equation indices.

The list of equation indices may be the same as the number of equations. However, if they are custom equations, they must avoid the first six indices, which are reserved for standard 6dof models. This means that the data index may not start at zero, which is why the data index vector is returned. It allows you to see for each entry in the slot, what the index is for that equation. This method also searches through all the included equation objects to retrieve their data index automatically. So if you have an imcomplete list, this method will automatically complete the list before returning the vector of the complete list of data indices.

Returns

Returns a vector containing all the equation indices currently in use.

Definition at line 1962 of file motionmodel.cpp.

15.26.3.8 string MotionModel::getDescription ( )

Description for the motion model.

Description for the motion model. Used by the user to provide a more extensive description of the motion model. Used purely for user information. Not used for model identification.

Returns

std::string. The description for the motion model. Variable passed by value.

Definition at line 2017 of file motionmodel.cpp.

15.26.3.9 double MotionModel::getFreq ( )

Gets the current operating frequency for the function. Only necessary when calculating true forces and using derivatives defined in the motion model. Otherwise, you can safely ignore this function.

Returns

Double precision variable that is the current wave frequency value. Variable returned by value.

Definition at line 198 of file motionmodel.cpp.

15.26.3.10 cx_mat MotionModel::getMatForceActive_hydro (int *force*)

Evaluates the motion model for a whole range of equations on the specified force.

Evaluates the motion model for a whole range of equations on the specified force. Returns a complex matrix that contains the results of the entire evaluation.

Parameters

force	Integer specifying which force object to evaluate. Integer specifies the vector occurrence index of the force.

Returns

Returns a complex matrix that contains the results of the entire evaluation. Returned argument passed by value.

Definition at line 1654 of file motionmodel.cpp.

15.26.3.11 cx_mat MotionModel::getMatForceActive_user (int *force*)

Evaluates the motion model for a whole range of equations on the specified force.

Evaluates the motion model for a whole range of equations on the specified force. Returns a complex matrix that contains the results of the entire evaluation.

Parameters

force	Integer specifying which force object to evaluate. Integer specifies the vector occurrence index of the force.

Returns

Returns a complex matrix that contains the results of the entire evaluation. Returned argument passed by value.

Definition at line 1623 of file motionmodel.cpp.

15.26.3.12 cx_mat MotionModel::getMatForceCross_hydro (int *force,* int *ord*)

Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative.

Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative. Returns a complex matrix that contains the results of the entire evaluation. Saves some time on computing effort.

Parameters

force	Integer specifying the force object to use. Integer specifies the vector occurrence index of the force.
ord	Integer specifying which order of derivative to use on the specified force object.

Returns

Returns a complex matrix that contains the results of the entire evaluation. Saves some time on computing effort.

Definition at line 1817 of file motionmodel.cpp.

15.26.3.13 cx_mat MotionModel::getMatForceCross_user (int *force,* int *ord*)

Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative.

Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative. Returns a complex matrix that contains the results of the entire evaluation. Saves some time on computing effort.

Parameters

force	Integer specifying the force object to use. Integer specifies the vector occurrence index of the force.
ord	Integer specifying which order of derivative to use on the specified force object.

Returns

Returns a complex matrix that contains the results of the entire evaluation. Saves some time on computing effort.

Definition at line 1774 of file motionmodel.cpp.

15.26.3.14 cx_mat MotionModel::getMatForceMass ( )

Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative.

Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative. Returns a complex matrix that contains the results of the entire evaluation. Saves some time on computing effort.

Returns

Returns a complex matrix that contains the results of the entire evaluation. Saves some time on computing effort.

Definition at line 1860 of file motionmodel.cpp.

15.26.3.15 cx_mat MotionModel::getMatForceReact_hydro (int *force*, int *ord*)

Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative.

Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative. Returns a complex matrix that contains the results of the entire evaluation. Saves some time on computing effort.

Parameters

force	Integer specifying the force object to use. Integer specifies the vector occurrence index of the force.
ord	Integer specifying which order of derivative to use on the specified force object.

Returns

Returns a complex matrix that contains the results of the entire evaluation. Saves some time on computing effort.

Definition at line 1731 of file motionmodel.cpp.

15.26.3.16 cx_mat MotionModel::getMatForceReact_user (int *force*, int *ord*)

Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative.

Evaluates the motion model for a whole range of equations and variable on the specified force and order of derivative. Returns a complex matrix that contains the results of the entire evaluation. Saves some time on computing effort.

Parameters

force	Integer specifying the force object to use. Integer specifies the vector occurrence index of the force.
ord	Integer specifying which order of derivative to use on the specified force object.

Returns

Returns a complex matrix that contains the results of the entire evaluation. Saves some time on computing effort.

Definition at line 1685 of file motionmodel.cpp.

15.26.3.17 string MotionModel::getName ( )

Name for the motion model.

Name for the motion model. Used by the user to identify the motion model.

Returns

The name to set for the motion model. std::string variable. Variable passed by value.

Definition at line 2005 of file motionmodel.cpp.

15.26.3.18 vector< Body > & MotionModel::listBody ( )

Provides direct access to the list of Bodies referenced by the motion model.

Returns

Reference to vector of Body objects. Variable passed by reference.

See Also

Body

Definition at line 2023 of file motionmodel.cpp.

15.26.3.19 Body & MotionModel::listBody (int *bodIn*)

Direct access to an individual Body from the list of Bodies contained in the motion model.

Parameters

bodIn	Integer specifying which Body object to access in the list of Bodies.

Returns

Returns reference to the Body object specified by input bodIn.

See Also

listBody()

Definition at line 2029 of file motionmodel.cpp.

15.26.3.20 vector< int > & MotionModel::listCompCrossBod_hydro ( )

Records the index of the body object referenced by the cross body.

Records the index of the body object referenced by the cross body. Each body object contains a list of pointers for the cross-body objects. Each cross-body force has a pointer associated with it. This pointer points to another body object. This allows comparison between memory addresses of different body objects. However, when the body objects are copied over, the pointers are now pointing to different, invalid memory addresses. to eliminate this problem in the motion model, the model will record the position of the body object in the vector of body objects. This forms a vector. Each entry in the vector represents one cross-body force for the current body. The integer entry in the vector is the integer index of the body that the cross-body force is linked to.

Returns

Returns a vector of integers. Returned variable is passed by reference. Each entry in the vector represents one cross-body force for the current body. The integer entry in the vector is the integer index of the body that the cross-body force is linked to.

Definition at line 109 of file motionmodel.cpp.

15.26.3.21 int & MotionModel::listCompCrossBod_hydro (int *crossbodIn*)

Records the index of the body object referenced by the cross body.

Records the index of the body object referenced by the cross body. Each body object contains a list of pointers for the cross-body objects. Each cross-body force has a pointer associated with it. This pointer points to another body object. This allows comparison between memory addresses of different body objects. However, when the body objects are copied over, the pointers are now pointing to different, invalid memory addresses. to eliminate this problem in the motion model, the model will record the position of the body object in the vector of body objects. This forms a vector. Each entry in the vector represents one cross-body force for the current body. The integer entry in the vector is the integer index of the body that the cross-body force is linked to.

Parameters

crossbodIn	Integer parameter. Specified the index of which value to retrieve from the list of values for the CrossBod indices.

Returns

Returns an integer. Variable passed by reference. Returned integer is the index of the Body object referenced by the cross-body force located at the index specified by CrossBod. Example: CrossBod (index) -> (vector of cross body forces) -> Index of Body object that cross-body force points to.

Definition at line 115 of file motionmodel.cpp.

15.26.3.22 vector< int > & MotionModel::listCompCrossBod_user ( )

Records the index of the body object referenced by the cross body.

Records the index of the body object referenced by the cross body. Each body object contains a list of pointers for the cross-body objects. Each cross-body force has a pointer associated with it. This pointer points to another body object. This allows comparison between memory addresses of different body objects. However, when the body objects are copied over, the pointers are now pointing to different, invalid memory addresses. to eliminate this problem in the motion model, the model will record the position of the body object in the vector of body objects. This forms a vector. Each entry in the vector represents one cross-body force for the current body. The integer entry in the vector is the integer index of the body that the cross-body force is linked to.

Returns

Returns a vector of integers. Returned variable is passed by reference. Each entry in the vector represents one cross-body force for the current body. The integer entry in the vector is the integer index of the body that the cross-body force is linked to.

Definition at line 121 of file motionmodel.cpp.

15.26.3.23 int & MotionModel::listCompCrossBod_user (int *crossbodIn*)

Records the index of the body object referenced by the cross body.

Records the index of the body object referenced by the cross body. Each body object contains a list of pointers for the cross-body objects. Each cross-body force has a pointer associated with it. This pointer points to another body object. This allows comparison between memory addresses of different body objects. However, when the body objects are copied over, the pointers are now pointing to different, invalid memory addresses. to eliminate this problem in the motion model, the model will record the position of the body object in the vector of body objects. This forms a vector. Each entry in the vector represents one cross-body force for the current body. The integer entry in the vector is the integer index of the body that the cross-body force is linked to.

Parameters

crossbodIn	Integer parameter. Specified the index of which value to retrieve from the list of values for the CrossBod indices.

Returns

Returns an integer. Variable passed by reference. Returned integer is the index of the Body object referenced by the cross-body force located at the index specified by CrossBod. Example: CrossBod (index) -> (vector of cross body forces) -> Index of Body object that cross-body force points to.

Definition at line 127 of file motionmodel.cpp.

15.26.3.24 vector< Body > & MotionModel::listData ( )

Provides direct access to the list of Bodies used as data for the motion model.

Returns

Reference to the vector of Body objects used as data. Variable passed by reference.

See Also

Bodyy

Definition at line 2035 of file motionmodel.cpp.

15.26.3.25 Body & MotionModel::listData (int *dataIn*)

Direct access to an individual Body from the list of Data contained in the motion model.

Parameters

dataIn	Integer specifying which Body object to access from the list of Data.

Returns

Returns reference to the Body object specified by dataIn.

See Also

listData()

Definition at line 2041 of file motionmodel.cpp.

15.26.3.26 std::vector< EquationofMotion * > & MotionModel::listDataEquation ( )

Provides direct access to the list of equation of motion objects used in the motion model.

This is the same as the listEquation() function, but just under a different name.

Returns

Reference to the vector of EquationofMotion objects. Variable passed by reference. Each entry in the vector is a pointer to the relevent equation of motion object.

See Also

EquationofMotion
MotionModel::listEquation()

Definition at line 2059 of file motionmodel.cpp.

15.26.3.27 EquationofMotion & MotionModel::listDataEquation (int *eqIn*)

Direct access to an individual EquationofMotion object from the list of Equations contained in the motion model.

This function specifies the requested equation through the use of the equation's data index. Not its normal occurrence index in the vector. The data index is the number of the data this equation will retrieve, not its sequence in the vector that stores it.

Parameters

eqIn	Integer. The data index of the EquationofMotion object requested.

Returns

Returns reference to the EquationofMotion object specified by eqIn. Returned variable passed by reference.

Definition at line 2065 of file motionmodel.cpp.

15.26.3.28 vector< int > & MotionModel::listDataIndex ( )

Returns a vector containing all equation indices. This may be the same as the number of equations. However, if they are custom equations, they must avoid the first six indices, which are reserved for standard 6dof models. This means that the data index may not start at zero, which is why the data index vector is returned. It allows you to see for each entry in the slot, what the index is for that equation.

Returns

Returns a vector containing all the equation indices currently in use. Returned vector is passed by reference.

Definition at line 1920 of file motionmodel.cpp.

15.26.3.29 int & MotionModel::listDataIndex (unsigned int *index*)

Returns an enry from a vector containing all equation indices.

The requested entry is specified by the input variable index. The list of equation data indices may be the same as the number of equations. However, if they are custom equations, they must avoid the first six indices, which are reeserved for standard 6dof models. This means that the data index may not start at zero, which is why the entries of the data index vector are exposed for retrieval and manipulation. It allows you to see for each entry in the slot, which the data index is for that equation.

Parameters

index	Integer variable. Passed by value. Specified the index of which entry in the data index you want to see. If the requested index is beyond the current limits of the vectors, the vector is automatically resized, but never larger than the number of current equations. Each entry in the index represents an equation.

Returns

Returns an integer variable. Variable passed by value. The returned variable is an entry from the vector of all equation data indices currently in use.

Definition at line 1927 of file motionmodel.cpp.

15.26.3.30 vector< EquationofMotion * > & MotionModel::listEquation ( )

Provides direct access to the list of equation of motion objects used in the motion model.

Returns

Reference to the vector of EquationofMotion objects. Variable passed by reference. Each entry in the vector is a pointer to the relevent equation of motion object.

See Also

EquationofMotion

Definition at line 2047 of file motionmodel.cpp.

15.26.3.31 EquationofMotion & MotionModel::listEquation (int *eqln*)

Direct access to an individual EquationofMotion object from the list of Equations contained in the motion model.

Parameters

eqln	Integer specifying which EquationofMotion object to access from the list of Equations

Returns

Returns reference to the EquationofMotion object specified by eqln.

Definition at line 2053 of file motionmodel.cpp.

15.26.3.32 int MotionModel::MaxDataIndex ( )

Returns the maximum number of the data index.

Returns the maximum number of the data index. This may be the same as the number of equations. Very few equations may be used. However, if they are custom equations, they must avoid the first six indices, which are reserved for standard 6dof models.

Returns

Returns integer number representing the maximum number of the data index found from all equations.

Definition at line 1977 of file motionmodel.cpp.

15.26.3.33 int MotionModel::numEquations ( )

Reports the number of equations used in the motion model.

Reports the number of equations used in the motion model. This lets the matBody object know how many equations to prepare for.

Returns

Returns the number of equations used in the motion model.

Definition at line 1913 of file motionmodel.cpp.

15.26.3.34 void MotionModel::Reset ( )

Resets the class data to have all input coefficients. Any evaluation after a reset will produce a value of zero. Force coefficients will be zero and force values will be zero.

Definition at line 133 of file motionmodel.cpp.

15.26.3.35 void MotionModel::setBody (int *bod*)

Sets the index for the body that all calculations are based on.

Parameters

Integer	input specifying the number of the body to use. Integer corresponds to the sequence of bodies in the vector supplied with the body.

Definition at line 76 of file motionmodel.cpp.

15.26.3.36 void MotionModel::setDescription (std::string *DescIn*)

Description for the motion model.

Description for the motion model. Used by the user to provide a more extensive description of the motion model. Used purely for user information. Not used for model identification.

Parameters

DescIn	std::string. The description for the motion model. Variable passed by value.

Definition at line 2011 of file motionmodel.cpp.

15.26.3.37 void MotionModel::setFreq (double *freq*)

Sets the current operating frequency for the function. Only necessary when calculating true forces and using derivatives defined in the motion model. Otherwise, you can safely ignore this function.

Parameters

Double	precision value that sets the current wave frequency value.

Definition at line 191 of file motionmodel.cpp.

15.26.3.38 void MotionModel::setlistBody (std::vector< Body > & *listBodIn*)

Inputs the list of body data.

Parameters

listBodIn	The vector of body objects to input.

Definition at line 69 of file motionmodel.cpp.

15.26.3.39 void MotionModel::setName (std::string *nameIn*)

Name for the motion model.

Name for the motion model. Used by the user to identify the motion model.

Parameters

nameIn	The name to set for the motion model. std::string variable. Variable passed by value.

Definition at line 1999 of file motionmodel.cpp.

15.26.3.40 void MotionModel::useForceActive_hydro (unsigned int *force,* unsigned int *eqn*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.
eqn	Integer specifying which equation to use in the selected force. Based on data index.

Definition at line 333 of file motionmodel.cpp.

15.26.3.41 void MotionModel::useForceActive_hydro (unsigned int *force*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. With only the force number specified, all equations are used as coefficients.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.

Definition at line 385 of file motionmodel.cpp.

15.26.3.42 void MotionModel::useForceActive_hydro ( )

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. All forces and all coefficients are used.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.

Definition at line 446 of file motionmodel.cpp.

15.26.3.43 void MotionModel::useForceActive_user (unsigned int *force,* unsigned int *eqn*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.
eqn	Integer specifying which equation to use in the selected force. Based on data index.

Definition at line 205 of file motionmodel.cpp.

15.26.3.44 void MotionModel::useForceActive_user (unsigned int *force*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. With only the force number specified, all equations are used as coefficients.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.

Definition at line 257 of file motionmodel.cpp.

15.26.3.45 void MotionModel::useForceActive_user ( )

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. All forces and all coefficients are used.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.

Definition at line 320 of file motionmodel.cpp.

15.26.3.46 void MotionModel::useForceCross_hydro (unsigned int *force,* unsigned int *ord,* unsigned int *eqn,* unsigned int *var*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.
ord	Integer specifying which order of derviative to use for the selected force.
eqn	Integer specifying which equation to use in the selected force. Based on data index.
var	Integer specifying which variable to use from the selected equation. Based on data index.

Definition at line 1270 of file motionmodel.cpp.

15.26.3.47 void MotionModel::useForceCross_hydro (unsigned int *force,* unsigned int *ord,* unsigned int *eqn*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables within the specified equation, derivative, and force.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.
ord	Integer specifying which order of derviative to use for the selected force.
eqn	Integer specifying which equation to use in the selected force. Based on data index.

Definition at line 1385 of file motionmodel.cpp.

15.26.3.48 void MotionModel::useForceCross_hydro (unsigned int *force,* unsigned int *ord*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables and all equations within the specified derivative and force.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.
ord	Integer specifying which order of derviative to use for the selected force.

Definition at line 1435 of file motionmodel.cpp.

15.26.3.49 void MotionModel::useForceCross_hydro (unsigned int *force*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables and all equations within all derivatives within the specified force

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.

Definition at line 1474 of file motionmodel.cpp.

15.26.3.50 void MotionModel::useForceCross_hydro ( )

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables and all equations within all derivatives within all forces available.

Definition at line 1505 of file motionmodel.cpp.

15.26.3.51 void MotionModel::useForceCross_user (unsigned int *force*, unsigned int *ord*, unsigned int *eqn*, unsigned int *var*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.
ord	Integer specifying which order of derviative to use for the selected force.
eqn	Integer specifying which equation to use in the selected force. Based on data index.
var	Integer specifying which variable to use from the selected equation. Based on data index.

Definition at line 1019 of file motionmodel.cpp.

15.26.3.52 void MotionModel::useForceCross_user (unsigned int *force*, unsigned int *ord*, unsigned int *eqn*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables within the specified equation, derivative, and force.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.
ord	Integer specifying which order of derviative to use for the selected force.
eqn	Integer specifying which equation to use in the selected force. Based on data index.

Definition at line 1136 of file motionmodel.cpp.

15.26.3.53 void MotionModel::useForceCross_user (unsigned int *force*, unsigned int *ord*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative.

An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables and all equations within the specified derivative and force.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.
ord	Integer specifying which order of derviative to use for the selected force.

Definition at line 1186 of file motionmodel.cpp.

15.26.3.54 void MotionModel::useForceCross_user (unsigned int *force*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other use-Force methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables and all equations within all derivatives within the specified force

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.

Definition at line 1225 of file motionmodel.cpp.

15.26.3.55 void MotionModel::useForceCross_user ( )

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other use-Force methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables and all equations within all derivatives within all forces available.

Definition at line 1256 of file motionmodel.cpp.

15.26.3.56 void MotionModel::useForceMass (unsigned int *eqn*, unsigned int *var*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other use-Force methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient.

Parameters

eqn	Integer specifying which equation to use in the selected force. Based on data index.
var	Integer specifying which variable to use from the selected equation. Based on data index.

Definition at line 1519 of file motionmodel.cpp.

15.26.3.57 void MotionModel::useForceMass (unsigned int *eqn*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This implementation will copy over all variables for the specified equation.

Parameters

eqn	Integer specifying which equation to use in the selected force. Based on data index.

Definition at line 1567 of file motionmodel.cpp.

15.26.3.58 void MotionModel::useForceMass ( )

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This implementation will copy over all variables for all equations.

Definition at line 1606 of file motionmodel.cpp.

15.26.3.59 void MotionModel::useForceReact_hydro (unsigned int *force*, unsigned int *ord*, unsigned int *eqn*, unsigned int *var*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.
ord	Integer specifying which order of derviative to use for the selected force.
eqn	Integer specifying which equation to use in the selected force. Based on data index.
var	Integer specifying which variable to use from the selected equation. Based on data index.

Definition at line 732 of file motionmodel.cpp.

15.26.3.60 void MotionModel::useForceReact_hydro (unsigned int *force*, unsigned int *ord*, unsigned int *eqn*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables within the specified equation, derivative, and force.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.
ord	Integer specifying which order of derviative to use for the selected force.
eqn	Integer specifying which equation to use in the selected force. Based on data index.

Definition at line 830 of file motionmodel.cpp.

15.26.3.61 void MotionModel::useForceReact_hydro (unsigned int *force,* unsigned int *ord*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other use-Force methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables and all equations within the specified derivative and force.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.
ord	Integer specifying which order of derviative to use for the selected force.

Definition at line 935 of file motionmodel.cpp.

15.26.3.62 void MotionModel::useForceReact_hydro (unsigned int *force*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other use-Force methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables and all equations within all derivatives within the specified force

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.

Definition at line 974 of file motionmodel.cpp.

15.26.3.63 void MotionModel::useForceReact_hydro ( )

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other use-Force methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables and all equations within all derivatives within all forces available.

Definition at line 1005 of file motionmodel.cpp.

15.26.3.64 void MotionModel::useForceReact_user (unsigned int *force*, unsigned int *ord*, unsigned int *eqn*, unsigned int *var*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.
ord	Integer specifying which order of derviative to use for the selected force.
eqn	Integer specifying which equation to use in the selected force. Based on data index.
var	Integer specifying which variable to use from the selected equation. Based on data index.

Definition at line 459 of file motionmodel.cpp.

15.26.3.65 void MotionModel::useForceReact_user (unsigned int *force*, unsigned int *ord*, unsigned int *eqn*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables within the specified equation, derivative, and force.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.
ord	Integer specifying which order of derviative to use for the selected force.
eqn	Integer specifying which equation to use in the selected force. Based on data index.

Definition at line 555 of file motionmodel.cpp.

15.26.3.66 void MotionModel::useForceReact_user (unsigned int *force*, unsigned int *ord*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables and all equations within the specified derivative and force.

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.
ord	Integer specifying which order of derviative to use for the selected force.

Definition at line 648 of file motionmodel.cpp.

15.26.3.67 void MotionModel::useForceReact_user (unsigned int *force*)

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables and all equations within all derivatives within the specified force

Parameters

force	Integer specifying which force to use in the set of forces for the given force type.

Definition at line 687 of file motionmodel.cpp.

15.26.3.68 void MotionModel::useForceReact_user ( )

Passes information to the object to use input coefficients from the entry specified.

Passes information to the object to use input coefficients from the entry specified. Limits inputs to only the force object type specified by the method. Calls to useForce methods are cumulative. Sucessive calls to different entries in the same force sequence will add their coefficients to the sets for evaluation. Can be combined with other useForce methods. Multiple calls to the same useForce method with the same index coordinates are not cumulative. An input coefficient can either be on or off, not multiple instances of the exact same coefficient. This method uses all coefficients for all variables and all equations within all derivatives within all forces available.

Definition at line 718 of file motionmodel.cpp.

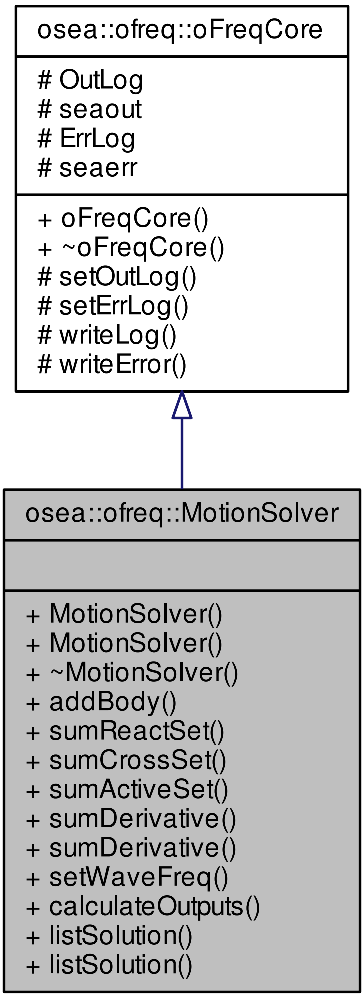
The documentation for this class was generated from the following files:

- bin/ofreq/motion_model/motionmodel.h
- bin/ofreq/motion_model/motionmodel.cpp

15.27 osea::ofreq::MotionSolver Class Reference

```
#include <motionsolver.h>
```

Inheritance diagram for osea::ofreq::MotionSolver:



Public Member Functions

- MotionSolver ()

  The default constructor.
- MotionSolver (std::vector< matBody > listBodIn)

  Constructor. Initialize with objects already added.
- ∼MotionSolver ()
- void addBody (matBody bodIn)

  Add a body to the motion solver set.
- matForceReact ∗ sumReactSet (std::vector< matForceReact > listForces)

  Sum Reactive forces for each set.
- std::vector< matForceCross ∗ > sumCrossSet (std::vector< matForceCross > listForces)

  Sum cross-body forces for each set.
- arma::cx_mat ∗ sumActiveSet (std::vector< matForceActive > listForces)

Sum active forces for each set.
- arma::cx_mat ∗ sumDerivative (matForceReact ∗forceIn)

  Sums all derivatives for the reactive force object entered.
- std::vector< arma::cx_mat ∗ > sumDerivative (std::vector< matForceCross ∗ > forceIn)

  Sums all derivatives for the cross-body force objects entered.
- void setWaveFreq (double freqIn)

  Sets the current wave frequency.
- void calculateOutputs ()

  Calculate the Solution.
- std::vector< arma::cx_mat > & listSolution ()

  Get the solution for the solved equation.
- arma::cx_mat & listSolution (unsigned int bod)

  Get the solution for the solved equation, for a single body.

Additional Inherited Members

15.27.1 Detailed Description

This class holds data for the motion solver and performs calculations on all of the data to get the solution matrix. The motion solver performs a series of operations. 1.) Sum each force object for all forces within a set. 2.) Sum each force object for all the derivatives defined within each force. Applied frequency dependence. 3.) Sum user reactive forces, hydro reactive forces, and body mass into a single object for each body. 4.) Sum user cross-body forces, hydro cross-body forces into a single object for each body. 5.) Sum user active forces and hydro active forces into a single object for each body. 6.) Assemble reactive forces and cross-body forces into a single global response matrix. ([A]) 7.) Assemble active forces into a single global active force matrix. ([F]) 8.) Solve the linear system system of equations formed by the equation [A] ∗ [x] = [F] 9.) Redistribute the solution back to each body object.

Definition at line 97 of file motionsolver.h.

15.27.2 Constructor & Destructor Documentation

15.27.2.1 MotionSolver::MotionSolver ( )

The default constructor.

Definition at line 37 of file motionsolver.cpp.

15.27.2.2 osea::ofreq::MotionSolver::MotionSolver (std::vector< **matBody** > *listBodIn*)

Constructor. Initialize with objects already added.

This constructor combines creation of the class with adding the body objects to the class at the same time. The list of matBody objects is added to the set of bodies that the motion solver solves. Bodies inputs are passed by value, and not by reference.

Parameters

listBodIn	The vector list of bodies to add to the motion solver object.

15.27.2.3 MotionSolver::∼MotionSolver ( )

The default destructor, nothing happens here.

Definition at line 51 of file motionsolver.cpp.

15.27.3 Member Function Documentation

15.27.3.1 void MotionSolver::addBody (matBody *bodIn*)

Add a body to the motion solver set.

Add a body to the motion solver set. After initialization, this is how the motion solver gets the correct data to perform math operations on.

Parameters

The	matBody object to add the motion solver set. Body inputs are added passed by value, and not by reference.

Definition at line 56 of file motionsolver.cpp.

15.27.3.2 void MotionSolver::calculateOutputs ( )

Calculate the Solution.

Definition at line 260 of file motionsolver.cpp.

15.27.3.3 vector< cx_mat > & MotionSolver::listSolution ( )

Get the solution for the solved equation.

Get the solution for the solved equation. Returns the full vector of complex matrices. Each element in the vector is a solution matrix specific to the body. The vector is ordered in the same sequence that the bodies were added to the motionsolver object. This is the vector of solutions for each Body object. It applies to a single wave frequency and single wave direction.

Returns

Returns the full vector of complex matrices. Each element in the vector is a solution matrix specific to the body. The vector is ordered in the same sequence that the bodies were added to the motionsolver object. Returned variable passed by reference.

Definition at line 493 of file motionsolver.cpp.

15.27.3.4 cx_mat & MotionSolver::listSolution (unsigned int *bod*)

Get the solution for the solved equation, for a single body.

Returns the solution for solved equations for a single Body object. This is a matrix of solutions for equations of motion. It applies to a single Body, single wave frequency, single wave direction. The Bodies are ordered in the same sequence in which they were added to the motionsolver object.

Parameters

bod	Integer. Specifies which Body object the solution should be retrieved for. The Bodies are ordered in the same sequence in which they were added to the motionsolver object.

Returns

Returns a matrix of complex doubles. This matrix contains the solution of motions for the Body specified. Returned variable passed by reference.

See Also

Body

Definition at line 499 of file motionsolver.cpp.

15.27.3.5 void MotionSolver::setWaveFreq (double *freqIn*)

Sets the current wave frequency.

Sets the current wave frequency. This is used in calculating the summations and must be set before calling the solve method.

Parameters

freqIn	The input wave frequency. A double precision floating point value. Used when summing derivatives.

Definition at line 253 of file motionsolver.cpp.

15.27.3.6 cx_mat $*$ MotionSolver::sumActiveSet (std::vector$<$ matForceActive $>$ *listForces*)

Sum active forces for each set.

Takes the input vector and sums all force objects together to create an aggregate force that is the total of all force objects supplied in the input vector (listForces). If the input vector is empty, the function returns a NULL pointer.

Parameters

listForces	Vector of matForceActive objects. Vector can be unlimited size. Each entry in the vector is one of the active forces to be added into the total aggregate active force.

Returns

The Sum of active force matrix. Variable is returned as pointer.

Definition at line 128 of file motionsolver.cpp.

15.27.3.7 vector$<$ matForceCross $*$ $>$ MotionSolver::sumCrossSet (std::vector$<$ matForceCross $>$ *listForces*)

Sum cross-body forces for each set.

This gets handled a little differently from reactive forces, as the linked body for the force depends on whether two objects are summed together. Output from this function is a vector of cross-body forces. Each entry in the vector contains a cross-body force object. If the input list of forces (listForces) is empty, the function returns a NULL pointer.

Parameters

CrossBodMat	The vector of cross-body force matrices.

Returns

A vector of complex matrices, with each entry in the vectors representing a cross-body force linked to a specific body. Returned variable is a pointer.

Definition at line 87 of file motionsolver.cpp.

15.27.3.8 cx_mat ∗ MotionSolver::sumDerivative (matForceReact ∗ *forceIn*)

Sums all derivatives for the reactive force object entered.

Matrix force objects normally store a separate matrix of coefficients for each derivative. However, to solve for object motions at a given frequency, it is necessary to combine the various derivatives into a single matrix. The function uses a formula to combine the various derivative formulas into a single output matrix. The output matrix is only valid for the specific frequency set at the time of calling this function. If the input object is a NULL pointer, the function also returns a NULL pointer.

Parameters

forceIn	The reactive force matrix. Variable is a pointer to a matForceReact object. Variable passed by value.

Returns

Single matrix that is the derivative sum of each matrix for each derivative contained within the input object. Returned variable is a pointer. Returned pointer is set to NULL if input pointer is NULL.

Definition at line 164 of file motionsolver.cpp.

15.27.3.9 vector< cx_mat ∗ > MotionSolver::sumDerivative (std::vector< matForceCross ∗ > *forceIn*)

Sums all derivatives for the cross-body force objects entered.

Matrix force objects normally store a separate matrix of coefficients for each derivative. However, to solve for object motions at a given frequency, it is necessary to combine the various derivatives into a single matrix. The function uses a formula to combine the various derivative formulas into a single output matrix. The output matrix is only valid for the specific frequency set at the time of calling this function. If the input object is a NULL pointer, the function also returns a NULL pointer. For the cross-body forces, the function expects a vector of cross-body forces. These are all for a single Body object. Each matForceCross object in the vector represents a cross-body force that depends on the motions of another body. So for N bodies, it is possible for the vector to contain up to N - 1 matForceCross objects.

Parameters

forceIn	The cross-body force matrix. Variable is a vector of pointers to matForceCross objects. Variable passed by value.

Returns

Returns vector of single matrices. Each matrix in the vector is the derivative sum of each matrix for each derivative contained within the input matForceCross object. Returned variable is a pointer to a vector of single matrices. Returned pointer is set to NULL if input pointer is NULL.

Definition at line 207 of file motionsolver.cpp.

15.27.3.10 matForceReact ∗ MotionSolver::sumReactSet (std::vector< matForceReact > *listForces*)

Sum Reactive forces for each set.

Sum Reactive forces for each set. This iterates through each reactive force in a set and adds the forces together. It respects derivatives in the summation. If the input list of forces is empty (listForces), the function returns a NULL pointer.

Parameters

listForces	The list of reactive forces associated with each body. This list may be anything from zero to infinite number of entries.

Returns

The Sum of reactive force matrices. Returned variable is a pointer.

Definition at line 62 of file motionsolver.cpp.

The documentation for this class was generated from the following files:

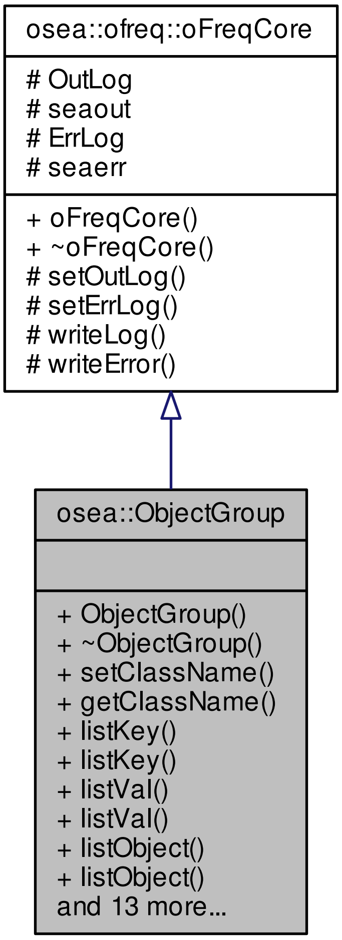
- bin/ofreq/motion_solver/motionsolver.h
- bin/ofreq/motion_solver/motionsolver.cpp

15.28 osea::ObjectGroup Class Reference

The ObjectGroup class contains groupings of object definitions captured from an input file. It is a data container to hold the segmented input file for interpretation. The container contains three things: 1.) Object class name (as specified by input file) 2.) Vector of keyword names 3.) Vector of keyword values. Each entry in the vector of values is also a vector. This allows the definition of lists. A list will be as long as it needs to be for specification of all values in the list. The index of the value is specified by its position in the vector list. The value is the entry.

```
#include <objectgroup.h>
```

Inheritance diagram for osea::ObjectGroup:



Public Member Functions

- ObjectGroup ()

  Default constructor.
- ∼ObjectGroup ()

  Default destructor.
- void setClassName (std::string input)

  Sets the class name, as specified by the input file.
- std::string getClassName ()

  Gets the class name, as specified by the input file.
- vecKeyword & listKey ()

  Provides direct access to thelist of key words.
- std::string & listKey (int index)

  Provides direct access to an individual item in the list of key words.
- vecValue & listVal ()

  Provides direct access to the list of key values.
- std::vector< std::string > & listVal (int index)

  Provides direct access to an individual item in the list of key values.
- std::vector< ObjectGroup ∗ > & listObject ()

  Provides direct access to the list of sub object definitions.
- ObjectGroup ∗ listObject (int index)

  Provides direct access to an individual item in the list of ObjectGroup objects.
- void addSubObject (ObjectGroup objIn)

  Adds an object to the list of ObjectGroup objects defined under this existing ObjectGroup.
- void addSubObject ()

  Adds an object to the list of ObjectGroup objects defined under this existing ObjectGroup.
- void addKeySet (std::string key, std::string val)

  Adds a new keyword-value set to the ObjectGroup definition.
- void addKeySet (std::string key, std::vector< std::string > val)

  Overloaded function that adds a new keyword-value set to the ObjectGroup definition. This version of the method is used for passing lists in the keyword-value set definition.
- void addKeyWord (std::string word, int index=-1)

  Adds a key word to the list of key words. Only adds the word. Not the value.
- void addKeyVal (std::string val, int index=-1)

  addKeyVal Adds a key value to the list of key values. Only adds the value. Not the word.
- void addKeyVal (std::vector< std::string > val, int index=-1)

  addKeyVal Adds a key value to the list of key values. Only adds the value. Not the word. This is a function overload that allows for adding vector lists of key values.
- std::string getKey (int index)

  Gets a key word as specified by the index.
- std::vector< std::string > getVal (int index)

  Gets the key value as specified by the index.
- void setVersion (std::string input)

  Sets the version property of the object.
- std::string getVersion ()

  Gets the version property of the object.
- void setFormat (std::string input)

  Sets the format property of the object.
- std::string getFormat ()

  Gets the format property of the object.

Additional Inherited Members

15.28.1 Detailed Description

The ObjectGroup class contains groupings of object definitions captured from an input file. It is a data container to hold the segmented input file for interpretation. The container contains three things: 1.) Object class name (as specified by input file) 2.) Vector of keyword names 3.) Vector of keyword values. Each entry in the vector of values is also a vector. This allows the definition of lists. A list will be as long as it needs to be for specification of all values in the list. The index of the value is specified by its position in the vector list. The value is the entry.

Definition at line 97 of file objectgroup.h.

15.28.2 Constructor & Destructor Documentation

15.28.2.1 ObjectGroup::ObjectGroup ( )

Default constructor.

Definition at line 36 of file objectgroup.cpp.

15.28.2.2 ObjectGroup::∼ObjectGroup ( )

Default destructor.

Definition at line 41 of file objectgroup.cpp.

15.28.3 Member Function Documentation

15.28.3.1 void osea::ObjectGroup::addKeySet (std::string *key*, std::string *val*)

Adds a new keyword-value set to the ObjectGroup definition.

Parameters

key	The key word to input. Variable passed by value.
val	The key value to input. Can only be a single value. Variable passed by value.

15.28.3.2 void osea::ObjectGroup::addKeySet (std::string *key*, std::vector< std::string > *val*)

Overloaded function that adds a new keyword-value set to the ObjectGroup definition. This version of the method is used for passing lists in the keyword-value set definition.

Parameters

key	The key word to input. Variable passed by value.
val	The list of key values to input. Vector list. The index of each entry in the vector represents its index in the input file. The value of the entry represents the value in the input file. Blank entries are allowed.

15.28.3.3 void osea::ObjectGroup::addKeyVal (std::string *val*, int *index* = -1)

addKeyVal Adds a key value to the list of key values. Only adds the value. Not the word.

Parameters

val	std::string input. The key value to add to the set. Variable passed by value. Assumes a single key value.
index	The index of where to add the key value to the set. If left at the default setting, the value is automatically added to the end of the current list. Variable passed by value.

15.28.3.4 void osea::ObjectGroup::addKeyVal (std::vector< std::string > *val*, int *index* = -1)

addKeyVal Adds a key value to the list of key values. Only adds the value. Not the word. This is a function overload that allows for adding vector lists of key values.

Parameters

val	Vector of std::string inputs. The vector of values to add to the key. Variable passed by value. Assumes a vector of key values.
index	The index of where to add the key value to the set. If left at the default setting, the value is automatically added to the end of the current list. Variable passed by value.

15.28.3.5 void ObjectGroup::addKeyWord (std::string *word*, int *index* = -1)

Adds a key word to the list of key words. Only adds the word. Not the value.

Parameters

word	std::string input. The key word to add to the set. Variable passed by value.
index	The index of where to add the key word to the set. If left at the default setting, the word is automatically added to the end of the current list. Variable passed by value.

Definition at line 127 of file objectgroup.cpp.

15.28.3.6 void ObjectGroup::addSubObject (ObjectGroup *objIn*)

Adds an object to the list of ObjectGroup objects defined under this existing ObjectGroup.

Parameters

objIn	An ObjectGroup object. Variable passed by value.

Definition at line 95 of file objectgroup.cpp.

15.28.3.7 void ObjectGroup::addSubObject ( )

Adds an object to the list of ObjectGroup objects defined under this existing ObjectGroup.

Definition at line 103 of file objectgroup.cpp.

15.28.3.8 string ObjectGroup::getClassName ( )

Gets the class name, as specified by the input file.

Returns

Returns std::string that represents the name of the class, as specified by the input file. Variable passed by value.

Definition at line 53 of file objectgroup.cpp.

15.28.3.9 string ObjectGroup::getFormat ( )

Gets the format property of the object.

Returns

Returns std::string output. The format property of the object. Variable passed by value.

Definition at line 196 of file objectgroup.cpp.

15.28.3.10 string ObjectGroup::getKey (int *index*)

Gets a key word as specified by the index.

Parameters

index	Integer. Specifies the index of which key word to grab.

Returns

Returns a std::string object that represents the key word. Variable passed by value.

Definition at line 166 of file objectgroup.cpp.

15.28.3.11 vector< string > ObjectGroup::getVal (int *index*)

Gets the key value as specified by the index.

Parameters

index	Integer. Specifies the index of which key word to grab.

Returns

Returns a vector of std::string objects that represent the key value. For cases of a single key value, the vector will only be one entry long. For cases of lists, the vector has an unlimited length. Variable passed by value.

Definition at line 172 of file objectgroup.cpp.

15.28.3.12 string ObjectGroup::getVersion ( )

Gets the version property of the object.

Returns

Returns std::string output. The version property of the object. Variable passed by value.

Definition at line 184 of file objectgroup.cpp.

15.28.3.13 vecKeyword & ObjectGroup::listKey ( )

Provides direct access to thelist of key words.

Returns

Returns a reference to the list of key words. Variable passed by reference.

Definition at line 59 of file objectgroup.cpp.

15.28.3.14 string & ObjectGroup::listKey (int *index*)

Provides direct access to an individual item in the list of key words.

Returns the key word specified by the index.

Parameters

index	Integer. The index of the key word to retrieve.

Returns

std::string. The key word specified by index. Returned variable passed by reference.

Definition at line 65 of file objectgroup.cpp.

15.28.3.15 vector< ObjectGroup * > & ObjectGroup::listObject ( )

Provides direct access to the list of sub object definitions.

Returns

Returns a reference to the list of objects. Variable passed by reference. Returned variable is a vector of pointers to ObjectGroup objects.

Definition at line 83 of file objectgroup.cpp.

15.28.3.16 ObjectGroup * ObjectGroup::listObject (int *index*)

Provides direct access to an individual item in the list of ObjectGroup objects.

Returns a pointer to the ObjectGroup object specified by the index.

Parameters

index	Integer. The index of the ObjectGroup object to retrieve.

Returns

Returns a pointer to an ObjectGroup object. The object specified by the index. Returned pointer is passed by value.

Definition at line 89 of file objectgroup.cpp.

15.28.3.17 vecValue & ObjectGroup::listVal ( )

Provides direct access to the list of key values.

Returns

Returns a reference to the list of key values. Variable passed by reference.

Definition at line 71 of file objectgroup.cpp.

15.28.3.18 vector< string > & ObjectGroup::listVal (int *index*)

Provides direct access to an individual item in the list of key values.

Returns the key value specified by the index.

Parameters

index	Integer. THe index of the key value to retrieve.

Returns

Vector of strings. The key value specified by the index. Returned variable passed by reference.

Definition at line 77 of file objectgroup.cpp.

15.28.3.19 void ObjectGroup::setClassName (std::string *input*)

Sets the class name, as specified by the input file.

Parameters

input	std::string. The name of the class, as specified by the input file. Variable passed by value.

Definition at line 47 of file objectgroup.cpp.

15.28.3.20 void ObjectGroup::setFormat (std::string *input*)

Sets the format property of the object.

Parameters

input	std::string input. The format property of the object. Variable passed by value.

Definition at line 190 of file objectgroup.cpp.

15.28.3.21 void ObjectGroup::setVersion (std::string *input*)

Sets the version property of the object.

Parameters

input	std::string input. The version property of the object. Variable passed by value.

Definition at line 178 of file objectgroup.cpp.

The documentation for this class was generated from the following files:

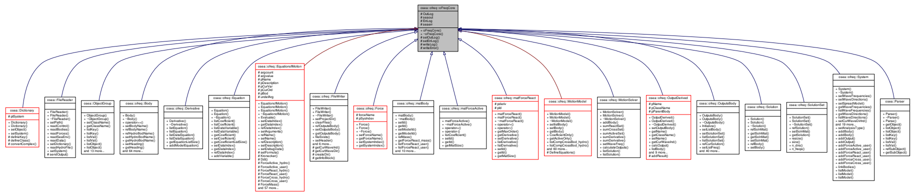
- bin/ofreq/file_reader/objectgroup.h
- bin/ofreq/file_reader/objectgroup.cpp

15.29 osea::ofreq::oFreqCore Class Reference

The core oFreq class. All oFreq classes inherit from this class.

```
#include <ofreqcore.h>
```

Inheritance diagram for osea::ofreq::oFreqCore:



Public Member Functions

- oFreqCore ()
- virtual ∼oFreqCore ()

Protected Member Functions

- void setOutLog (std::string dirIn)

  Opens the output log file and prepares it for writing.
- void setErrLog (std::string dirIn)

  Opens the error log file and prepares it for writing.
- void writeLog (std::string mesIn)

  Writes output of a log message to the log file. Adds in a date and time stamp to the log.
- void writeError (std::string mesIn)

  Writes output of an error message to the error file. Adds in a date, time, and class name to the log.

Static Protected Attributes

- static std::ofstream OutLog

  The log file for an oFreq run. Records normal actions for the program. Informs user of regular program developments. Through inheritence of the oFreqCore class, this object is available to every object in the oFreq application. Also provided with a second name of seaout.
- static std::ofstream & seaout = OutLog
- static std::ofstream ErrLog

  The error log file for an oFreq run. Records any errors or warnings for the program. Informs user of errors and warnings and where they occurred. Through inheritence of the oFreqCore class, this object is available for every object in the oFreq application. Also provided with a second name of seaerr.
- static std::ofstream & seaerr = ErrLog

15.29.1 Detailed Description

The core oFreq class. All oFreq classes inherit from this class.

Core oFreq class. All oFreq classes inherit from this class. Includes definition for anything fundamental and common to the entire program. Major items are any code used for application debugging. Also includes some objects to give everything access to log and error files for the program.

Definition at line 91 of file ofreqcore.h.

15.29.2 Constructor & Destructor Documentation

15.29.2.1 oFreqCore::oFreqCore ( )

Default constructor. Nothing happens here.

Definition at line 52 of file ofreqcore.cpp.

15.29.2.2 oFreqCore::∼oFreqCore ( ) [virtual]

Default destructor. Nothing happens here.

Definition at line 58 of file ofreqcore.cpp.

15.29.3 Member Function Documentation

15.29.3.1 void oFreqCore::setErrLog (std::string *dirIn*) [protected]

Opens the error log file and prepares it for writing.

Parameters

dirIn	std::string parameter. Designates the directory path to use for writing the error file.

Definition at line 91 of file ofreqcore.cpp.

15.29.3.2 void oFreqCore::setOutLog (std::string *dirIn*) [protected]

Opens the output log file and prepares it for writing.

Parameters

dirIn	std::string parameter. Designates the directory path to use for writing the log file.

Definition at line 75 of file ofreqcore.cpp.

15.29.3.3 void oFreqCore::writeError (std::string *mesIn*) [protected]

Writes output of an error message to the error file. Adds in a date, time, and class name to the log.

Parameters

mesIn	std::string variable. The message to write to the log file.

Definition at line 121 of file ofreqcore.cpp.

15.29.3.4 void oFreqCore::writeLog (std::string *mesIn*) [protected]

Writes output of a log message to the log file. Adds in a date and time stamp to the log.

Parameters

mesIn	std::string variable. The message to write to the log file.

Definition at line 107 of file ofreqcore.cpp.

15.29.4 Member Data Documentation

15.29.4.1 std::ofstream oFreqCore::ErrLog [static], [protected]

The error log file for an oFreq run. Records any errors or warnings for the program. Informs user of errors and warnings and where they occurred. Through inheritence of the oFreqCore class, this object is available for every object in the oFreq application. Also provided with a second name of seaerr.

Definition at line 132 of file ofreqcore.h.

15.29.4.2 std::ofstream oFreqCore::OutLog [static], [protected]

The log file for an oFreq run. Records normal actions for the program. Informs user of regular program developments. Through inheritence of the oFreqCore class, this object is available to every object in the oFreq application. Also provided with a second name of seaout.

Definition at line 123 of file ofreqcore.h.

15.29.4.3 std::ofstream & oFreqCore::seaerr = ErrLog [static], [protected]

Definition at line 133 of file ofreqcore.h.

15.29.4.4 std::ofstream & oFreqCore::seaout = OutLog [static], [protected]

Definition at line 124 of file ofreqcore.h.

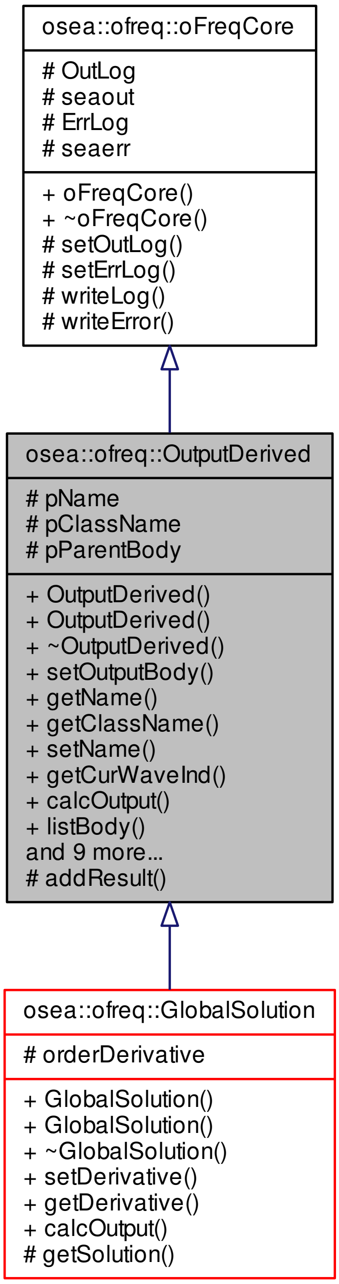
The documentation for this class was generated from the following files:

- bin/ofreq/global_objects/ofreqcore.h
- bin/ofreq/global_objects/ofreqcore.cpp

15.30 osea::ofreq::OutputDerived Class Reference

```
#include <outputderived.h>
```

Inheritance diagram for osea::ofreq::OutputDerived:



Public Member Functions

- OutputDerived ()

    Default constructor. Nothing happens here.
- OutputDerived (OutputsBody ∗input)

    Constructor that also sets the pointer to the OutputsBody object which contains the OutputDerived object.
- ∼OutputDerived ()

- void setOutputBody (OutputsBody ∗input)

  Sets the pointer to the OutputsBody object which contains this OutputDerived object.
- std::string getName ()

  Returns the name of the OutputDerived object. This is a name set by the user to idenfity the object. If multiple objects of the same type are created, the name can also distinguish between the various objects.
- std::string getClassName ()

  Returns the name of the class used to create the OutputDerived object. This is the name set to identify the type of output object. If multiple objects of the same type are created, the class name will be the same for all objects.
- void setName (std::string nameIn)

  Sets the name of the OutputDerived object. This is a name set by the user to idenfity the object. If multiple objects of the same type are created, the name can also distinguish between the various objects.
- int getCurWaveInd ()

  Gets the index of the current wave direction used by the OutputDerived object. Some child classes will need this to determine proper calculation of output. This differs from the getCurWaveDir() method in that the other method returns the actual value of the angle. This returns the index of the current angle in the list of wave directions.
- virtual int calcOutput (int freqIn=-1)=0

  Pure virtual member. Calculates the output from the OutputDerived object.
- std::vector< Body > & listBody ()

  Returns the list of body objects. Only a c++ reference to the list of body objects.
- Body & listBody (int bodIn)

  Returns a reference to an individual Body in the list of Body objects.
- std::vector< SolutionSet > & listSolutionSet ()

  Returns a vector of SolutionSet objects.
- SolutionSet & listSolutionSet (int index)

  Returns a vector of SolutionSet objects.
- std::vector< double > & listFreq ()

  Gets the list of frequencies. Frequencies are in radians per second.
- double & listFreq (int index)

  Returns individual frequency from the list of wave frequencies.
- std::vector< double > & listWaveDir ()

  Gets the list of wave directions. Wave directions are measured in radians. True North is zero, with positive angles going counter-clockwise.
- double & listWaveDir (int index)

  Returns individual wave direction from the list of wave directions.
- double getCurWaveDir ()

  Gets the current wave direction. Output is the actual value for the current wave direction, in units of radians.
- int getCurBodyIndex ()

  Gets the integer index of the current body. This represents the Body object that the derived output is associated with.

Protected Member Functions

- void addResult (arma::cx_mat ∗input, int index=-1)

  Adds a result to the list of results in the OutputsBody.

Protected Attributes

- std::string pName

  The name for the derived output object. This is a name set by the user to idenfity the object. If multiple objects of the same type are created, the name can also distinguish between the various objects.
- std::string pClassName

The actual name of the class. This is used in outputs writing to file. The class name must be hard coded as a variable because functions such as typeid produce unreliable formatting of the output. The class name is automatically set by the class constructor. Getter functions can only retrieve the variable. Not alter it.

- OutputsBody ∗ pParentBody

  Pointer to the OutputsBody object.

Additional Inherited Members

15.30.1 Detailed Description

This abstract class represents the Derived Outputs. Derived outputs are any additional information that needs to be calculated from basic information within a body. That can be anything from taking derivatives to calculating an empirical equation for motion sickness incidence. The constructor for this class requires a pointer to the parent class to access any further information required for calculation.

The Derived Output class can not be used directly. Individual types of Derived Outputs must be developed and inherit this Derived Output class. This abstract class provides a common framework that all Derived Outputs must share. The most important part is the calcOutput() method. Everything that uses a Derived Output object expects to have this calcOutput() method, and will call it by that name.

See Also

OutputDerived::calcOutput();

Definition at line 101 of file outputderived.h.

15.30.2 Constructor & Destructor Documentation

15.30.2.1 OutputDerived::OutputDerived ( )

Default constructor. Nothing happens here.

Definition at line 38 of file outputderived.cpp.

15.30.2.2 OutputDerived::OutputDerived (OutputsBody ∗ *input*)

Constructor that also sets the pointer to the OutputsBody object which contains the OutputDerived object.

Parameters

input	Pointer to the OutputsBody objec that contains this OutputDerived object. Pointer passed by value.

See Also

setOutputsBody()

Definition at line 43 of file outputderived.cpp.

15.30.2.3 OutputDerived::∼OutputDerived ( )

The default destructor. Nothing happens here.

Definition at line 50 of file outputderived.cpp.

15.30.3 Member Function Documentation

15.30.3.1 void OutputDerived::addResult (arma::cx_mat ∗ *input*, int *index* = -1) [protected]

Adds a result to the list of results in the OutputsBody.

The list of results contains all the results from calculating each DerivedOutput. The DerivedOutput objects also have direct access to this list. But this function handles all the tedious tasks of resizing the list and preventing anything from going out of bounds.

Parameters

input	The result that you wish to add to the list of results. Input is a pointer to a matrix of complex numbers. Please be sure to create all your matrices on the stack so they don't get destroyed once they go out of scope. Don't worry about memory cleanup. The OutputsBody object has a Reset() function that automatically deletes all variables from the list of results and clears the memory.
index	[Optional] Integer input. The index specifies the index in the vector in which you wish to enter the result. This input is optional. If no index is specified, the function automatically adds the result as a new entry on the end of the list.

Definition at line 149 of file outputderived.cpp.

15.30.3.2 virtual int osea::ofreq::OutputDerived::calcOutput (int *freqIn* = -1) [pure virtual]

Pure virtual member. Calculates the output from the OutputDerived object.

Writes results of calculation to the Results matrix in the OutputsBody object that contains this OutputDerived object. Calling the calcOutput() function only generates the results. They must be retrieved from the OutputsBody object in a separate function, using getResult() function.

Results written to the Results matrix are always stored in a matrix of complex values. The exact meaning and organization of the complex matrix changes with each type of OutputDerived object created as a child of this class. The cx_mat data type is used because that is the most natural data type for the largest number of OutputDerived objects. It isn't always the best, but it can usually work well for the intended purposes.

Parameters

freqIn	The wave frequency to use for calculating the OutputDerived object. Specifies the index of the wave frequency to retrieve from the list of wave frequencies. Most outputs will depend on the wave frequency.

Returns

Returns an integer for output. This integer is not the calculation result. It reports on whether the calculation is successful. A returned value of zero (0) means a successful calculation. Other returned values are error codes, each with their own meaning.

See Also

OutputsBody::getResult()

Implemented in osea::ofreq::GlobalSolution.

15.30.3.3 std::string OutputDerived::getClassName ( )

Returns the name of the class used to create the OutputDerived object. This is the name set to identify the type of output object. If multiple objects of the same type are created, the class name will be the same for all objects.

Returns

Returns the class name of the OutputDerived object. std::string variable. Variable passed by value.

Definition at line 67 of file outputderived.cpp.

15.30.3.4 int OutputDerived::getCurBodyIndex ( )

Gets the integer index of the current body. This represents the Body object that the derived output is associated with.

Returns

Returns the integer index of the current Body object associated with this derived output object. Variable is passed by value.

Definition at line 133 of file outputderived.cpp.

15.30.3.5 double OutputDerived::getCurWaveDir ( )

Gets the current wave direction. Output is the actual value for the current wave direction, in units of radians.

Returns

Returns a double that is the current wave direction, in units of radians. Variable is passed by value.

Definition at line 127 of file outputderived.cpp.

15.30.3.6 int OutputDerived::getCurWaveInd ( )

Gets the index of the current wave direction used by the OutputDerived object. Some child classes will need this to determine proper calculation of output. This differs from the getCurWaveDir() method in that the other method returns the actual value of the angle. This returns the index of the current angle in the list of wave directions.

Returns

Returns an integer that represents the index of the current angle in the list of wave directions. Variable passed by value.

Definition at line 139 of file outputderived.cpp.

15.30.3.7 std::string OutputDerived::getName ( )

Returns the name of the OutputDerived object. This is a name set by the user to idenfity the object. If multiple objects of the same type are created, the name can also distinguish between the various objects.

Returns

Returns the name of the OutputDerived object. std::string variable. Variable passed by value.

Definition at line 61 of file outputderived.cpp.

15.30.3.8 vector< Body > & OutputDerived::listBody ( )

Returns the list of body objects. Only a c++ reference to the list of body objects.

Returns

Returns reference to the list of Body objects. Variable passed by reference.

Definition at line 79 of file outputderived.cpp.

15.30.3.9 Body & OutputDerived::listBody (int *bodIn*)

Returns a reference to an individual Body in the list of Body objects.

Parameters

bodIn	The integer index of the body you wish to retrieve from the list of Body objects.

Returns

Returns the Body object requested. Returned variable is passed by reference.

Definition at line 85 of file outputderived.cpp.

15.30.3.10 vector< double > & OutputDerived::listFreq ( )

Gets the list of frequencies. Frequencies are in radians per second.

Returns

Returns the list of wave frequencies. Variable is passed by reference. Variable is stored internally as a pointer.

Definition at line 103 of file outputderived.cpp.

15.30.3.11 double & OutputDerived::listFreq (int *index*)

Returns individual frequency from the list of wave frequencies.

Returns the frequency specified by the index.

Parameters

index	Integer. The index which specifies which wave frequency to return.

Returns

Double. Returns individual frequency from the list of wave frequencies. Returned variabled is passed by reference.

Definition at line 109 of file outputderived.cpp.

15.30.3.12 vector< SolutionSet > & OutputDerived::listSolutionSet ( )

Returns a vector of SolutionSet objects.

Returns

Returns a vector of SolutionSet objects. Internal storage is just a set of pointers to the object. Variable is passed by reference.

Definition at line 91 of file outputderived.cpp.

15.30.3.13 SolutionSet & OutputDerived::listSolutionSet (int *index*)

Returns a vector of SolutionSet objects.

Parameters

index	Integer. Specifies the index for which to retrieve the solution set. If the requested index is out of bounds, the program will return an error.

Returns

Returns a vector of SolutionSet objects. Internal storage is just a set of pointers to the object. Variable is passed by reference.

Definition at line 97 of file outputderived.cpp.

15.30.3.14 vector< double > & OutputDerived::listWaveDir ( )

Gets the list of wave directions. Wave directions are measured in radians. True North is zero, with positive angles going counter-clockwise.

Returns

Returns the vector of doubles containing the wave directions. Variable passed by reference. Variable is stored internally as a pointer.

Definition at line 115 of file outputderived.cpp.

15.30.3.15 double & OutputDerived::listWaveDir (int *index*)

Returns individual wave direction from the list of wave directions.

Returns the wave direction specified by the index. Wave directions are measured in radians. True North is zero, with positive angles going counter-clockwise.

Parameters

index	Integer. The index which specifies which wave direction to return.

Returns

Double. Returns individual wave direction from the list of wave directions. Returned variable is passed by reference.

Definition at line 121 of file outputderived.cpp.

15.30.3.16 void OutputDerived::setName (std::string *nameIn*)

Sets the name of the OutputDerived object. This is a name set by the user to idenfity the object. If multiple objects of the same type are created, the name can also distinguish between the various objects.

Parameters

nameIn	std::string variable. Sets the name of the OutputDerived object. Variable passed by value.

Definition at line 73 of file outputderived.cpp.

15.30.3.17 void OutputDerived::setOutputBody (OutputsBody * *input*)

Sets the pointer to the OutputsBody object which contains this OutputDerived object.

The OutputsBody object contains critical information that each DerivedOutput object may require. This information is made avaiable through the pParentBody pointer. Available information includes:

- list of Body objects.
- list of SolutionSet objects.
- list of wave frequencies. Wave frequency recorded in units of radians per second.
- list of wave directions. Wave direction recorded in units of radians. Zero is true North direction. Oriented positive counter-clockwise.
- The current wave direction used for calculating the DerivedOutput objects.

Parameters

input	Pointer to the OutputsBody objec that contains this OutputDerived object. Pointer passed by value.

See Also

OutputsBody

Definition at line 55 of file outputderived.cpp.

15.30.4 Member Data Documentation

15.30.4.1 std::string osea::ofreq::OutputDerived::pClassName [protected]

The actual name of the class. This is used in outputs writing to file. The class name must be hard coded as a variable because functions such as typeid produce unreliable formatting of the output. The class name is automatically set by the class constructor. Getter functions can only retrieve the variable. Not alter it.

Definition at line 308 of file outputderived.h.

15.30.4.2 std::string osea::ofreq::OutputDerived::pName [protected]

The name for the derived output object. This is a name set by the user to idenfity the object. If multiple objects of the same type are created, the name can also distinguish between the various objects.

Definition at line 299 of file outputderived.h.

15.30.4.3 OutputsBody* osea::ofreq::OutputDerived::pParentBody [protected]

Pointer to the OutputsBody object.

The OutputsBody object contains critical information that each DerivedOutput object may require. This information is made avaiable through the pParentBody pointer. Available information includes:

- list of Body objects.
- list of SolutionSet objects.
- list of wave frequencies. Wave frequency recorded in units of radians per second.
- list of wave directions. Wave direction recorded in units of radians. Zero is true North direction. Oriented positive counter-clockwise.

- THe current wave direction used for calculating the DerivedOutput objects.

See Also

OutputsBody

Definition at line 342 of file outputderived.h.

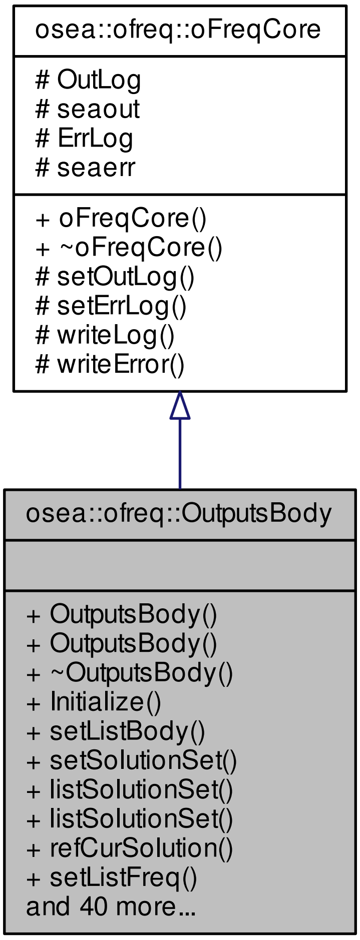
The documentation for this class was generated from the following files:

- bin/ofreq/derived_outputs/outputderived.h
- bin/ofreq/derived_outputs/outputderived.cpp

15.31 osea::ofreq::OutputsBody Class Reference

```
#include <outputsbody.h>
```

Inheritance diagram for osea::ofreq::OutputsBody:



Public Member Functions

- OutputsBody ()

  Default constructor for the OutputsBody object. Nothing happens here.
- OutputsBody (std::vector< Body > &listBod, std::vector< SolutionSet > &listSoln, std::vector< double > &listFreq, std::vector< double > &listWaveDir)

  Overloaded constructor for the OutputsBody object. Includes inputs for Body list, Solution list, frequencies list, and wave directions list. All inputs are passed by reference and held as constant variables to avoid changing the referenced variables.
- ∼OutputsBody ()

  Default destructor. Nothing happens here.
- void Initialize ()

  Creates each of the OutputDerived objects in their respective lists.
- void setListBody (std::vector< Body > &listIn)

  Sets the list of Body objects to calculate derived outputs for. Derived outputs are calculated for only one Body object in the list. The rest are included as reference for cross-body forces.
- void setSolutionSet (std::vector< SolutionSet > &listIn)

  Sets the list of SolutionSet objects to calculate derived outputs for. Derived outputs are calculated for only one SolutionSet object in the list. The rest are included as reference for cross-body forces.
- std::vector< SolutionSet > & listSolutionSet ()

  Provides the list of the SolutionSet objects.
- SolutionSet & listSolutionSet (int index)

  Provides a single entry from the list of the SolutionSet objects.
- osea::ofreq::SolutionSet & refCurSolution ()

  Provides access to the solution object for the current body. Saves the trouble of trying to remember which is the current body.
- void setListFreq (std::vector< double > &listIn)

  Sets the list of wave frequencies.
- std::vector< double > & listFreq ()

  Gets the list of wave frequencies. Provides direct access to the list.
- double & listFreq (int index)

  Returns individual frequency from the list of wave frequencies.
- void setListWaveDir (std::vector< double > &listIn)

  Sets the list of wave directions.
- std::vector< double > & listWaveDir ()

  Gets the list of wave directions. Provides direct access to the list.
- double & listWaveDir (int index)

  Returns individual wave direction from the list of wave directions.
- void setCurWaveDir (int index)

  Sets the current wave direction.
- int getCurWaveInd ()

  Returns the index of the current wave directio entry.
- double getCurWaveDir ()

  Gets the current wave direction. Output is the actual value for the current wave direction, in units of radians.
- void setCurBody (int index)

  Sets the current Body to use. Input selects from the list of supplied Body objects.
- int getCurBodyIndex ()

  Gets the index of the current Body in the list of Body objects.
- Body & refCurBody ()

  Gets the current Body objects used by the class for calculating Derived Outputs. The returned value depends on the last input of setCurBody.
- std::vector< Body > & listBody ()

Provides direct access to the list of Bodies.
- Body & listBody (int bodIn)

  Direct access to an individual Body from the list of Bodies.
- void setCurOutput (OutputDerived ∗input)

  Sets the pointer to the last OutputDerived object that calculated the results and wrote them for access.
- OutputDerived & refCurOutput ()

  Provides direct access to the last OutputDerived object that calculated the results.
- OutputDerived ∗ getCurOutput ()

  Provides direct access to the last OutputDerived object that calculated the results.
- std::vector< arma::cx_mat ∗ > & listResult ()

  The results from calculation of a DerivedOutput.
- arma::cx_mat & listResult (unsigned int index)

  The results from calculation of a DerivedOutput.
- arma::cx_mat ∗& refResult (unsigned int index)

  The results from calculation of a DerivedOutput.
- void ClearResult ()

  Clears all calculated results.
- std::vector< GlobalMotion ∗ > & listGlobalMotion ()

  Returns the list of GlobalMotion objects.
- GlobalMotion & listGlobalMotion (unsigned int index)

  Returns the GlobalMotion object from the list of GlobalMotion objects.
- int calcGlobalMotion (unsigned int index=0)

  Calculates the GlobalMotion object from the list of GlobalMotion objects.
- void addGlobalMotion (GlobalMotion ∗input)

  Adds a new GlobalMotion object to the list of GlobalMotion objects. This version sets the new GlobalMotion object equal to the information passed in.
- void addGlobalMotion ()

  Overloaded function for adding a GlobalMotion object to the list of GlobalMotion objects. This version has no input GlobalMotion object and automatically creates a blank GlobalMotion object.
- int calcGlobalVelocity (unsigned int index=0)

  Calculates the GlobalVelocity object from the list of GlobalVelocity objects.
- std::vector< GlobalVelocity ∗ > & listGlobalVelocity ()

  Returns the list of GlobalVelocity objects.
- GlobalVelocity & listGlobalVelocity (unsigned int index)

  Returns the GlobalVelocity object from the list of GlobalVelocity objects.
- void addGlobalVelocity (GlobalVelocity ∗input)

  Adds a new GlobalVelocity object to the list of GlobalVelocity objects. This version sets the new GlobalVelocity object equal to the information passed in.
- void addGlobalVelocity ()

  Overloaded function for adding a GlobalVelocity object to the list of GlobalVelocity objects. This version has no input GlobalVelocity object and automatically creates a blank GlobalVelocity object.
- int calcGlobalAcceleration (unsigned int index=0)

  Calculates the GlobalAcceleration object from the list of GlobalAcceleration objects.
- std::vector
  < GlobalAcceleration ∗ > & listGlobalAcceleration ()

  Returns the list of GlobalAcceleration objects.
- GlobalAcceleration & listGlobalAcceleration (unsigned int index)

  Returns the GlobalAcceleration object from the list of GlobalAcceleration objects.
- void addGlobalAcceleration (GlobalAcceleration ∗input)

  Adds a new GlobalAcceleration object to the list of GlobalAcceleration objects. This version sets the new Global-Acceleration object equal to the information passed in.
- void addGlobalAcceleration ()

Overloaded function for adding a GlobalAcceleration object to the list of GlobalAcceleration objects. This version has no input GlobalAcceleration object and automatically creates a blank GlobalAcceleration object.

- int calcGlobalSolution (unsigned int index=0)

  Calculates the GlobalSolution object from the list of GlobalSolution objects.
- std::vector< GlobalSolution * > & listGlobalSolution ()

  Returns the list of GlobalSolution objects.
- GlobalSolution & listGlobalSolution (unsigned int index)

  Returns the GlobalSolution object from the list of GlobalSolution objects.
- void addGlobalSolution (GlobalSolution *input)

  Adds a new GlobalSolution object to the list of GlobalSolution objects. This version sets the new GlobalSolution object equal to the information passed in.
- void addGlobalSolution ()

  Overloaded function for adding a GlobalSolution object to the list of GlobalSolution objects. This version has no input GlobalSolution object and automatically creates a blank GlobalSolution object.

Additional Inherited Members

15.31.1 Detailed Description

This class holds all types of derived outputs for a single body object. Each derived output type is contained in a vector. This allows multiple instances of each output type. To save on memory, an output type is not calculated until it is requested. And then the derived output immediately returns the calculations. Outputs are typically returned as a vector of objects. Each row in the vector represents a different wave frequency. The object contained in the returned vector The meaning may change with each derived output type. You should check the documentation for each derived output.

Each derived output is listed as its own item in the class. Each derived output class has the following methods associated with it. (For this generic example, the term Output refers to any derived output object.)

1. refOutput: Pointer to the output object itself.
2. getOutput: Calculates the output and writes the result out as a cx_matrix object.
3. addOutput: Adds a new output object to the list of objects. Optionally takes the supplied output. Otherwise creates a new output object in the vector list.

To use the outputsBody class, you must supply several properties to it. These properties give the outputs class full knowledge of the current state of affairs. This allows the different derived outputs to have the most flexibility for how to calculate results. Set the following properties:

Data Sets:

1. Bodies list
2. Solutions list
3. Frequencies list
4. Wave directions list

Individual properties

1. Current body for outputsBody object.
2. Current wave direction.

Once these properties are set, you may calculate derived outputs. Inputs for derived outputs may be set before all these properties are set.

The following derived outputs are available:

See Also

GlobalSolution
GlobalMotion
GlobalVelocity
GlobalAcceleration

Developers note: The original scheme had the OutputDerived class include a pointer to the contain parent class, OutputsBody. But this creates a cyclic dependency of header files, and will not compile. The only resolution I found to this was to not include the parent class and pass all the necessary information to each individual OutputDerived object. This is tedious, but within the reasons of the methods defined by the OutputsBody class. And it allow compilation. All data items are passed by reference to avoid excess memory duplication.

Definition at line 145 of file outputsbody.h.

15.31.2 Constructor & Destructor Documentation

15.31.2.1 OutputsBody::OutputsBody ( )

Default constructor for the OutputsBody object. Nothing happens here.

Definition at line 34 of file outputsbody.cpp.

15.31.2.2 osea::ofreq::OutputsBody::OutputsBody (std::vector< **Body** > & *listBod,* std::vector< **SolutionSet** > & *listSoln,* std::vector< double > & *listFreq,* std::vector< double > & *listWaveDir*)

Overloaded constructor for the OutputsBody object. Includes inputs for Body list, Solution list, frequencies list, and wave directions list. All inputs are passed by reference and held as constant variables to avoid changing the referenced variables.

Parameters

listBod	The vector of Body objects to use for the object. Contains all the information about Body forces.
listSoln	The vector of SolutionSet objects to use for the object. Contains all the information about solutions for each body. Each SolutionSet object in the vector represents the solutions for all frequencies for a single Body object.
listFreq	The vector of wave frequencies to use for the object. Each wave frequency corresponds to a Solution object in the SolutionSet object.
listWaveDir	The vector of wave directions to use for the object. This is provided mostly for reference.

15.31.2.3 OutputsBody::∼OutputsBody ( )

Default destructor. Nothing happens here.

Definition at line 59 of file outputsbody.cpp.

15.31.3 Member Function Documentation

15.31.3.1 void OutputsBody::addGlobalAcceleration (**GlobalAcceleration** ∗ *input*)

Adds a new GlobalAcceleration object to the list of GlobalAcceleration objects. This version sets the new GlobalAcceleration object equal to the information passed in.

Parameters

input	Pointer to the new GlobalAcceleration object to add to the list of GlobalAcceleration objects. Pointer is passed by value.

Definition at line 493 of file outputsbody.cpp.

15.31.3.2 void OutputsBody::addGlobalAcceleration ( )

Overloaded function for adding a GlobalAcceleration object to the list of GlobalAcceleration objects. This version has no input GlobalAcceleration object and automatically creates a blank GlobalAcceleration object.

Definition at line 501 of file outputsbody.cpp.

15.31.3.3 void OutputsBody::addGlobalMotion (GlobalMotion ∗ *input*)

Adds a new GlobalMotion object to the list of GlobalMotion objects. This version sets the new GlobalMotion object equal to the information passed in.

Parameters

input	Pointer to the new GlobalMotion object to add to the list of GlobalMotion objects. Pointer is passed by value.

Definition at line 351 of file outputsbody.cpp.

15.31.3.4 void OutputsBody::addGlobalMotion ( )

Overloaded function for adding a GlobalMotion object to the list of GlobalMotion objects. This version has no input GlobalMotion object and automatically creates a blank GlobalMotion object.

Definition at line 360 of file outputsbody.cpp.

15.31.3.5 void OutputsBody::addGlobalSolution (GlobalSolution ∗ *input*)

Adds a new GlobalSolution object to the list of GlobalSolution objects. This version sets the new GlobalSolution object equal to the information passed in.

Parameters

input	Pointer to the new GlobalSolution object to add to the list of GlobalSolution objects. Pointer is passed by value.

Definition at line 562 of file outputsbody.cpp.

15.31.3.6 void OutputsBody::addGlobalSolution ( )

Overloaded function for adding a GlobalSolution object to the list of GlobalSolution objects. This version has no input GlobalSolution object and automatically creates a blank GlobalSolution object.

Definition at line 570 of file outputsbody.cpp.

15.31.3.7 void OutputsBody::addGlobalVelocity (GlobalVelocity ∗ *input*)

Adds a new GlobalVelocity object to the list of GlobalVelocity objects. This version sets the new GlobalVelocity object equal to the information passed in.

Parameters

input	Pointer to the new GlobalVelocity object to add to the list of GlobalVelocity objects. Pointer is passed by value.

Definition at line 422 of file outputsbody.cpp.

15.31.3.8 void OutputsBody::addGlobalVelocity ( )

Overloaded function for adding a GlobalVelocity object to the list of GlobalVelocity objects. This version has no input GlobalVelocity object and automatically creates a blank GlobalVelocity object.

Definition at line 430 of file outputsbody.cpp.

15.31.3.9 int OutputsBody::calcGlobalAcceleration (unsigned int *index* = 0)

Calculates the GlobalAcceleration object from the list of GlobalAcceleration objects.

Outputs from calculation is written to the results matrix. You can retrieve the results from the calculation by use of the getResult() function. Calculating any other DerivedOutput will erase your results from the Results matrix and you will need to recalculate them.

Parameters

index	The index of which GlobalAcceleration object to retrieve from the list of objects. For this Derived Output, there is only one GlobalAcceleration object per OutputsBody. The default value selects this object.

Returns

Returns an integer for output. This integer is not the calculation result. It reports on whether the calculation is successful. A returned value of zero (0) means a successful calculation. Other returned values are error codes, each with their own meaning.

See Also

GlobalAcceleration

Definition at line 462 of file outputsbody.cpp.

15.31.3.10 int OutputsBody::calcGlobalMotion (unsigned int *index* = 0)

Calculates the GlobalMotion object from the list of GlobalMotion objects.

Outputs from calculation is written to the results matrix. You can retrieve the results from the calculation by use of the getResult() function. Calculating any other DerivedOutput will erase your results from the Results matrix and you will need to recalculate them.

Parameters

index	The index of which GlobalMotion object to retrieve from the list of objects. For this Derived Output, there is only one GlobalMotion object per OutputsBody. The default value selects this object.

Returns

Returns an integer for output. This integer is not the calculation result. It reports on whether the calculation is successful. A returned value of zero (0) means a successful calculation. Other returned values are error codes, each with their own meaning.

See Also

GlobalMotion

Definition at line 320 of file outputsbody.cpp.

15.31.3.11 int OutputsBody::calcGlobalSolution (unsigned int *index* = 0)

Calculates the GlobalSolution object from the list of GlobalSolution objects.

Outputs from calculation is written to the results matrix. You can retrieve the results from the calculation by use of the getResult() function. Calculating any other DerivedOutput will erase your results from the Results matrix and you will need to recalculate them.

Parameters

index	The index of which GlobalSolution object to retrieve from the list of objects. For this Derived Output, there is only one GlobalSolution object per OutputsBody. The default value selects this object.

Returns

Returns an integer for output. This integer is not the calculation result. It reports on whether the calculation is successful. A returned value of zero (0) means a successful calculation. Other returned values are error codes, each with their own meaning.

See Also

GlobalSolution

Definition at line 531 of file outputsbody.cpp.

15.31.3.12 int OutputsBody::calcGlobalVelocity (unsigned int *index* = 0)

Calculates the GlobalVelocity object from the list of GlobalVelocity objects.

Outputs from calculation is written to the results matrix. You can retrieve the results from the calculation by use of the getResult() function. Calculating any other DerivedOutput will erase your results from the Results matrix and you will need to recalculate them.

Parameters

index	The index of which GlobalVelocity object to retrieve from the list of objects. For this Derived Output, there is only one GlobalVelocity object per OutputsBody. The default value selects this object.

Returns

Returns an integer for output. This integer is not the calculation result. It reports on whether the calculation is successful. A returned value of zero (0) means a successful calculation. Other returned values are error codes, each with their own meaning.

See Also

GlobalVelocity

Definition at line 391 of file outputsbody.cpp.

15.31.3.13 void OutputsBody::ClearResult ( )

Clears all calculated results.

Clears the calculated results from all internal storage. And removes any pointers to the DerivedOutput object that calculated the result. The DerivedOutput object itself is not deleted.

Definition at line 277 of file outputsbody.cpp.

15.31.3.14 int OutputsBody::getCurBodyIndex ( )

Gets the index of the current Body in the list of Body objects.

Returns

Returns the index of the current Body assigned to this OutputsBody object. This differs from other function refCurBody() because refCurBody() returns a pointer to the body directly. But this function, getCurBodyIndex() returns the integer index of the Body object in the vector list of Body objects. Returned variable is passed by value.

Definition at line 201 of file outputsbody.cpp.

15.31.3.15 OutputDerived * OutputsBody::getCurOutput ( )

Provides direct access to the last OutputDerived object that calculated the results.

When a call is made to calculate the outputs of a DerivedOutput object, that object writes its results to the Results storage in the OutputsBody. The pointer is then set to that OutputDerived object. This is in case you need to access the OutputDerived object for any reason. You don't need to remember which object did the calculation. You can just access the object.

Returns

Returns a pointer to the OutputDerived object that performed the last calculation. Pointer passed by value.

Definition at line 237 of file outputsbody.cpp.

15.31.3.16 double OutputsBody::getCurWaveDir ( )

Gets the current wave direction. Output is the actual value for the current wave direction, in units of radians.

Returns

Returns a double that is the current wave direction, in units of radians. Variable is passed by value.

Definition at line 189 of file outputsbody.cpp.

15.31.3.17 int OutputsBody::getCurWaveInd ( )

Returns the index of the current wave directio entry.

Returns

Integer. Returns the index of the current wave directio entry. Variable passed by value.

Definition at line 183 of file outputsbody.cpp.

15.31.3.18 void OutputsBody::Initialize ( )

Creates each of the OutputDerived objects in their respective lists.

Each list of OutputDerived object can contain any number of objects. The initialize function is called to generate each of these OutputDerived objects.

Definition at line 86 of file outputsbody.cpp.

15.31.3.19 std::vector< Body > & OutputsBody::listBody ( )

Provides direct access to the list of Bodies.

Returns

Reference to vector of Body objects. Variable passed by reference.

See Also

Body

Definition at line 213 of file outputsbody.cpp.

15.31.3.20 Body & OutputsBody::listBody (int *bodIn*)

Direct access to an individual Body from the list of Bodies.

Parameters

bodIn	Integer specifying which Body object to access in the list of Bodies.

Returns

Returns reference to the Body object specified by input bodIn.

See Also

listBody()

Definition at line 219 of file outputsbody.cpp.

15.31.3.21 vector< double > & OutputsBody::listFreq ( )

Gets the list of wave frequencies. Provides direct access to the list.

Returns

Returns the vector of doubles representing the wave frequencies. Frequencies entered in units of radians per second. Variable is passed by reference.

Definition at line 147 of file outputsbody.cpp.

15.31.3.22 double & OutputsBody::listFreq (int *index*)

Returns individual frequency from the list of wave frequencies.

Returns the frequency specified by the index.

Parameters

index	Integer. The index which specifies which wave frequency to return.

Returns

Double. Returns individual frequency from the list of wave frequencies. Returned variabled is passed by reference.

Definition at line 153 of file outputsbody.cpp.

15.31.3.23 vector< GlobalAcceleration * > & OutputsBody::listGlobalAcceleration ( )

Returns the list of GlobalAcceleration objects.

Returns

Returns a vector of pointers to each of the global motion objects. Returned variable passed by reference.

Definition at line 439 of file outputsbody.cpp.

15.31.3.24 GlobalAcceleration & OutputsBody::listGlobalAcceleration (unsigned int *index*)

Returns the GlobalAcceleration object from the list of GlobalAcceleration objects.

See Also

GlobalAcceleration

Parameters

index	The index of which GlobalAcceleration object to retrieve from the list of objects. For this Derived Output, there is only one GlobalAcceleration object per OutputsBody. The default value selects this object.

Returns

Returns the GlobalAcceleration object from the list of GlobalAcceleration objects. Returned variable is passed by reference. Returns only the object specified by the input index.

Definition at line 445 of file outputsbody.cpp.

15.31.3.25 std::vector< GlobalMotion * > & OutputsBody::listGlobalMotion ( )

Returns the list of GlobalMotion objects.

Returns

Returns a vector of pointers to each of the global motion objects. Returned variable passed by reference.

Definition at line 298 of file outputsbody.cpp.

15.31.3.26 GlobalMotion & OutputsBody::listGlobalMotion (unsigned int *index*)

Returns the GlobalMotion object from the list of GlobalMotion objects.

See Also

GlobalMotion

Parameters

index	The index of which GlobalMotion object to retrieve from the list of objects. For this Derived Output, there is only one GlobalMotion object per OutputsBody. The default value selects this object.

Returns

Returns the GlobalMotion object from the list of GlobalMotion objects. Returned variable is passed by reference. Returns only the object specified by the input index.

Definition at line 304 of file outputsbody.cpp.

15.31.3.27 vector< GlobalSolution ∗ > & OutputsBody::listGlobalSolution ( )

Returns the list of GlobalSolution objects.

Returns

Returns a vector of pointers to each of the global motion objects. Returned variable passed by reference.

Definition at line 509 of file outputsbody.cpp.

15.31.3.28 GlobalSolution & OutputsBody::listGlobalSolution (unsigned int *index*)

Returns the GlobalSolution object from the list of GlobalSolution objects.

See Also

GlobalSolution

Parameters

index	The index of which GlobalSolution object to retrieve from the list of objects. For this Derived Output, there is only one GlobalSolution object per OutputsBody. The default value selects this object.

Returns

Returns the GlobalSolution object from the list of GlobalSolution objects. Returned variable is passed by reference. Returns only the object specified by the input index.

Definition at line 515 of file outputsbody.cpp.

15.31.3.29 std::vector< GlobalVelocity ∗ > & OutputsBody::listGlobalVelocity ( )

Returns the list of GlobalVelocity objects.

Returns

Returns a vector of pointers to each of the global motion objects. Returned variable passed by reference.

Definition at line 369 of file outputsbody.cpp.

15.31.3.30 GlobalVelocity & OutputsBody::listGlobalVelocity (unsigned int *index*)

Returns the GlobalVelocity object from the list of GlobalVelocity objects.

See Also

GlobalVelocity

Parameters

index	The index of which GlobalVelocity object to retrieve from the list of objects. For this Derived Output, there is only one GlobalVelocity object per OutputsBody. The default value selects this object.

Returns

Returns the GlobalVelocity object from the list of GlobalVelocity objects. Returned variable is passed by reference. Returns only the object specified by the input index.

Definition at line 375 of file outputsbody.cpp.

15.31.3.31 std::vector< arma::cx_mat $*$ > & OutputsBody::listResult ( )

The results from calculation of a DerivedOutput.

When a call is made to calculate a DerivedOutput object, the object stores the results of its calculation in the results matrix. Those results can be accessed through this method. This method is also used by the DerivedOutput object to write the results. DerivedOutput object automatically resizes the output buffer as needed. The output buffer is a vector storing pointers to matrices of complex numbers.

Returns

Returns direct access to the stored results matrix. Returned variable is a vector storing pointers to matrices of complex numbers. matrix of undetermined size. Returned variable is passed by reference.

Definition at line 243 of file outputsbody.cpp.

15.31.3.32 arma::cx_mat & OutputsBody::listResult (unsigned int *index*)

The results from calculation of a DerivedOutput.

When a call is made to calculate a DerivedOutput object, the object stores the results of its calculation in the results matrix. Those results can be accessed through this method. This method is also used by the DerivedOutput object to write the results. DerivedOutput object automatically resizes the output buffer as needed. The output buffer is a vector storing pointers to matrices of complex numbers.

Parameters

index	Integer input that specifies which matrix to retrieve from the list of results. Most commonly, the index represents the index of a wave frequency from the list of wave frequencies. (i.e. The list is organized by wave frequencies.)

Returns

Returns direct access to the stored results matrix. Returned variable is a matrix of complex numbers. Matrix of undetermined size. Returned variable is passed by reference.

Definition at line 249 of file outputsbody.cpp.

15.31.3.33 std::vector< SolutionSet > & OutputsBody::listSolutionSet ( )

Provides the list of the SolutionSet objects.

Derived outputs are calculated for only one SolutionSet object in the list. The rest are included as reference for cross-body forces.

Returns

Returns a vector containing the SolutionSet objects. Variable passed by reference.

Definition at line 116 of file outputsbody.cpp.

15.31.3.34 osea::ofreq::SolutionSet & OutputsBody::listSolutionSet (int *index*)

Provides a single entry from the list of the SolutionSet objects.

Derived outputs are calculated for only one SolutionSet object in the list. The rest are included as reference for cross-body forces. This implementation of the function only returns a single entry from the list.

Parameters

index	Integer. Specifies the index for which to retrieve the solution set. If the requested index is out of bounds, the program will return an error.

Returns

Returns a single SolutionSet object requested from the list of SolutionSet objects. Requested variable is passed by reference.

Definition at line 122 of file outputsbody.cpp.

15.31.3.35 vector< double > & OutputsBody::listWaveDir ( )

Gets the list of wave directions. Provides direct access to the list.

Returns

Returns the vector of doubles representing the wave directions. Directions entered in units of radians. Variable is passed by reference.

Definition at line 165 of file outputsbody.cpp.

15.31.3.36 double & OutputsBody::listWaveDir (int *index*)

Returns individual wave direction from the list of wave directions.

Returns the wave direction specified by the index. Wave directions are measured in radians. True North is zero, with positive angles going counter-clockwise.

Parameters

index	Integer. The index which specifies which wave direction to return.

Returns

Double. Returns individual wave direction from the list of wave directions. Returned variable is passed by reference.

Definition at line 171 of file outputsbody.cpp.

15.31.3.37 Body & OutputsBody::refCurBody ( )

Gets the current Body objects used by the class for calculating Derived Outputs. The returned value depends on the last input of setCurBody.

Returns

Returns a pointer the Body object used by the class for calculating Derived Outputs. The returned variable is passed by reference.

Definition at line 207 of file outputsbody.cpp.

15.31.3.38 OutputDerived & OutputsBody::refCurOutput ( )

Provides direct access to the last OutputDerived object that calculated the results.

When a call is made to calculate the outputs of a DerivedOutput object, that object writes its results to the Results storage in the OutputsBody. The pointer is then set to that OutputDerived object. This is in case you need to access the OutputDerived object for any reason. You don't need to remember which object did the calculation. You can just access the object.

Returns

Returns the OutputDerived object that performed the last calculation. Variable passed by reference.

Definition at line 231 of file outputsbody.cpp.

15.31.3.39 osea::ofreq::SolutionSet & OutputsBody::refCurSolution ( )

Provides access to the solution object for the current body. Saves the trouble of trying to remember which is the current body.

Returns

Returns the solution object for the current body. Returned variable passed by reference.

Definition at line 135 of file outputsbody.cpp.

15.31.3.40 arma::cx_mat ∗& OutputsBody::refResult (unsigned int *index*)

The results from calculation of a DerivedOutput.

When a call is made to calculate a DerivedOutput object, the object stores the results of its calculation in the results matrix. Those results can be accessed through this method. This method is also used by the DerivedOutput object to write the results. DerivedOutput object automatically resizes the output buffer as needed. The output buffer is a vector storing pointers to matrices of complex numbers.

Parameters

index	Integer input that specifies which matrix to retrieve from the list of results.

Returns

Returns a pointer to the stored results matrix. Returned variable is a pointer to a matrix of complex numbers. Matrix of undetermined size. Returned variable is passed by reference.

Definition at line 263 of file outputsbody.cpp.

15.31.3.41 void OutputsBody::setCurBody (int *index*)

Sets the current Body to use. Input selects from the list of supplied Body objects.

Parameters

index	Integer input that selects from the list of supplied Body objects. Variable is passed by value.

Definition at line 195 of file outputsbody.cpp.

15.31.3.42 void OutputsBody::setCurOutput (OutputDerived ∗ *input*)

Sets the pointer to the last OutputDerived object that calculated the results and wrote them for access.

When a call is made to calculate the outputs of a DerivedOutput object, that object writes its results to the Results storage in the OutputsBody. The pointer is then set to that OutputDerived object. This is in case you need to access the OutputDerived object for any reason. You don't need to remember which object did the calculation. You can just access the object.

Parameters

input	Pointer to the OutputDerived object that performed the last results calculation.

Definition at line 225 of file outputsbody.cpp.

15.31.3.43 void OutputsBody::setCurWaveDir (int *index*)

Sets the current wave direction.

Parameters

index	Integer input specifying the index of the current wave direction from the list set by setListWave-Dir.

Definition at line 177 of file outputsbody.cpp.

15.31.3.44 void OutputsBody::setListBody (std::vector< Body > & *listIn*)

Sets the list of Body objects to calculate derived outputs for. Derived outputs are calculated for only one Body object in the list. The rest are included as reference for cross-body forces.

Parameters

listIn	The vector of Body objects to assign to this OutputsBody. Input is passed by reference. Input is held as a constant value, so that it can not be modified by the class.

Definition at line 104 of file outputsbody.cpp.

15.31.3.45 void OutputsBody::setListFreq (std::vector< double > & *listIn*)

Sets the list of wave frequencies.

Parameters

listIn	The vector of doubles representing the wave frequencies. Frequencies entered in units of radians per second. Input is passed by reference and held as a constant so that the class can not change the frequencies.

Definition at line 141 of file outputsbody.cpp.

15.31.3.46 void OutputsBody::setListWaveDir (std::vector< double > & *listIn*)

Sets the list of wave directions.

Parameters

listIn	The vector of doubles representing the wave directions. Directions entered in units of radians. Input is passed by reference and held as a constant so that the class can not change the directions.

Definition at line 159 of file outputsbody.cpp.

15.31.3.47 void OutputsBody::setSolutionSet (std::vector< SolutionSet > & *listIn*)

Sets the list of SolutionSet objects to calculate derived outputs for. Derived outputs are calculated for only one SolutionSet object in the list. The rest are included as reference for cross-body forces.

Parameters

listIn	The vector of SolutionSet objects to assign to this OutputsBody. Input is passed by reference. Inputs is held as a constant value, so that it can not be modified by the class.

Definition at line 110 of file outputsbody.cpp.

The documentation for this class was generated from the following files:

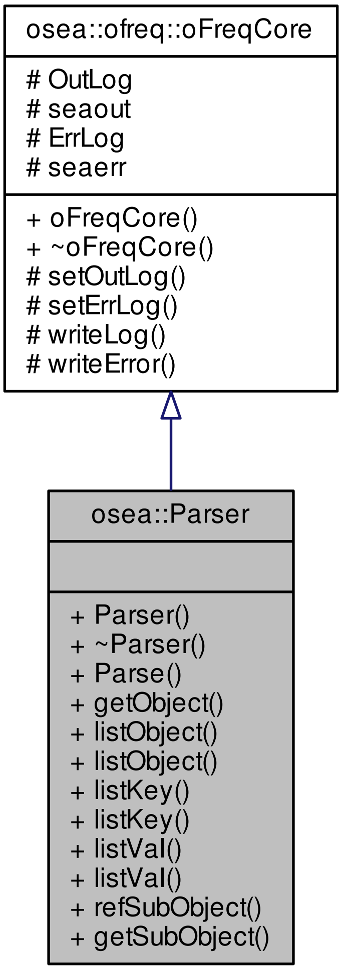
- bin/ofreq/derived_outputs/outputsbody.h
- bin/ofreq/derived_outputs/outputsbody.cpp

15.32 osea::Parser Class Reference

The Parser class takes an input segment of strings and segments that segment. It strips out comments. It recognizes quotation marks and groups those segments together. Parser finally returns a series of ObjectGroup objects. Each ObjectGroup contains the classname and a list of keyword value pairs.

`#include <parser.h>`

Inheritance diagram for osea::Parser:



Public Member Functions

- Parser ()

  Default constructor. Nothing happens here.
- ∼Parser ()
- void Parse (std::istream &infile, int bracket_count=0)

  Parses the input files into a vector of key-value pairs.
- ObjectGroup getObject (int index=0)

  Gets the ObjectGroup object.
- std::vector< ObjectGroup > & listObject ()

  Provides direct access to the vector of ObjectGroup objects processed by the Parser object.
- ObjectGroup & listObject (unsigned int index)

  Provides direct access to the ObjectGroup object specified by index.
- vecKeyword & listKey ()

  Provides direct access to thelist of key words.

- std::string & listKey (int index)

    Provides direct access to an individual item in the list of key words.
- vecValue & listVal ()

    Provides direct access to the list of key values.
- std::vector< std::string > & listVal (int index)

    Provides direct access to an individual item in the list of key values.
- std::vector< ObjectGroup * > & refSubObject (int index1=0)

    Provides direct access to the list of SubObjects.
- ObjectGroup getSubObject (int index, int index1=0)

    Returns the subObject referenced by the index.

Additional Inherited Members

15.32.1 Detailed Description

The Parser class takes an input segment of strings and segments that segment. It strips out comments. It recognizes quotation marks and groups those segments together. Parser finally returns a series of ObjectGroup objects. Each ObjectGroup contains the classname and a list of keyword value pairs.

The parser and ObjectGroup are both setup to allow endless recursion of class definitions. You can easily define classes within classes in the input files, with no limits.

Definition at line 102 of file parser.h.

15.32.2 Constructor & Destructor Documentation

15.32.2.1 Parser::Parser ( )

Default constructor. Nothing happens here.

Definition at line 54 of file parser.cpp.

15.32.2.2 Parser::~Parser ( )

Default destructor. Nothing happens here.

Definition at line 64 of file parser.cpp.

15.32.3 Member Function Documentation

15.32.3.1 ObjectGroup Parser::getObject (int *index* = 0)

Gets the ObjectGroup object.

Parameters

index	Index of which ObjectGroup object to retrieve. If not specified, the default is the first object in the list.

Returns

Returns vector of ObjectGroup objects. Each entry in the vector represents an object. Variable passed by value.

Definition at line 117 of file parser.cpp.

15.32.3.2 ObjectGroup Parser::getSubObject (int *index*, int *index1* = 0)

Returns the subObject referenced by the index.

Parameters

index	Specifies which SubObject to return from the list of SubObjects.
index1	Index of which ObjectGroup object to retrieve. If not specified, the default is the first object in the list.

Returns

Returns a SubObject from the list of SubObjects. Returned variable is passed by value. Contains the actual object and not a pointer to the object.

Definition at line 165 of file parser.cpp.

15.32.3.3 vecKeyword & Parser::listKey ( )

Provides direct access to thelist of key words.

Returns

Returns a reference to the list of key words. Variable passed by reference.

Definition at line 135 of file parser.cpp.

15.32.3.4 string & Parser::listKey (int *index*)

Provides direct access to an individual item in the list of key words.

Returns the key word specified by the index.

Parameters

index	Integer. The index of the key word to retrieve.

Returns

std::string. The key word specified by index. Returned variable passed by reference.

Definition at line 141 of file parser.cpp.

15.32.3.5 vector< ObjectGroup > & Parser::listObject ( )

Provides direct access to the vector of ObjectGroup objects processed by the Parser object.

Returns

Returns reference to the vector of ObjectGroup objects. Variable passed by reference.

Definition at line 123 of file parser.cpp.

15.32.3.6 ObjectGroup & Parser::listObject (unsigned int *index*)

Provides direct access to the ObjectGroup object specified by index.

Returns entry from the list of ObjectGroup objects. Returned entry is specified by index.

Parameters

index	Unsigned integer. The index of the ObjectGroup object to return from the list.

Returns

Returns ObjectGroup object. Returned variable is passed by reference.

See Also

ObjectGroup

Definition at line 129 of file parser.cpp.

15.32.3.7 vecValue & Parser::listVal ( )

Provides direct access to the list of key values.

Returns

Returns a reference to the list of key values. Variable passed by reference.

Definition at line 147 of file parser.cpp.

15.32.3.8 vector< string > & Parser::listVal (int *index*)

Provides direct access to an individual item in the list of key values.

Returns the key value specified by the index.

Parameters

index	Integer. THe index of the key value to retrieve.

Returns

Vector of strings. The key value specified by the index. Returned variable passed by reference.

Definition at line 153 of file parser.cpp.

15.32.3.9 void Parser::Parse (std::istream & *infile*, int *bracket_count* = 0)

Parses the input files into a vector of key-value pairs.

Parameters

infile	The input file to parse. Istream. Variable passed by reference.
bracket_count	The count of object definition brackets. Used as a termination condition in the parsing process. Provides control for any recursive instances of the Parse() function. Default is 0. The bracket count updates while parsing, as the Parser finds new opening ({) and closing (}) brackets. Parsing continues as long as the bracket count is equal to or above the bracket count fed into the Parser as an argument. At this point, the function behavior does not change with what the actual value of bracket_count is. Parsing only requires that bracket_count is >= its starting value. However, several development attempts indicated that the behavior the Parse function may become dependent on the value of bracket_count. So it is left in as an argument to allow functionality expansion in any future development.

See Also

ParseCommands()

Definition at line 70 of file parser.cpp.

15.32.3.10 vector< **ObjectGroup** * > & Parser::refSubObject (int *index1* = 0)

Provides direct access to the list of SubObjects.

Parameters

index1	Index of which ObjectGroup object to retrieve. If not specified, the default is the first object in the list.

Returns

Returns a reference to the list of SubObjects detected by the parser. Returned variable passed by reference. Returned variable is a vector of pointers to the SubObjects.

Definition at line 159 of file parser.cpp.

The documentation for this class was generated from the following files:

- bin/ofreq/file_reader/parser.h
- bin/ofreq/file_reader/parser.cpp

15.33 SeaEnviroment Class Reference

```
#include <seaenviroment.h>
```

Public Member Functions

- SeaEnviroment ()
- ∼SeaEnviroment ()
- void testPrint ()
- void setWaveSpectrumName (string)
- void setWaveSpectrumFrequencies (vector< double >)
- void setWaveSpectrumWaveEnergy (vector< double >)
- void setSpreadModelName (string)
- void setSpreadModelDirectionAngle (double)
- void setSpreadModelWaveSpectrumName (string)
- void setSpreadModelScalingFactor (double)

15.33.1 Detailed Description

This class holds all data for the sea enviroment.

Definition at line 53 of file seaenviroment.h.

15.33.2 Constructor & Destructor Documentation

15.33.2.1 SeaEnviroment::SeaEnviroment ( )

The default constructor.

15.33.2.2 SeaEnviroment::∼SeaEnviroment ( )

The default destructor, nothing happens here.

15.33.3 Member Function Documentation

15.33.3.1 void SeaEnviroment::setSpreadModelDirectionAngle (double)

Sets spread model direction angle.

Parameters

val	The direction angle.

15.33.3.2 void SeaEnviroment::setSpreadModelName (string)

Sets the spread model name.

Parameters

newName	The name of the spread model to be used.

15.33.3.3 void SeaEnviroment::setSpreadModelScalingFactor (double)

Sets scaling factor.

Parameters

val	The scaling factor.

15.33.3.4 void SeaEnviroment::setSpreadModelWaveSpectrumName (string)

Sets spread model direction angle.

Parameters

newName	The name of the wave spectrum.

15.33.3.5 void SeaEnviroment::setWaveSpectrumFrequencies (vector< double >)

Sets the wave frequencies.

Parameters

vecIn	The list of wave frequencies.

15.33.3.6 void SeaEnviroment::setWaveSpectrumName (string)

Sets the wave spectrum.

Parameters

newName	The name of the wave spectrum.

15.33.3.7 void SeaEnviroment::setWaveSpectrumWaveEnergy (vector< double >)

Sets the wave energy.

Parameters

vecIn	The list of wave energy.

15.33.3.8 void SeaEnviroment::testPrint ( )

Test print to console the values of all data members.

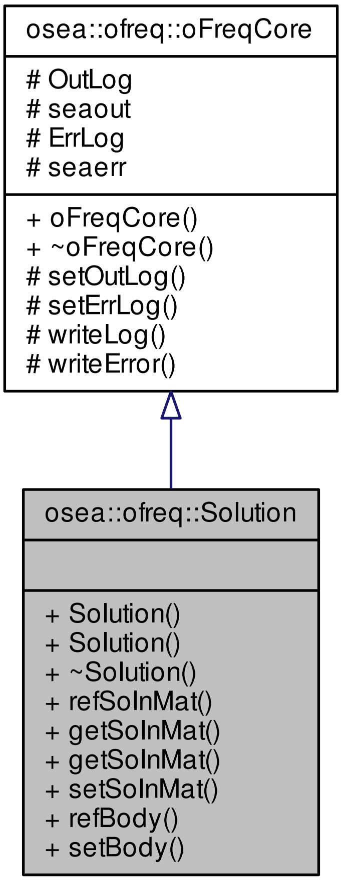
The documentation for this class was generated from the following file:

- bin/ofreq/wave_calcs/seaenviroment.h

15.34 osea::ofreq::Solution Class Reference

```
#include <solution.h>
```

Inheritance diagram for osea::ofreq::Solution:



Public Member Functions

- Solution ()

  Default constructor.
- Solution (Body &bodIn)

  Creates the object and sets the reference body for the object.
- ∼Solution ()

  Default destructor.
- arma::cx_mat & refSolnMat ()

  Returns direct access to the solution matrix.
- arma::cx_mat getSolnMat ()

  Returns the solution matrix.
- std::complex< double > getSolnMat (int index)

  Returns a single entry in the solution matrix. Entry specified by index.
- void setSolnMat (arma::cx_mat matIn)

  Sets the solution matrix as a whole.
- Body & refBody ()

Returns reference pointer to the reference body. Variable passed by reference.

- void setBody (Body ∗input)

  Sets the body object that the solution object is relevant to.

Additional Inherited Members

15.34.1 Detailed Description

This class defines a solution object. The solution object records the basic value of motion solution. The motion solution is translated back into body coordinate system.

Definition at line 89 of file solution.h.

15.34.2 Constructor & Destructor Documentation

15.34.2.1 Solution::Solution ( )

Default constructor.

Default constructor. Nothing done here.

Definition at line 38 of file solution.cpp.

15.34.2.2 osea::ofreq::Solution::Solution (Body & *bodIn*)

Creates the object and sets the reference body for the object.

Creates the object and sets the reference body for the object.

Parameters

bodIn	The reference body. Variabled passed by reference.

15.34.2.3 Solution::∼Solution ( )

Default destructor.

Default destructor. Nothing done here.

Definition at line 44 of file solution.cpp.

15.34.3 Member Function Documentation

15.34.3.1 cx_mat Solution::getSolnMat ( )

Returns the solution matrix.

It gets filled with the output from the motion solver. Output is a column matrix (n by 1) of complex numbers. Output is in units of meters.

Returns the solution matrix as a whole.

Returns

Returned value is a complex number matrix. Returned variable is passed by value.

Definition at line 64 of file solution.cpp.

15.34.3.2 complex< double > Solution::getSolnMat (int *index*)

Returns a single entry in the solution matrix. Entry specified by index.

Parameters

index	The index of the entry to retrieve from the solution matrix.

Returns

Returns a complex number data type. Single value from the solution matrix. Value specified by the index input. Variable passed by value.

Definition at line 57 of file solution.cpp.

15.34.3.3 Body & Solution::refBody ( )

Returns reference pointer to the reference body. Variable passed by reference.

Returns reference pointer to the reference body. Variable passed by reference. Used for direct access to the reference body and all its member functions.

Returns

Returns reference pointer to the reference body. Variable passed by reference.

Definition at line 79 of file solution.cpp.

15.34.3.4 cx_mat & Solution::refSolnMat ( )

Returns direct access to the solution matrix.

Returns direct access to the solution matrix. Provides a pointer to the solution matrix. It gets filled with the output from the motion solver. Output is a column matrix (n by 1) of complex numbers. Output is in units of meters.

Returns

Returns direct access to the solution matrix. Provides a pointer to the solution matrix. Returned variable is passed by reference.

Definition at line 50 of file solution.cpp.

15.34.3.5 void Solution::setBody (Body * *input*)

Sets the body object that the solution object is relevant to.

The solution set in this object is specific to a single body. The setBody method allows you to create a pointer to that Body object for reference and use in other sections of code.

Parameters

input	Pointer to the Body object. Pointer variable passed by value.

Definition at line 86 of file solution.cpp.

15.34.3.6 void Solution::setSolnMat (arma::cx_mat *matIn*)

Sets the solution matrix as a whole.

Parameters

matIn	The input matrix to set as the solution matrix.

Definition at line 71 of file solution.cpp.

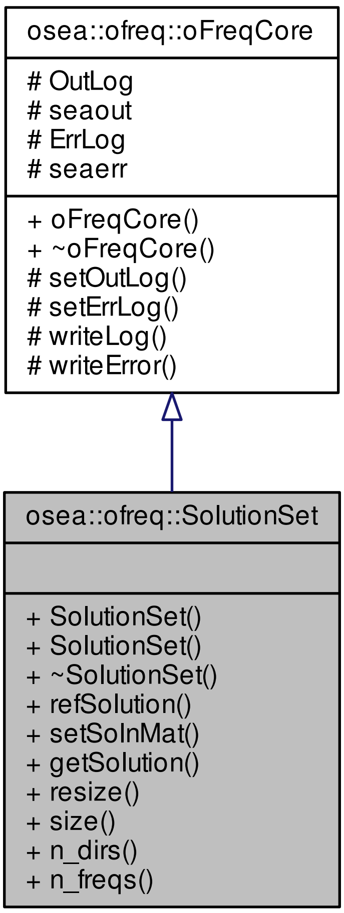
The documentation for this class was generated from the following files:

- bin/ofreq/global_objects/solution.h
- bin/ofreq/global_objects/solution.cpp

15.35 osea::ofreq::SolutionSet Class Reference

```
#include <solutionset.h>
```

Inheritance diagram for osea::ofreq::SolutionSet:



Public Member Functions

- SolutionSet ()

  Default constructor.
- SolutionSet (int dir, int freq)

  Constructor with number of wave directions and wave frequencies specified. Automatically allocates dynamic memory for the specified number of wave directions and wave frequencies.
- ∼SolutionSet ()
- Solution & refSolution (int dir, int freq)

  Returns a pointer to the solution object at the specified wave direction and frequency.
- void setSolnMat (int dir, int freq, Solution soln)

  Sets the solution object at the specified wave direction and frequency.
- Solution getSolution (int dir, int freq)

  Gets the solution object at the specified wave direction and frequency.
- void resize (int dir, int freq)

  Resizes the 2-D array of Solution objects. Any existing objects in the array are preserved. If the array is sized smaller than the existing number of Solution objects, any object beyond the new index range are deleted.
- std::vector< int > size ()

  Returns the size of the matrix as a vector of two elements.
- int n_dirs ()

  Returns the number of rows as an integer. This is the number of wave directions.
- int n_freqs ()

  Returns the number of columns as an integer. This is the number of wave frequencies.

Additional Inherited Members

15.35.1 Detailed Description

This class records the list of solutions obtained for a single Body object. It is essentially a 2D version of the std::vector<object> class. But since vector can't handle 2-d storage solutions, this class was created.

Definition at line 89 of file solutionset.h.

15.35.2 Constructor & Destructor Documentation

15.35.2.1 SolutionSet::SolutionSet ( )

Default constructor.

Definition at line 33 of file solutionset.cpp.

15.35.2.2 SolutionSet::SolutionSet (int *dir*, int *freq*)

Constructor with number of wave directions and wave frequencies specified. Automatically allocates dynamic memory for the specified number of wave directions and wave frequencies.

Parameters

dir	The new number of wave directions to resize the array to.
freq	The new number of wave frequencies to resize the array to.

Definition at line 38 of file solutionset.cpp.

15.35.2.3 SolutionSet::∼SolutionSet ( )

Default destructor. Frees any memory dynamically assigned.

Definition at line 45 of file solutionset.cpp.

15.35.3 Member Function Documentation

15.35.3.1 Solution SolutionSet::getSolution (int *dir*, int *freq*)

Gets the solution object at the specified wave direction and frequency.

Parameters

dir	Integer. Index of the wave direction desired.
freq	Integer. Index of the wave frequency desired.

Returns

Returns the Solution object at the specified wave direction and wave frequency. Returned variable is passed by value.

Definition at line 84 of file solutionset.cpp.

15.35.3.2 int SolutionSet::n_dirs ( )

Returns the number of rows as an integer. This is the number of wave directions.

Returns

Integer. Returns the number of rows as an integer. This is the number of wave directions.

Definition at line 113 of file solutionset.cpp.

15.35.3.3 int SolutionSet::n_freqs ( )

Returns the number of columns as an integer. This is the number of wave frequencies.

Returns

Integer. Returns the number of columns as an integer. This is the number of wave frequencies.

Definition at line 119 of file solutionset.cpp.

15.35.3.4 Solution & SolutionSet::refSolution (int *dir*, int *freq*)

Returns a pointer to the solution object at the specified wave direction and frequency.

Parameters

dir	Integer. Index of the wave direction desired.
freq	Integer. Index of the wave frequency desired.

Returns

Returns a pointer to the Solution object at the specified wave direction and wave frequency. Returned variable is passed by reference.

Definition at line 58 of file solutionset.cpp.

15.35.3.5 void SolutionSet::resize (int *dir*, int *freq*)

Resizes the 2-D array of Solution objects. Any existing objects in the array are preserved. If the array is sized smaller than the existing number of Solution objects, any object beyond the new index range are deleted.

Parameters

dir	The new number of wave directions to resize the array to.
freq	The new number of wave frequencies to resize the array to.

Definition at line 91 of file solutionset.cpp.

15.35.3.6 void SolutionSet::setSolnMat (int *dir*, int *freq*, Solution *soln*)

Sets the solution object at the specified wave direction and frequency.

Parameters

dir	Integer. Index of the wave direction desired.
freq	Integer. Index of the wave frequency desired.
soln	The solution object to pass in.

Definition at line 65 of file solutionset.cpp.

15.35.3.7 vector< int > SolutionSet::size ( )

Returns the size of the matrix as a vector of two elements.

Returns

Integer. Returns the size of the matrix as a vector of two elements.

Definition at line 102 of file solutionset.cpp.

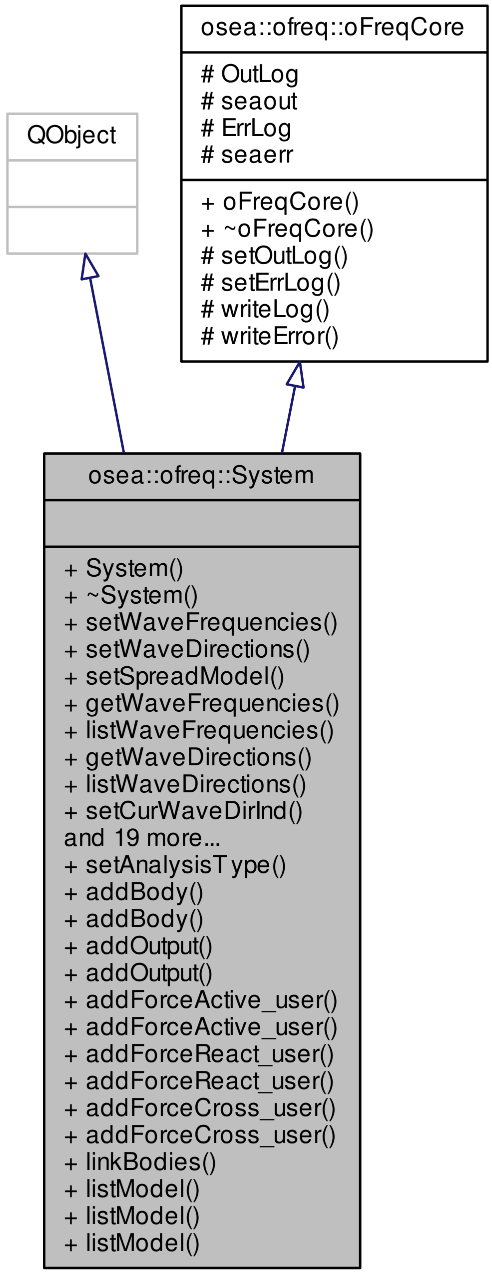
The documentation for this class was generated from the following files:

- bin/ofreq/global_objects/solutionset.h
- bin/ofreq/global_objects/solutionset.cpp

15.36 osea::ofreq::System Class Reference

```
#include <system.h>
```

Inheritance diagram for osea::ofreq::System:



Public Slots

- void setAnalysisType (std::string)
- void addBody (Body input)

  Adds another Body object to the list of Body objects. Sets the new Body object equal to the input.
- void addBody ()

  Adds another Body object to the list of Body objects. Uses a blank new Body object.

- void addOutput (OutputsBody input)

  Adds another OutputsBody object to the list of OutputsBody objects. Sets the object equal to the input.
- void addOutput ()

  Adds another OutputsBody object to the list of OutputsBody objects. Uses a blank new OutputsBody object.
- void addForceActive_user (ForceActive input)

  Adds another ForceActive object to the list of forceActive_user objects. Sets the object equal to the input.
- void addForceActive_user ()

  Adds another ForceActive Object to the list of forceActive_user objects. Uses a blank new ForceActive object.
- void addForceReact_user (ForceReact input)

  Adds another ForceReact object to the list of forceReact_user objects. Sets the object equal to the input.
- void addForceReact_user ()

  Adds another ForceReact Object to the list of forceReact_user objects. Uses a blank new ForceReact object.
- void addForceCross_user (ForceCross input)

  Adds another ForceCross object to the list of forceCross_user objects. Sets the object equal to the input.
- void addForceCross_user ()

  Adds another ForceCross Object to the list of forceCross_user objects. Uses a blank new ForceCross object.
- void linkBodies (int bodID)

  This converts the cross-body force links from simple identification by name into actual pointers to each body. The linkBodies command must exist so that all bodies can be read into the program before linking takes place. This command searches through the list of bodies to get a matching body name and creates a pointer link to that body. At the end, it clears the list of names for linked bodies, to reduce memory requirements.
- std::vector< MotionModel ∗ > & listModel ()

  Provides access to the full list of motion models.
- ofreq::MotionModel & listModel (unsigned int index)

  Provides access to one item in the list of motion models.
- ofreq::MotionModel & listModel (std::string modelName)

  Provides access to one item in the list of motion models.

Signals

- void ReferenceSystem (System ∗mySystem)

  Returns the System object for information. Used mainly to access the list of forced defined for the system object.

Public Member Functions

- System ()
- virtual ∼System ()
- void setWaveFrequencies (std::vector< double >)
- void setWaveDirections (std::vector< double >)
- void setSpreadModel (std::string)
- std::vector< double > getWaveFrequencies ()
- std::vector< double > & listWaveFrequencies ()

  Provides direct access to the list of wave frequencies.
- std::vector< double > getWaveDirections ()
- std::vector< double > & listWaveDirections ()

  Provides direct access to the list of wave directions.
- void setCurWaveDirInd (int input)

  Sets the current wave index.
- void setCurFreqInd (int input)

  Sets the current frequency index.
- int getCurWaveDirInd ()

Returns the current wave direction index.

- int getCurFreqInd ()

  Returns the current wave frequency index.
- double getCurWaveDir ()

  Returns the current wave direction. Actual value of wave direction angle. Angle specified as radians with zero as True North, positive counter-clockwise.
- double getCurFreq ()

  Returns the current wave frequency. Actual value of the wave frequency. Value specified with units of radians per second.
- std::vector< Body > & listBody ()

  Returns direct access to the list of Body objects. Includes all the properties included by a std::vector<> class.
- Body & listBody (int input)

  Returns direct access to a single Body object.
- std::vector< OutputsBody > & listOutput ()

  Returns direct access to the list of OutputsBody objects. Includes all the properties included by a std::vector<> class.
- OutputsBody & listOutput (int input)

  Returns direct access to a single OutputsBody object.
- void clearForce (std::string forceClass="")

  Clears the vector of force objects for the specified force type. This is useful to free system memory. Force type is specified by a std::string input. If no std::string input is supplied, all force objects are cleared from the system object. All force objects should already be copied to their respective Body objects before issuing this function.
- ForceActive ∗ refForceActive_user (std::string forceName)

  Gets the forceActive_user object referenced by the name specified in the input. The Bodies input file will define the forceActive_user object by a name. This uses that name to retrieve the forceActive_user object.
- std::vector< ForceActive > & listForceActive_user ()

  Exposes the vector of forceActive_user objects. Provides direct access to the vector.
- ForceActive & listForceActive_user (unsigned int forceIndex)

  Returns reference to the forceActive_user object referenced by the index specified in the input.
- ForceReact ∗ refForceReact_user (std::string forceName)

  Gets the forceReact_user object referenced by the name specified in the input. The Bodies input file will define the forceReact_user object by a name. This uses that name to retrieve the forceReact_user object.
- std::vector< ForceReact > & listForceReact_user ()

  Exposes the vector of forceReact_user objects. Provides direct access to the vector.
- ForceReact & listForceReact_user (unsigned int forceIndex)

  Returns reference to the forceReact_user object referenced by the index specified in the input.
- ForceCross ∗ refForceCross_user (std::string forceName)

  Gets the forceCross_user object referenced by the name specified in the input. The Bodies input file will define the forceCross_user object by a name. This uses that name to retrieve the forceCross_user object.
- std::vector< ForceCross > & listForceCross_user ()

  Exposes the vector of forceCross_user objects. Provides direct access to the vector.
- ForceCross & listForceCross_user (unsigned int forceIndex)

  Returns reference to the forceCross_user object referenced by the index specified in the input.

Additional Inherited Members

15.36.1 Detailed Description

This class holds data for the system object. The system object controls the overall behavior of the program. It also decides which analysis type to run: motion or resonant frequency. The system object controls the current wave environment settings.

Definition at line 98 of file system.h.

15.36.2 Constructor & Destructor Documentation

15.36.2.1 System::System ( )

The default constructor.

Definition at line 42 of file system.cpp.

15.36.2.2 System::∼System ( ) [virtual]

The default destructor, clears any dynamic memory.

Definition at line 49 of file system.cpp.

15.36.3 Member Function Documentation

15.36.3.1 void System::addBody (**Body** *input*) [slot]

Adds another Body object to the list of Body objects. Sets the new Body object equal to the input.

Parameters

input	Body object to add into the list of stored Body objects. Variable is passed by value and stored independant inside the System class.

See Also

Body

Definition at line 336 of file system.cpp.

15.36.3.2 void System::addBody ( ) [slot]

Adds another Body object to the list of Body objects. Uses a blank new Body object.

See Also

Body

Definition at line 345 of file system.cpp.

15.36.3.3 void System::addForceActive_user (**ForceActive** *input*) [slot]

Adds another ForceActive object to the list of forceActive_user objects. Sets the object equal to the input.

Parameters

input	ForceActive object to add to the list of forceActive_user objects. Variable is passed by value and stored independant inside the System class.

See Also

ForceActive

Definition at line 366 of file system.cpp.

15.36.3.4 void System::addForceActive_user ( ) [slot]

Adds another ForceActive Object to the list of forceActive_user objects. Uses a blank new ForceActive object.

See Also

ForceActive

Definition at line 372 of file system.cpp.

15.36.3.5 void System::addForceCross_user (ForceCross *input*) [slot]

Adds another ForceCross object to the list of forceCross_user objects. Sets the object equal to the input.

Parameters

input	ForceCross object to add to the list of forceCross_user objects. Variable is passed by value and stored independant inside the System class.

See Also

ForceCross

Definition at line 390 of file system.cpp.

15.36.3.6 void System::addForceCross_user ( ) [slot]

Adds another ForceCross Object to the list of forceCross_user objects. Uses a blank new ForceCross object.

See Also

ForceCross

Definition at line 396 of file system.cpp.

15.36.3.7 void System::addForceReact_user (ForceReact *input*) [slot]

Adds another ForceReact object to the list of forceReact_user objects. Sets the object equal to the input.

Parameters

input	ForceReact object to add to the list of forceReact_user objects. Variable is passed by value and stored independant inside the System class.

See Also

ForceReact

Definition at line 378 of file system.cpp.

15.36.3.8 void System::addForceReact_user ( ) [slot]

Adds another ForceReact Object to the list of forceReact_user objects. Uses a blank new ForceReact object.

See Also

ForceReact

Definition at line 384 of file system.cpp.

15.36.3.9 void System::addOutput (OutputsBody *input*) [slot]

Adds another OutputsBody object to the list of OutputsBody objects. Sets the object equal to the input.

Parameters

input	OutputsBody object to add into the list of stored OutputsBody objects. Variable is passed by value and sotred independant inside the System class.

See Also

OutputsBody

Definition at line 354 of file system.cpp.

15.36.3.10 void System::addOutput ( ) [slot]

Adds another OutputsBody object to the list of OutputsBody objects. Uses a blank new OutputsBody object.

See Also

OutputsBody

Definition at line 360 of file system.cpp.

15.36.3.11 void System::clearForce (std::string *forceClass* = " ")

Clears the vector of force objects for the specified force type. This is useful to free system memory. Force type is specified by a std::string input. If no std::string input is supplied, all force objects are cleared from the system object. All force objects should already be copied to their respective Body objects before issuing this function.

Parameters

forceClass	std::string input designating which force object type to clear. Valid values are: ForceActive: Clears the ForceActive class of objects. ForceReact: Clears the ForceReact class of objects. ForceCross: Clears the ForceCross class of objects. "": Clears all three object classes of objects. other: If an unknown input is encountered, no objects are cleared.

Definition at line 162 of file system.cpp.

15.36.3.12 double System::getCurFreq ( )

Returns the current wave frequency. Actual value of the wave frequency. Value specified with units of radians per second.

Returns

Double. Returns the current wave frequency. Actual value of the wave frequency. Value specified with units of radians per second.

Definition at line 132 of file system.cpp.

15.36.3.13 int System::getCurFreqInd ( )

Returns the current wave frequency index.

Returns

Integer. Returns the current wave frequency index. Variable passed by value.

Definition at line 120 of file system.cpp.

15.36.3.14 double System::getCurWaveDir ( )

Returns the current wave direction. Actual value of wave direction angle. Angle specified as radians with zero as True North, positive counter-clockwise.

Returns

Double. Returns the current wave direction. Actual value of wave direction angle. Angle specified as radians with zero as True North, positive counter-clockwise. Variable passed by value.

Definition at line 126 of file system.cpp.

15.36.3.15 int System::getCurWaveDirInd ( )

Returns the current wave direction index.

Returns

Integer. Returns the current wave direction index. Variable passed by value.

Definition at line 114 of file system.cpp.

15.36.3.16 vector< double > System::getWaveDirections ( )

Retrieve the list of wave directions.

Returns

The list of wave directions.

Definition at line 90 of file system.cpp.

15.36.3.17 vector< double > System::getWaveFrequencies ( )

Retrieve the list of wave frequencies.

Returns

The list of wave frequencies.

Definition at line 78 of file system.cpp.

15.36.3.18 void System::linkBodies (int *bodID*) [slot]

This converts the cross-body force links from simple identification by name into actual pointers to each body. The linkBodies command must exist so that all bodies can be read into the program before linking takes place. This command searches through the list of bodies to get a matching body name and creates a pointer link to that body. At the end, it clears the list of names for linked bodies, to reduce memory requirements.

Parameters

bodID	An integer variable that describes the base body which the function should process all links for. parameter passed by value.

See Also

Body::listNamedLink_user()
Body::listNamedLink_hydro();

Definition at line 402 of file system.cpp.

15.36.3.19 vector< Body > & System::listBody ( )

Returns direct access to the list of Body objects. Includes all the properties included by a std::vector<> class.

Returns

Returns a vector of Body objects. Returned variable passed by reference.

See Also

Body

Definition at line 138 of file system.cpp.

15.36.3.20 Body & System::listBody (int *input*)

Returns direct access to a single Body object.

Parameters

input	Specifies index of which Body object to access in the list of Body objects.

Returns

Returns a Body object. Returned variable is passed by reference.

Definition at line 144 of file system.cpp.

15.36.3.21 vector< ForceActive > & System::listForceActive_user ( )

Exposes the vector of forceActive_user objects. Provides direct access to the vector.

Returns

Returns a reference to the vector of forceActive_user objects. Variable passed by reference.

Definition at line 216 of file system.cpp.

15.36.3.22 ForceActive & System::listForceActive_user (unsigned int *forceIndex*)

Returns reference to the forceActive_user object referenced by the index specified in the input.

Parameters

forceIndex	Integer variable which defines the index of which forceActive_user object to retrieve. Variable passed by reference.

Returns

Returns a reference to the ForceActive_user object. Variable passed by reference

Definition at line 223 of file system.cpp.

15.36.3.23 vector< ForceCross > & System::listForceCross_user ( )

Exposes the vector of forceCross_user objects. Provides direct access to the vector.

Returns

Returns a reference to the vector of forceCross_user objects. Variable passed by reference.

Definition at line 308 of file system.cpp.

15.36.3.24 ForceCross & System::listForceCross_user (unsigned int *forceIndex*)

Returns reference to the forceCross_user object referenced by the index specified in the input.

Parameters

forceIndex	Integer variable which defines the index of which forceCross_user object to retrieve. Variable passed by value.

Returns

Returns a reference to the ForceCross_user object. Variable passed by reference

Definition at line 314 of file system.cpp.

15.36.3.25 vector< ForceReact > & System::listForceReact_user ( )

Exposes the vector of forceReact_user objects. Provides direct access to the vector.

Returns

Returns a reference to the vector of forceReact_user objects. Variable passed by reference.

Definition at line 263 of file system.cpp.

15.36.3.26 ForceReact & System::listForceReact_user (unsigned int *forceIndex*)

Returns reference to the forceReact_user object referenced by the index specified in the input.

Parameters

forceIndex	Integer variable which defines the index of which forceReact_user object to retrieve. Variable passed by value.

Returns

Returns a reference to the ForceReact_user object. Variable passed by reference

Definition at line 269 of file system.cpp.

15.36.3.27 std::vector< MotionModel * > & System::listModel ( ) [slot]

Provides access to the full list of motion models.

Returns a vector to the full list of motion models. The list of motion models are all the various model classes available to the system at run time. Each object in the vector is a different class, but all classes in the vector are derived from the MotionModel class.

See Also

MotionModel

Returns

Returns a vector of objects of different types. Each object is derived from the MotionModel class. Returned variable is passed by reference.

Definition at line 446 of file system.cpp.

15.36.3.28 MotionModel & System::listModel (unsigned int *index*) [slot]

Provides access to one item in the list of motion models.

Returns a single MotionModel based object. Each model will be a different class, but all classes are derived from the MotionModel object. Each object in the vector is a different class of motion model.

Parameters

index	Integer. The index of which item in the vector you want.

Returns

Returns an object of a class derived from the MotionModel class. Returned variable is passed by reference.

Definition at line 452 of file system.cpp.

15.36.3.29 MotionModel & System::listModel (std::string *modelName*) [slot]

Provides access to one item in the list of motion models.

Returns a single MotionModel based object. Each model will be a different class, but all classes are derived from the MotionModel object. Each object in the vector is a different class of motion model.

Parameters

modelName	String input. The name of the motion model, as specified by the user. Must match the predefined name of the model exactly.

Returns

Returns an object of a class derived from the MotionModel class. Returned variable is passed by reference.

Definition at line 458 of file system.cpp.

15.36.3.30 vector< OutputsBody > & System::listOutput ( )

Returns direct access to the list of OutputsBody objects. Includes all the properties included by a std::vector<> class.

Returns

Returns a vector of OutputsBody objects. Returned variable passed by reference.

See Also

OutputsBody

Definition at line 150 of file system.cpp.

15.36.3.31 OutputsBody & System::listOutput (int *input*)

Returns direct access to a single OutputsBody object.

Parameters

input	Specifies the index of which OutputsBody object to access in the list of OutputsBody objects.

Returns

Returns an OutputsBody object. Returned variable is passed by reference.

Definition at line 156 of file system.cpp.

15.36.3.32 vector< double > & System::listWaveDirections ( )

Provides direct access to the list of wave directions.

Returns

Pointer to the list of wave directions Variable passed by reference.

Definition at line 96 of file system.cpp.

15.36.3.33 vector< double > & System::listWaveFrequencies ( )

Provides direct access to the list of wave frequencies.

Returns

Pointer to the list of wave frequencies. Variable passed by reference.

Definition at line 84 of file system.cpp.

15.36.3.34 void osea::ofreq::System::ReferenceSystem (System ∗ *mySystem*) [signal]

Returns the System object for information. Used mainly to access the list of forced defined for the system object.

Parameters

Returns	a pointer to the System object. Variable passed by reference.

15.36.3.35 ForceActive ∗ System::refForceActive_user (std::string *forceName*)

Gets the forceActive_user object referenced by the name specified in the input. The Bodies input file will define the forceActive_user object by a name. This uses that name to retrieve the forceActive_user object.

Parameters

forceName	std::string input. Variable passed by value. The name of the forceActive_user object. Must match exactly what is defined in the Forces.in input file which defines the forceActive_user object.

Returns

Returns a pointer to the forceActive_user object as requested. Pointer passed by value.

Definition at line 190 of file system.cpp.

15.36.3.36 ForceCross ∗ System::refForceCross_user (std::string *forceName*)

Gets the forceCross_user object referenced by the name specified in the input. The Bodies input file will define the forceCross_user object by a name. This uses that name to retrieve the forceCross_user object.

Parameters

forceName	std::string input. Variable passed by value. The name of the forceCross_user object. Must match exactly what is defined in the Forces.in input file which defines the forceCross_user object.

Returns

Returns a pointer to the forceCross_user object as requested. Pointer passed by value.

Definition at line 282 of file system.cpp.

15.36.3.37 ForceReact ∗ System::refForceReact_user (std::string *forceName*)

Gets the forceReact_user object referenced by the name specified in the input. The Bodies input file will define the forceReact_user object by a name. This uses that name to retrieve the forceReact_user object.

Parameters

forceName	std::string input. Variable passed by value. The name of the forceReact_user object. Must match exactly what is defined in the Forces.in input file which defines the forceReact_user object.

Returns

Returns a pointer to the forceReact_user object as requested. Pointer passed by value.

Definition at line 237 of file system.cpp.

15.36.3.38 void System::setAnalysisType (std::string) [slot]

Sets the analysis ype.

Parameters

analysisTypeIn	The analysis type.

Definition at line 330 of file system.cpp.

15.36.3.39 void System::setCurFreqInd (int *input*)

Sets the current frequency index.

Parameters

input	Integer specifying the index of the current wave frequency in the list of wave frequencies.

Definition at line 108 of file system.cpp.

15.36.3.40 void System::setCurWaveDirInd (int *input*)

Sets the current wave index.

Parameters

input	Integer specifying the index of the current wave direction in the list of wave directions.

Definition at line 102 of file system.cpp.

15.36.3.41 void System::setSpreadModel (std::string)

Sets the spread model.

Parameters

spreadIn	The spread model.

Definition at line 72 of file system.cpp.

15.36.3.42 void System::setWaveDirections (std::vector< double >)

Sets the wave directions.

Parameters

vecIn	The list of wave directions.

Definition at line 66 of file system.cpp.

15.36.3.43 void System::setWaveFrequencies (std::vector< double >)

Sets the wave frequencies.

Parameters

vecIn	The list of wave frequencies.

Definition at line 60 of file system.cpp.

The documentation for this class was generated from the following files:

- bin/ofreq/global_objects/system.h
- bin/ofreq/global_objects/system.cpp

Chapter 16

File Documentation

16.1 100_doc/130_doc_developer/CodingFormatStandards.dox File Reference

16.2 100_doc/130_doc_developer/CodingStandard/CPP_Comments.dox File Reference

16.3 100_doc/130_doc_developer/CodingStandard/HeaderComment.dox File Reference

16.4 100_doc/130_doc_developer/CodingStandard/ObjectParadigm.dox File Reference

16.5 100_doc/130_doc_developer/CodingStandard/Qt_Platform_Code.dox File Reference

16.6 100_doc/130_doc_developer/InputFormatting.dox File Reference

16.7 100_doc/130_doc_developer/Inputs/InputSyntax.dox File Reference

16.8 100_doc/130_doc_developer/Inputs/InputValues.dox File Reference

16.9 100_doc/130_doc_developer/Inputs/WhyTextFiles.dox File Reference

16.10 100_doc/130_doc_developer/mainpage.dox File Reference

16.11 100_doc/130_doc_developer/ProgramExecution.dox File Reference

16.12 100_doc/130_doc_developer/UML_Process.dox File Reference

16.13 100_doc/130_doc_developer/UML_Process/UML_FileReader.dox File Reference

16.14 100_doc/130_doc_developer/UML_Process/UML_MotionModel.dox File Reference

16.15 100_doc/130_doc_developer/UML_Process/UML_MotionSolver.dox File Reference

16.16 100_doc/130_doc_developer/UML_Process/UML_OutputsCalculation.dox File Reference

16.17 100_doc/130_doc_developer/UML_Process/UML_Overall.dox File Reference

16.18 100_doc/130_doc_developer/UML_Process/UML_ResonantSolver.dox File Reference

16.19 100_doc/130_doc_developer/UML_Process/UML_WaveCalculation.dox File Reference

16.20 100_doc/130_doc_developer/UML_Process/UML_WritingFiles.dox File Reference

16.21 100_doc/130_doc_developer/validation.dox File Reference

16.22 100_doc/130_doc_developer/Validation/MultiBodyTest1.dox File Reference

16.23 100_doc/130_doc_developer/Validation/MultiBodyTest2.dox File Reference

16.24 100_doc/130_doc_developer/Validation/MultiBodyTest3.dox File Reference

16.25 100_doc/130_doc_developer/Validation/MultiBodyTest4.dox File Reference

16.26 100_doc/130_doc_developer/Validation/SimpleTest1.dox File Reference

16.27 100_doc/130_doc_developer/Validation/SimpleTest2.dox File Reference

16.28 100_doc/130_doc_developer/Validation/SimpleTest3.dox File Reference

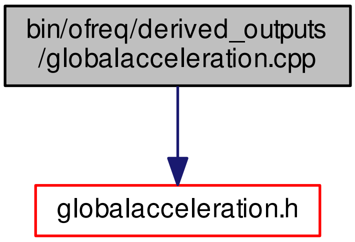
16.29 100_doc/130_doc_developer/Validation/SimpleTest4.dox File Reference

16.30 100_doc/130_doc_developer/Validation/TestFrequency.dox File Reference

16.31 bin/ofreq/derived_outputs/globalacceleration.cpp File Reference

```
#include "globalacceleration.h"
```

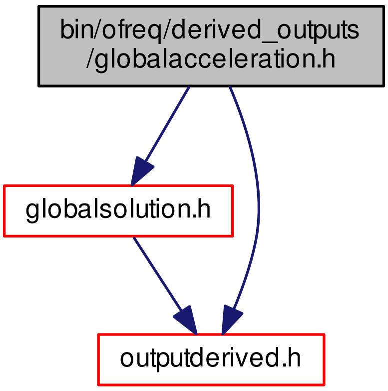
Include dependency graph for globalacceleration.cpp:



16.32 bin/ofreq/derived_outputs/globalacceleration.h File Reference

```
#include "globalsolution.h"
#include "outputderived.h"
```

Include dependency graph for globalacceleration.h:



Classes

- class osea::ofreq::GlobalAcceleration

Namespaces

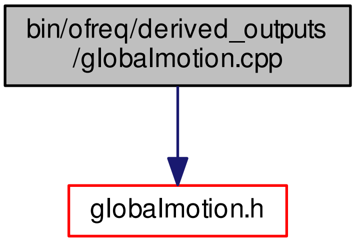
- namespace osea

- namespace osea::ofreq

16.33 bin/ofreq/derived_outputs/globalmotion.cpp File Reference

```
#include "globalmotion.h"
```

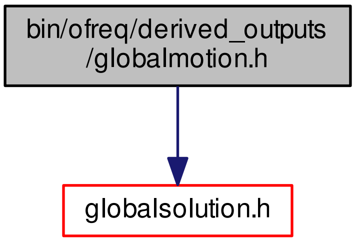
Include dependency graph for globalmotion.cpp:



16.34 bin/ofreq/derived_outputs/globalmotion.h File Reference

```
#include "globalsolution.h"
```

Include dependency graph for globalmotion.h:



Classes

- class osea::ofreq::GlobalMotion

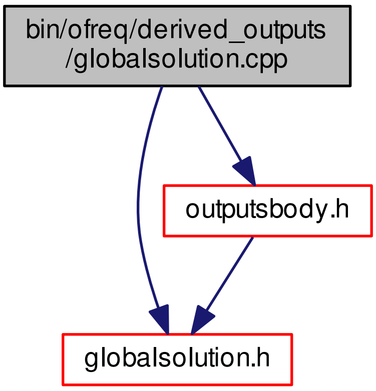
Namespaces

- namespace osea
- namespace osea::ofreq

16.35 bin/ofreq/derived_outputs/globalsolution.cpp File Reference

```
#include "globalsolution.h"
#include "outputsbody.h"
```

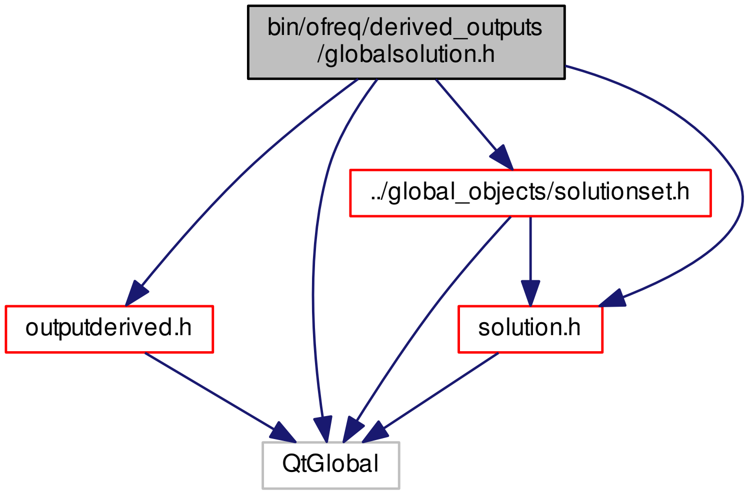
Include dependency graph for globalsolution.cpp:



16.36 bin/ofreq/derived_outputs/globalsolution.h File Reference

```
#include "outputderived.h"
#include <QtGlobal>
#include "../global_objects/solutionset.h"
#include "../global_objects/solution.h"
```

Include dependency graph for globalsolution.h:



Classes

- class osea::ofreq::GlobalSolution

Namespaces

- namespace osea
- namespace osea::ofreq

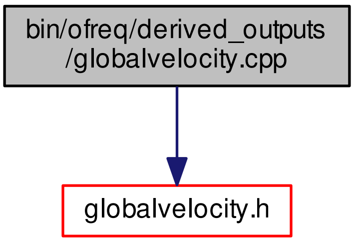
Typedefs

- typedef std::complex$<$ double $>$ osea::ofreq::complexDouble

16.37 bin/ofreq/derived_outputs/globalvelocity.cpp File Reference

```
#include "globalvelocity.h"
```

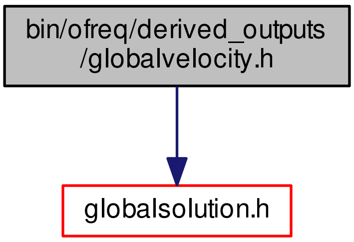
Include dependency graph for globalvelocity.cpp:



16.38 bin/ofreq/derived_outputs/globalvelocity.h File Reference

```
#include "globalsolution.h"
```

Include dependency graph for globalvelocity.h:



Classes

- class osea::ofreq::GlobalVelocity

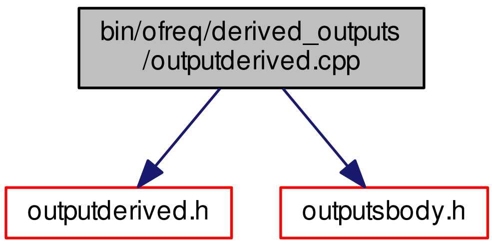
Namespaces

- namespace osea
- namespace osea::ofreq

16.39 bin/ofreq/derived_outputs/outputderived.cpp File Reference

```
#include "outputderived.h"
#include "outputsbody.h"
```

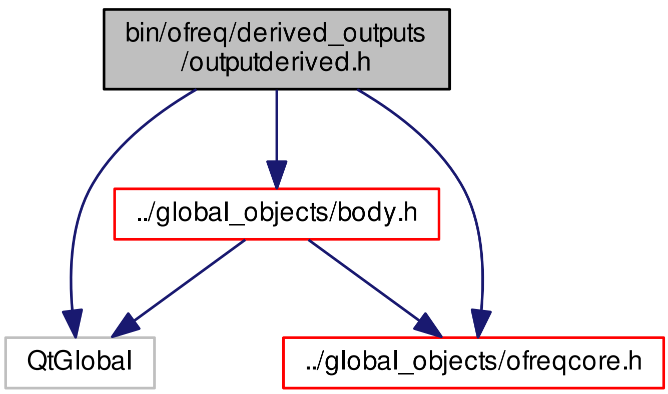
Include dependency graph for outputderived.cpp:



16.40 bin/ofreq/derived_outputs/outputderived.h File Reference

```
#include <QtGlobal>
#include "../global_objects/body.h"
#include "../global_objects/ofreqcore.h"
```

Include dependency graph for outputderived.h:



Classes

- class osea::ofreq::OutputDerived

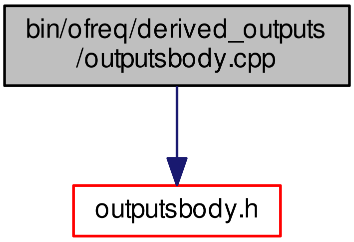
Namespaces

- namespace osea
- namespace osea::ofreq

16.41 bin/ofreq/derived_outputs/outputsbody.cpp File Reference

```
#include "outputsbody.h"
```

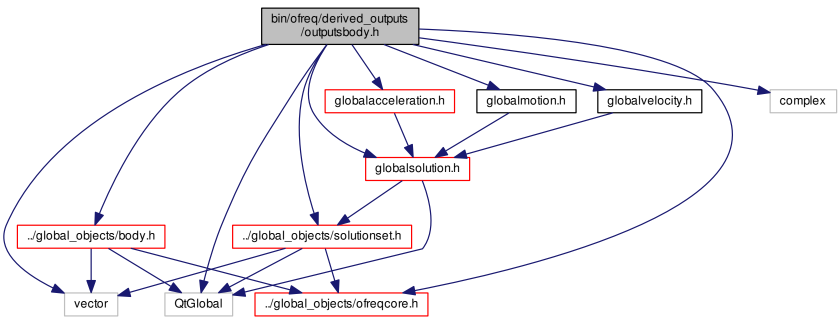
Include dependency graph for outputsbody.cpp:



16.42 bin/ofreq/derived_outputs/outputsbody.h File Reference

```
#include <vector>
#include <complex>
#include <QtGlobal>
#include "globalsolution.h"
#include "globalacceleration.h"
#include "globalmotion.h"
#include "globalvelocity.h"
#include "../global_objects/solutionset.h"
#include "../global_objects/body.h"
#include "../global_objects/ofreqcore.h"
```

Include dependency graph for outputsbody.h:



Classes

- class osea::ofreq::OutputsBody

Namespaces

- namespace osea
- namespace osea::ofreq

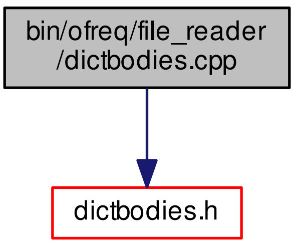
Typedefs

- typedef std::vector
  < std::complex< double > > osea::ofreq::cx_vector

    Type definition for a vector of complex numbers. Used to return the calculated output.

16.43 bin/ofreq/file_reader/dictbodies.cpp File Reference

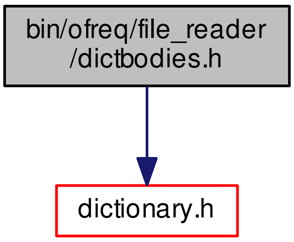
`#include "dictbodies.h"`
Include dependency graph for dictbodies.cpp:



16.44 bin/ofreq/file_reader/dictbodies.h File Reference

```
#include "dictionary.h"
```

Include dependency graph for dictbodies.h:



Classes

- class osea::ofreq::dictBodies

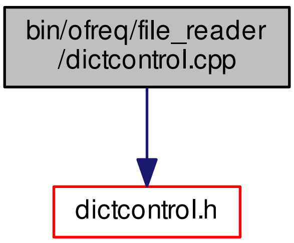
Namespaces

- namespace osea
- namespace osea::ofreq

16.45 bin/ofreq/file_reader/dictcontrol.cpp File Reference

```
#include "dictcontrol.h"
```

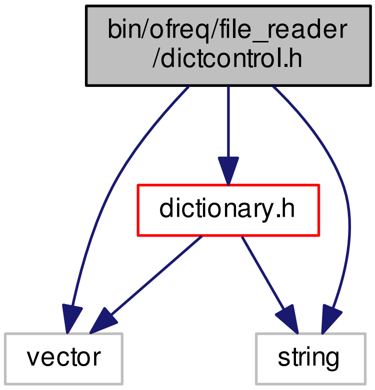
Include dependency graph for dictcontrol.cpp:



16.46 bin/ofreq/file_reader/dictcontrol.h File Reference

```
#include <vector>
#include <string>
#include "dictionary.h"
```

Include dependency graph for dictcontrol.h:



Classes

- class osea::ofreq::dictControl

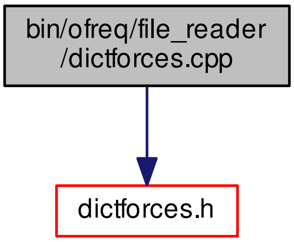
Namespaces

- namespace osea
- namespace osea::ofreq

16.47 bin/ofreq/file_reader/dictforces.cpp File Reference

```
#include "dictforces.h"
```

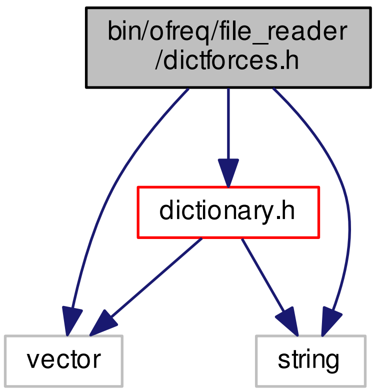
Include dependency graph for dictforces.cpp:



16.48 bin/ofreq/file_reader/dictforces.h File Reference

```
#include <vector>
#include <string>
#include "dictionary.h"
```

Include dependency graph for dictforces.h:



Classes

- class osea::ofreq::dictForces

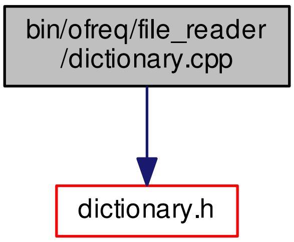
Namespaces

- namespace osea
- namespace osea::ofreq

16.49 bin/ofreq/file_reader/dictionary.cpp File Reference

```
#include "dictionary.h"
```

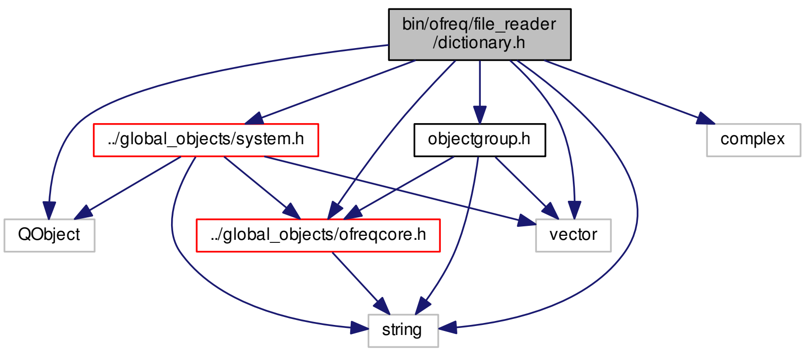
Include dependency graph for dictionary.cpp:



16.50 bin/ofreq/file_reader/dictionary.h File Reference

```
#include <QObject>
#include "objectgroup.h"
#include "../global_objects/system.h"
#include "../global_objects/ofreqcore.h"
#include <complex>
#include <vector>
#include <string>
```

Include dependency graph for dictionary.h:



Classes

- class osea::Dictionary

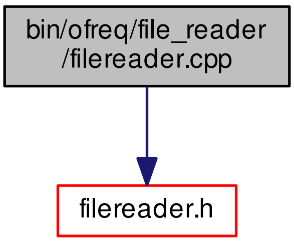
Namespaces

- namespace osea

16.51 bin/ofreq/file_reader/filereader.cpp File Reference

```
#include "filereader.h"
```

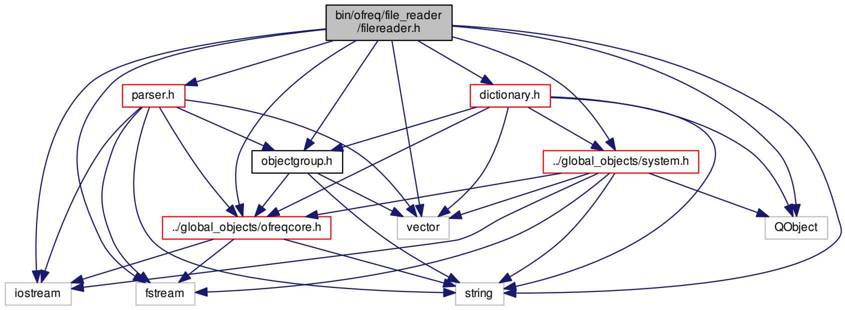
Include dependency graph for filereader.cpp:



16.52 bin/ofreq/file_reader/filereader.h File Reference

```
#include <QObject>
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
#include "parser.h"
#include "objectgroup.h"
#include "../global_objects/system.h"
#include "../global_objects/ofreqcore.h"
#include "dictionary.h"
```

Include dependency graph for filereader.h:



Classes

- class osea::FileReader

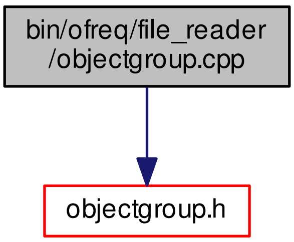
Namespaces

- namespace osea

16.53 bin/ofreq/file_reader/objectgroup.cpp File Reference

```
#include "objectgroup.h"
```

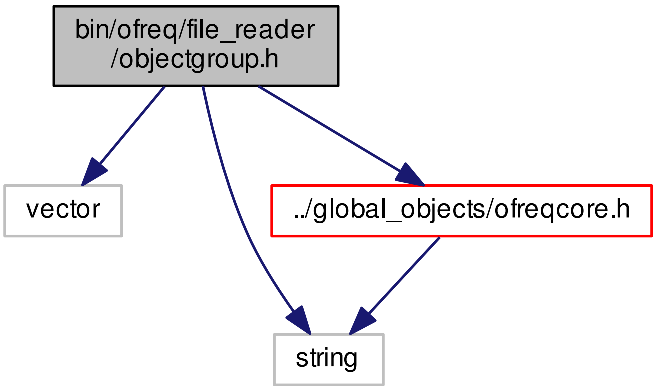
Include dependency graph for objectgroup.cpp:



16.54 bin/ofreq/file_reader/objectgroup.h File Reference

```
#include <vector>
#include <string>
#include "../global_objects/ofreqcore.h"
```

Include dependency graph for objectgroup.h:



Classes

- class osea::ObjectGroup

  The ObjectGroup class contains groupings of object definitions captured from an input file. It is a data container to hold the segmented input file for interpretation. The container contains three things: 1.) Object class name (as specified by input file) 2.) Vector of keyword names 3.) Vector of keyword values. Each entry in the vector of values is also a vector. This allows the definition of lists. A list will be as long as it needs to be for specification of all values in the list. The index of the value is specified by its position in the vector list. The value is the entry.

Namespaces

- namespace osea

Typedefs

- typedef std::vector
  $<$ std::vector$<$ std::string $>$ $>$ osea::vecValue

  Type definition used to store key values. Must be a vector of vectors because a value may also be a list of values.

- typedef std::vector$<$ std::string $>$ osea::vecKeyword

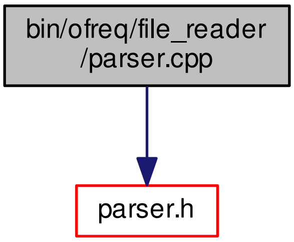
  Type defintion used to store keywords.

- typedef std::vector
  $<$ ObjectGroup $*$ $>$ osea::vecObject

16.55 bin/ofreq/file_reader/parser.cpp File Reference

```
#include "parser.h"
```

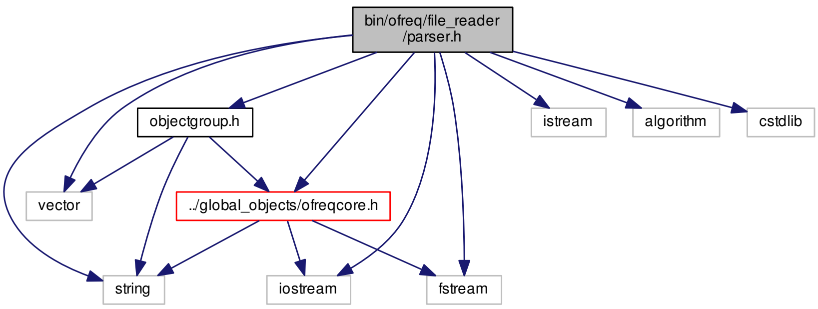
Include dependency graph for parser.cpp:



16.56 bin/ofreq/file_reader/parser.h File Reference

```
#include <vector>
#include <string>
#include <istream>
#include <iostream>
#include <algorithm>
#include <cstdlib>
#include <fstream>
#include "objectgroup.h"
#include "../global_objects/ofreqcore.h"
```

Include dependency graph for parser.h:



Classes

- class osea::Parser

  The Parser class takes an input segment of strings and segments that segment. It strips out comments. It recognizes quotation marks and groups those segments together. Parser finally returns a series of ObjectGroup objects. Each ObjectGroup contains the classname and a list of keyword value pairs.

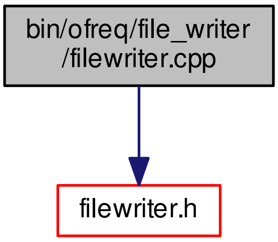
Namespaces

- namespace osea

16.57 bin/ofreq/file_writer/filewriter.cpp File Reference

```
#include "filewriter.h"
```

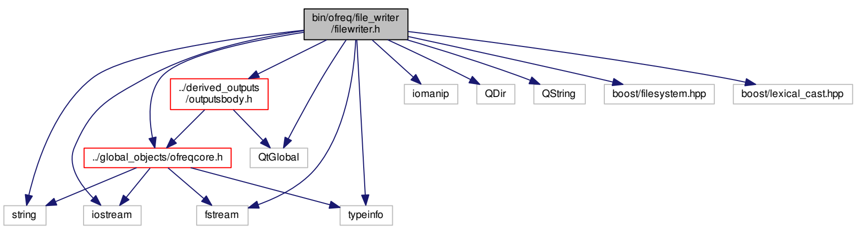
Include dependency graph for filewriter.cpp:



16.58 bin/ofreq/file_writer/filewriter.h File Reference

```
#include <string>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <typeinfo>
#include <QtGlobal>
#include <QDir>
#include <QString>
#include "../derived_outputs/outputsbody.h"
#include <boost/filesystem.hpp>
#include <boost/lexical_cast.hpp>
#include "../global_objects/ofreqcore.h"
```

Include dependency graph for filewriter.h:



Classes

- class osea::ofreq::FileWriter

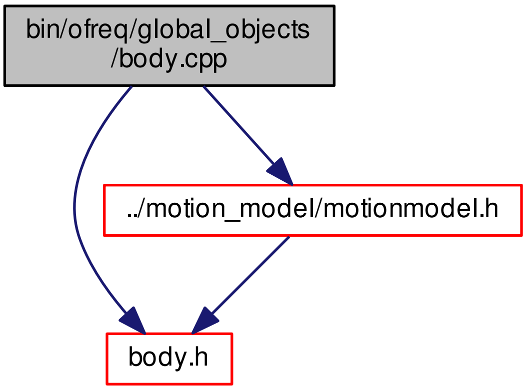
Namespaces

- namespace osea
- namespace osea::ofreq

16.59 bin/ofreq/global_objects/body.cpp File Reference

```
#include "body.h"
#include "../motion_model/motionmodel.h"
```

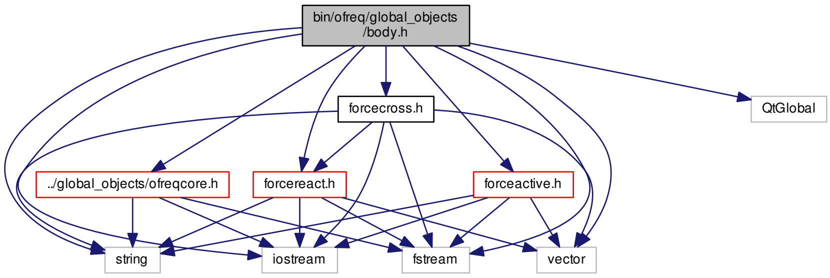
Include dependency graph for body.cpp:



16.60 bin/ofreq/global_objects/body.h File Reference

```
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
#include <QtGlobal>
#include "forceactive.h"
#include "forcecross.h"
#include "forcereact.h"
#include "../global_objects/ofreqcore.h"
```

Include dependency graph for body.h:



Classes

- class osea::ofreq::Body

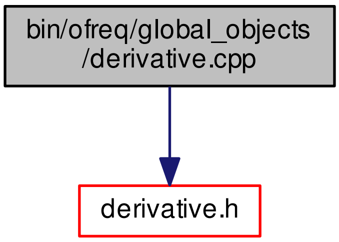
Namespaces

- namespace osea
- namespace osea::ofreq

16.61 bin/ofreq/global_objects/derivative.cpp File Reference

```
#include "derivative.h"
```

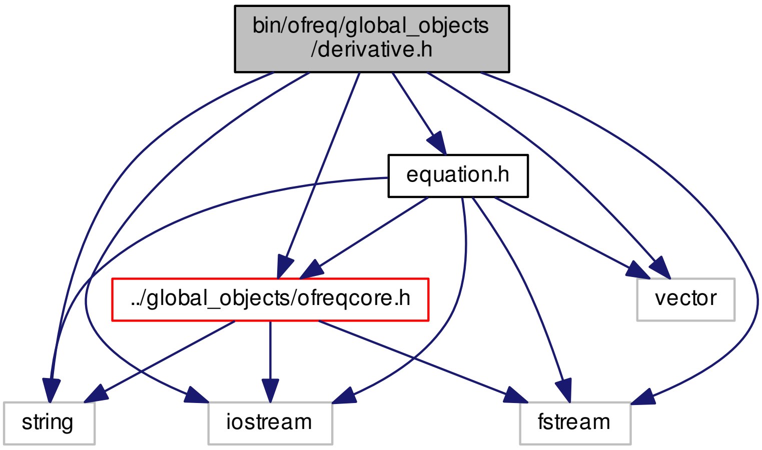
Include dependency graph for derivative.cpp:



16.62 bin/ofreq/global_objects/derivative.h File Reference

```
#include "equation.h"
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
#include "../global_objects/ofreqcore.h"
```

Include dependency graph for derivative.h:



Classes

- class osea::ofreq::Derivative

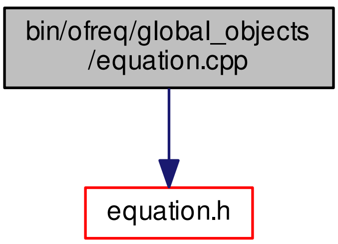
Namespaces

- namespace osea
- namespace osea::ofreq

16.63 bin/ofreq/global_objects/equation.cpp File Reference

`#include "equation.h"`

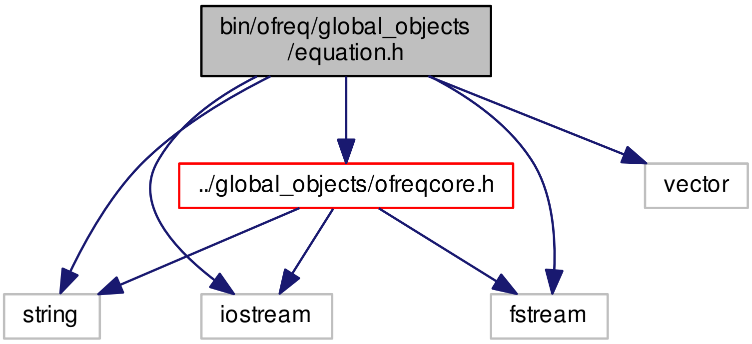
Include dependency graph for equation.cpp:



16.64 bin/ofreq/global_objects/equation.h File Reference

```
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
#include "../global_objects/ofreqcore.h"
```

Include dependency graph for equation.h:



Classes

- class osea::ofreq::Equation

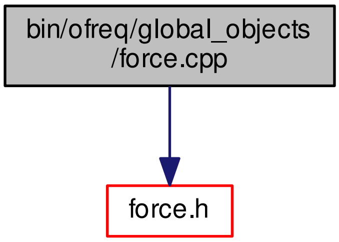
Namespaces

- namespace osea
- namespace osea::ofreq

16.65 bin/ofreq/global_objects/force.cpp File Reference

```
#include "force.h"
```

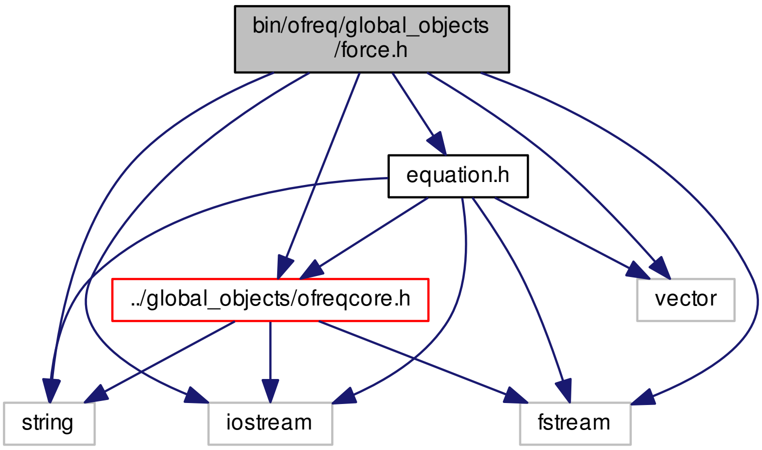
Include dependency graph for force.cpp:



16.66 bin/ofreq/global_objects/force.h File Reference

```
#include "equation.h"
#include "../global_objects/ofreqcore.h"
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
```

Include dependency graph for force.h:



Classes

- class osea::ofreq::Force

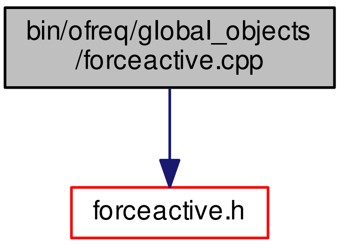
Namespaces

- namespace osea
- namespace osea::ofreq

16.67 bin/ofreq/global_objects/forceactive.cpp File Reference

```
#include "forceactive.h"
```

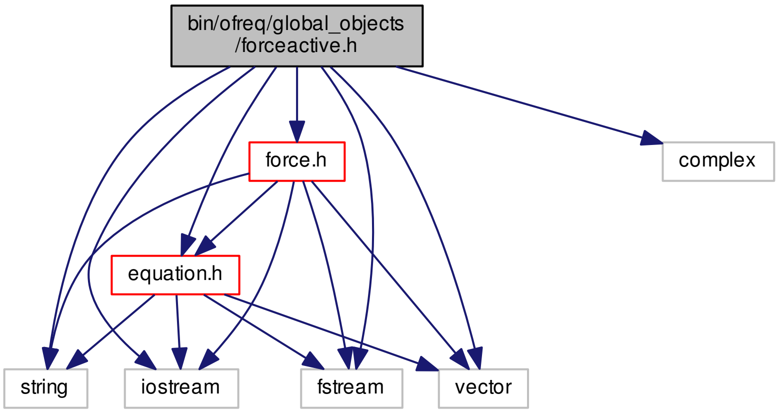
Include dependency graph for forceactive.cpp:



16.68 bin/ofreq/global_objects/forceactive.h File Reference

```
#include "force.h"
#include "equation.h"
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
#include <complex>
```

Include dependency graph for forceactive.h:



Classes

- class osea::ofreq::ForceActive

Namespaces

- namespace osea
- namespace osea::ofreq

Macros

- #define FORCEACTIVE_H

16.68.1 Macro Definition Documentation

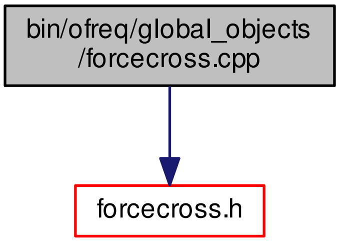
16.68.1.1 #define FORCEACTIVE_H

Definition at line 40 of file forceactive.h.

16.69 bin/ofreq/global_objects/forcecross.cpp File Reference

```
#include "forcecross.h"
```

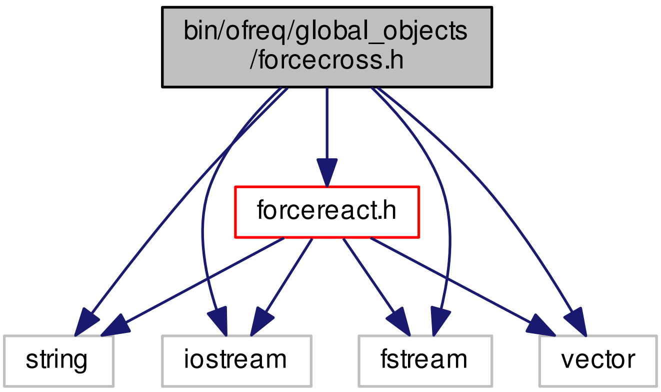
Include dependency graph for forcecross.cpp:



16.70 bin/ofreq/global_objects/forcecross.h File Reference

```
#include "forcereact.h"
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
```

Include dependency graph for forcecross.h:



Classes

- class osea::ofreq::ForceCross

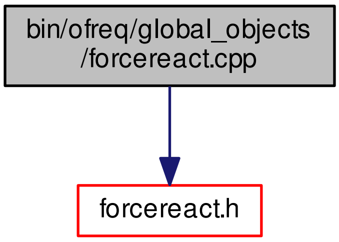
Namespaces

- namespace osea
- namespace osea::ofreq

16.71 bin/ofreq/global_objects/forcereact.cpp File Reference

```
#include "forcereact.h"
```

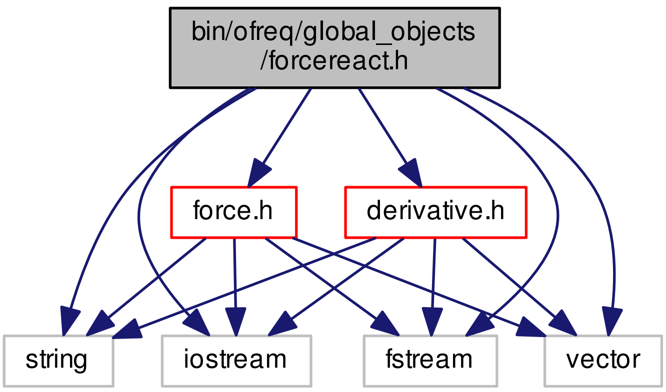
Include dependency graph for forcereact.cpp:



16.72 bin/ofreq/global_objects/forcereact.h File Reference

```
#include "force.h"
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
#include "derivative.h"
```

Include dependency graph for forcereact.h:



Classes

- class osea::ofreq::ForceReact

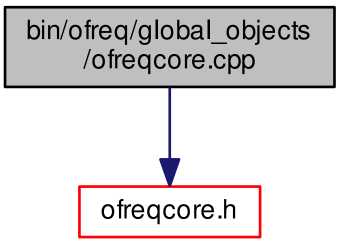
Namespaces

- namespace osea
- namespace osea::ofreq

16.73 bin/ofreq/global_objects/ofreqcore.cpp File Reference

```
#include "ofreqcore.h"
```

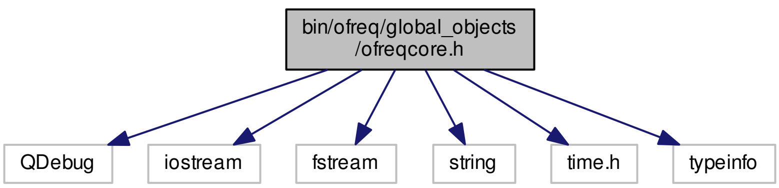
Include dependency graph for ofreqcore.cpp:



16.74 bin/ofreq/global_objects/ofreqcore.h File Reference

```
#include <QDebug>
#include <iostream>
#include <fstream>
#include <string>
#include <time.h>
#include <typeinfo>
```

Include dependency graph for ofreqcore.h:



Classes

- class osea::ofreq::oFreqCore

  The core oFreq class. All oFreq classes inherit from this class.

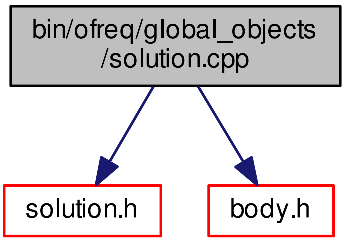
Namespaces

- namespace osea
- namespace osea::ofreq

16.75 bin/ofreq/global_objects/solution.cpp File Reference

```
#include "solution.h"
#include "body.h"
```

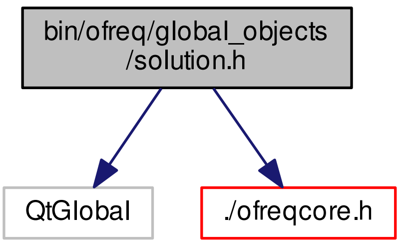
Include dependency graph for solution.cpp:



16.76 bin/ofreq/global_objects/solution.h File Reference

```
#include <QtGlobal>
#include "./ofreqcore.h"
```

Include dependency graph for solution.h:



Classes

- class osea::ofreq::Solution

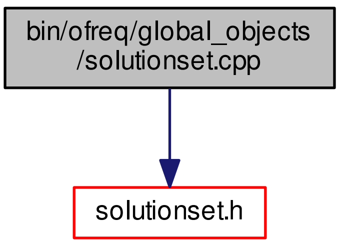
Namespaces

- namespace osea
- namespace osea::ofreq

16.77 bin/ofreq/global_objects/solutionset.cpp File Reference

```
#include "solutionset.h"
```

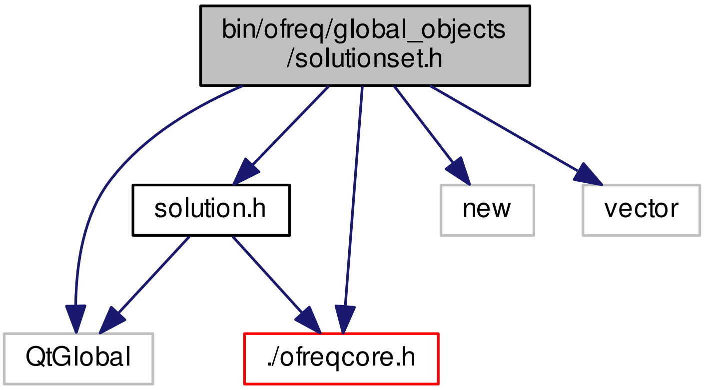
Include dependency graph for solutionset.cpp:



16.78 bin/ofreq/global_objects/solutionset.h File Reference

```
#include "solution.h"
#include <new>
#include <vector>
#include <QtGlobal>
#include "./ofreqcore.h"
```

Include dependency graph for solutionset.h:



Classes

- class osea::ofreq::SolutionSet

Namespaces

- namespace osea
- namespace osea::ofreq

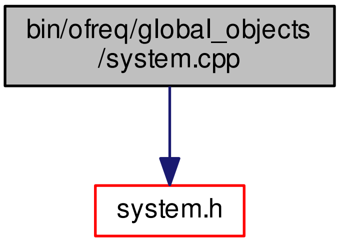
Typedefs

- typedef std::vector$<$ Solution $*$ $>$ osea::ofreq::ptSoln

16.79 bin/ofreq/global_objects/system.cpp File Reference

```
#include "system.h"
```

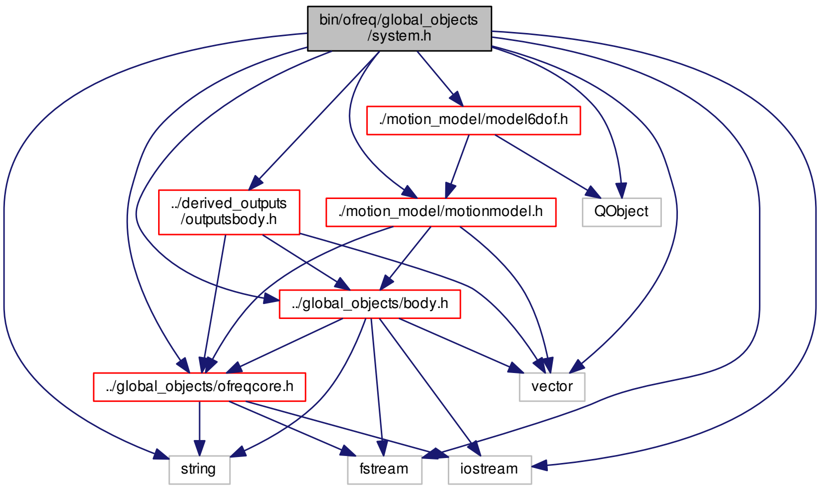
Include dependency graph for system.cpp:



16.80 bin/ofreq/global_objects/system.h File Reference

```
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
#include <QObject>
#include "../global_objects/body.h"
#include "../derived_outputs/outputsbody.h"
#include "./ofreqcore.h"
#include "./motion_model/motionmodel.h"
#include "./motion_model/model6dof.h"
```

Include dependency graph for system.h:



Classes

- class osea::ofreq::System

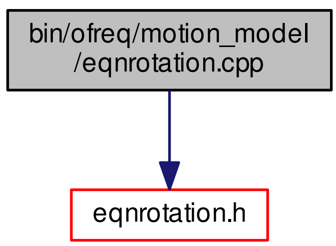
Namespaces

- namespace osea
- namespace osea::ofreq

16.81 bin/ofreq/motion_model/eqnrotation.cpp File Reference

```
#include "eqnrotation.h"
```

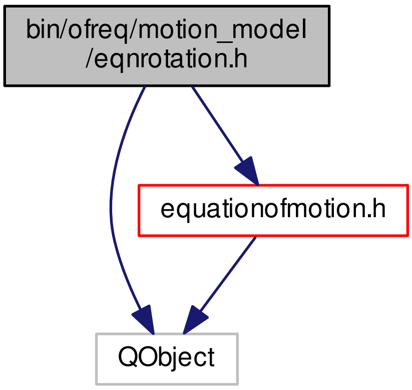
Include dependency graph for eqnrotation.cpp:



16.82 bin/ofreq/motion_model/eqnrotation.h File Reference

```
#include <QObject>
#include "equationofmotion.h"
```

Include dependency graph for eqnrotation.h:



Classes

- class osea::ofreq::EqnRotation

  The EqnRotation class.

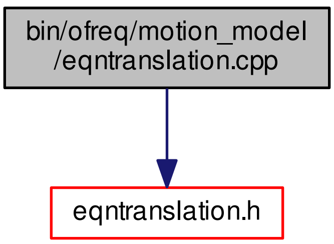
Namespaces

- namespace osea
- namespace osea::ofreq

16.83 bin/ofreq/motion_model/eqntranslation.cpp File Reference

```
#include "eqntranslation.h"
```

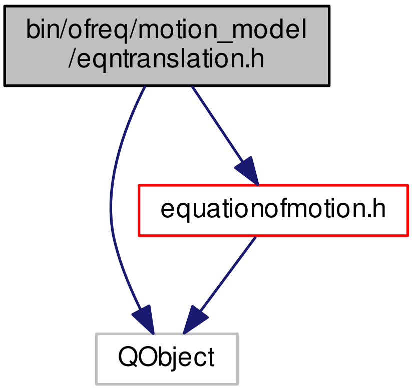
Include dependency graph for eqntranslation.cpp:



16.84 bin/ofreq/motion_model/eqntranslation.h File Reference

```
#include <QObject>
#include "equationofmotion.h"
```

Include dependency graph for eqntranslation.h:



Classes

- class osea::ofreq::EqnTranslation

    The EqnTranslation class.

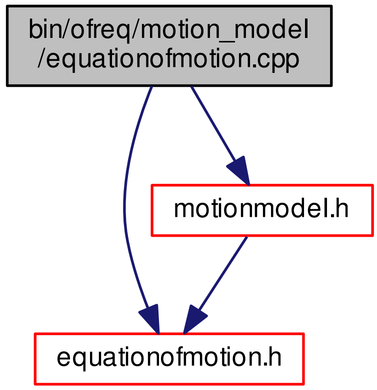
Namespaces

- namespace osea
- namespace osea::ofreq

16.85 bin/ofreq/motion_model/equationofmotion.cpp File Reference

```
#include "equationofmotion.h"
#include "motionmodel.h"
```

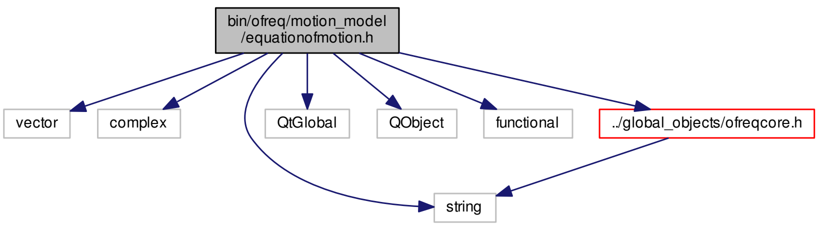
Include dependency graph for equationofmotion.cpp:



16.86 bin/ofreq/motion_model/equationofmotion.h File Reference

```
#include <vector>
#include <complex>
#include <string>
#include <QtGlobal>
#include <QObject>
#include <functional>
#include "../global_objects/ofreqcore.h"
```

Include dependency graph for equationofmotion.h:



Classes

- class osea::ofreq::EquationofMotion

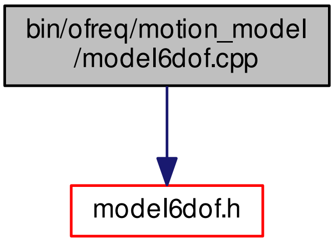
Namespaces

- namespace osea
- namespace osea::ofreq

16.87 bin/ofreq/motion_model/model6dof.cpp File Reference

```
#include "model6dof.h"
```

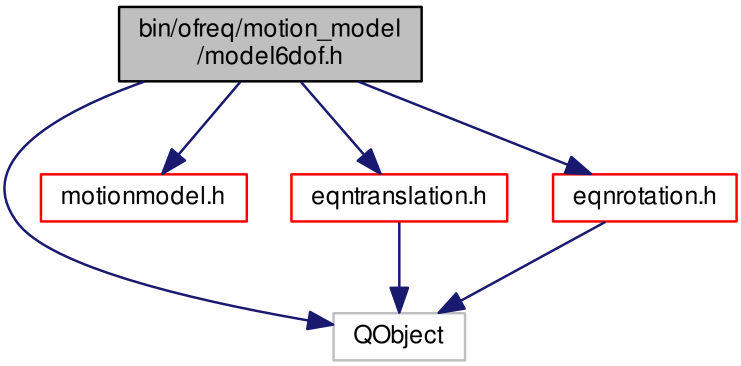
Include dependency graph for model6dof.cpp:



16.88 bin/ofreq/motion_model/model6dof.h File Reference

```
#include <QObject>
#include "motionmodel.h"
#include "eqntranslation.h"
#include "eqnrotation.h"
```

Include dependency graph for model6dof.h:



Classes

- class osea::ofreq::Model6DOF

    The motion model for standard six-degree of freedom rigid-body dynamics problems.

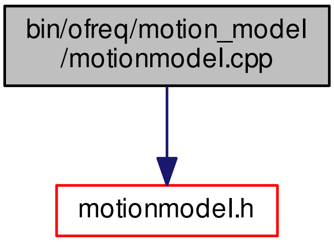
Namespaces

- namespace osea
- namespace osea::ofreq

16.89 bin/ofreq/motion_model/motionmodel.cpp File Reference

```
#include "motionmodel.h"
```

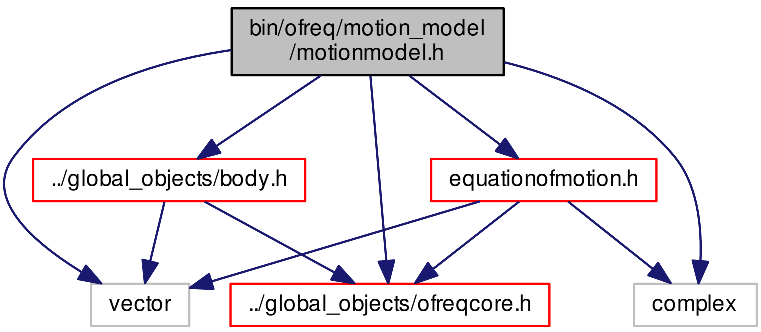
Include dependency graph for motionmodel.cpp:



16.90 bin/ofreq/motion_model/motionmodel.h File Reference

```
#include <vector>
#include <complex>
#include "../global_objects/body.h"
#include "equationofmotion.h"
#include "../global_objects/ofreqcore.h"
```

Include dependency graph for motionmodel.h:



Classes

- class osea::ofreq::MotionModel

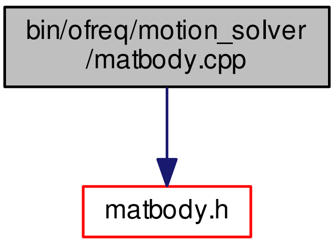
Namespaces

- namespace osea
- namespace osea::ofreq

16.91 bin/ofreq/motion_solver/matbody.cpp File Reference

```
#include "matbody.h"
```

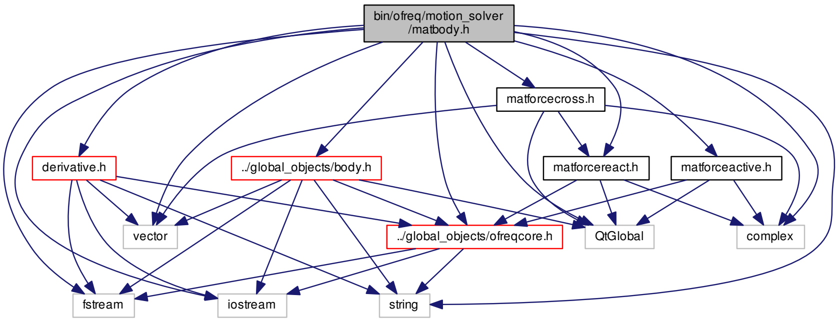
Include dependency graph for matbody.cpp:



16.92 bin/ofreq/motion_solver/matbody.h File Reference

```
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
#include <complex>
#include <QtGlobal>
#include "../global_objects/body.h"
#include "../global_objects/derivative.h"
#include "../global_objects/ofreqcore.h"
#include "matforcereact.h"
#include "matforceactive.h"
#include "matforcecross.h"
```

Include dependency graph for matbody.h:



Classes

- class osea::ofreq::matBody

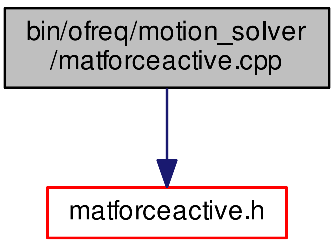
Namespaces

- namespace osea
- namespace osea::ofreq

16.93 bin/ofreq/motion_solver/matforceactive.cpp File Reference

```
#include "matforceactive.h"
```

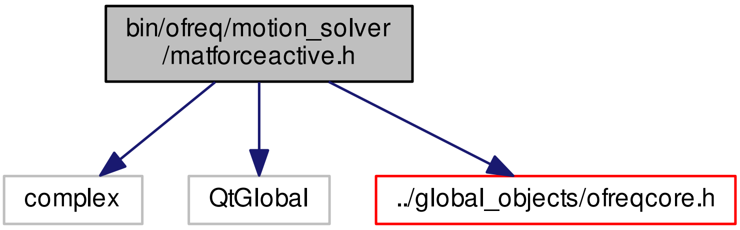
Include dependency graph for matforceactive.cpp:



16.94 bin/ofreq/motion_solver/matforceactive.h File Reference

```
#include <complex>
#include <QtGlobal>
#include "../global_objects/ofreqcore.h"
```

Include dependency graph for matforceactive.h:



Classes

- class osea::ofreq::matForceActive

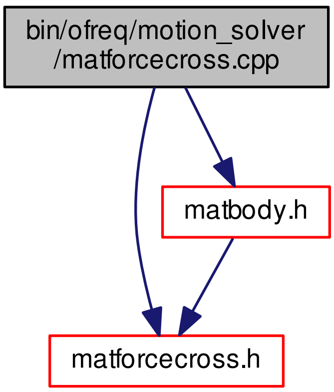
Namespaces

- namespace osea
- namespace osea::ofreq

16.95 bin/ofreq/motion_solver/matforcecross.cpp File Reference

```
#include "matforcecross.h"
#include "matbody.h"
```

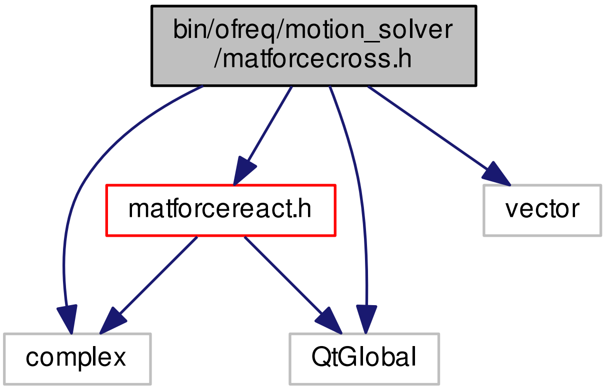
Include dependency graph for matforcecross.cpp:



16.96 bin/ofreq/motion_solver/matforcecross.h File Reference

```
#include "matforcereact.h"
#include <vector>
#include <complex>
#include <QtGlobal>
```

Include dependency graph for matforcecross.h:



Classes

- class osea::ofreq::matForceCross

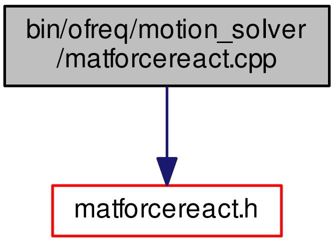
Namespaces

- namespace osea
- namespace osea::ofreq

16.97 bin/ofreq/motion_solver/matforcereact.cpp File Reference

```
#include "matforcereact.h"
```

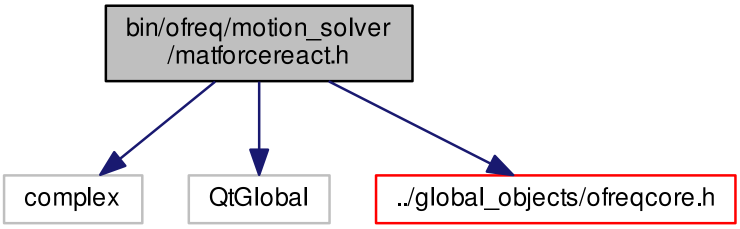
Include dependency graph for matforcereact.cpp:



16.98 bin/ofreq/motion_solver/matforcereact.h File Reference

```
#include <complex>
#include <QtGlobal>
#include "../global_objects/ofreqcore.h"
```

Include dependency graph for matforcereact.h:



Classes

- class osea::ofreq::matForceReact

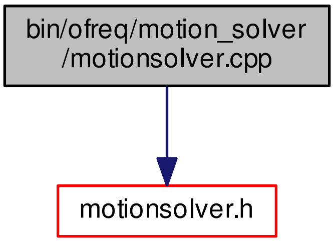
Namespaces

- namespace osea
- namespace osea::ofreq

16.99 bin/ofreq/motion_solver/motionsolver.cpp File Reference

```
#include "motionsolver.h"
```

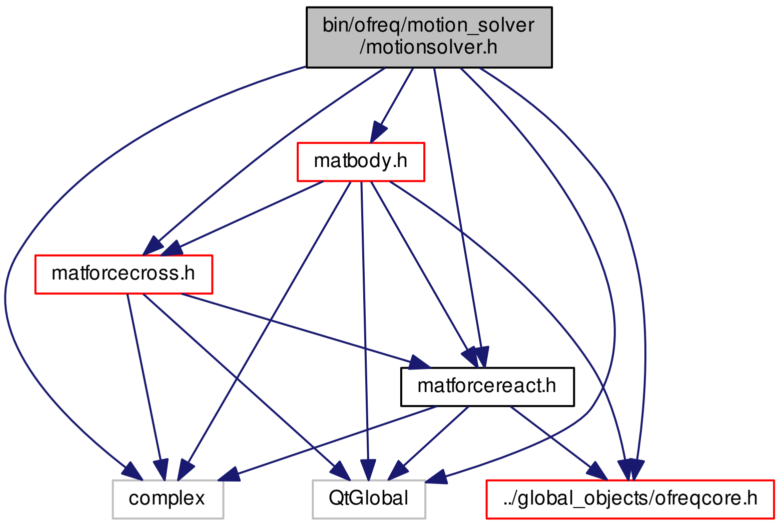
Include dependency graph for motionsolver.cpp:



16.100 bin/ofreq/motion_solver/motionsolver.h File Reference

```
#include <complex>
#include <QtGlobal>
#include "matbody.h"
#include "matforcereact.h"
#include "matforcecross.h"
#include "../global_objects/ofreqcore.h"
```

Include dependency graph for motionsolver.h:



Classes

- class osea::ofreq::MotionSolver

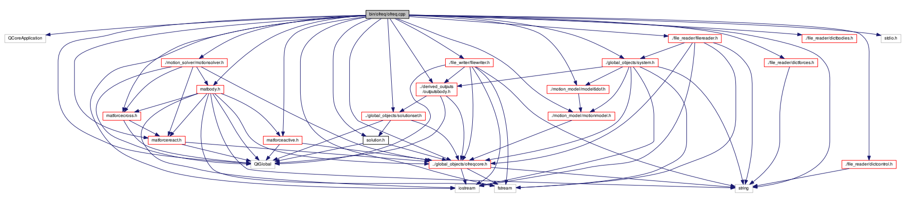
Namespaces

- namespace osea
- namespace osea::ofreq

16.101 bin/ofreq/ofreq.cpp File Reference

```
#include <QCoreApplication>
#include "./motion_solver/motionsolver.h"
#include "./motion_model/motionmodel.h"
#include "./motion_solver/matbody.h"
#include "./motion_solver/matforceactive.h"
#include "./motion_solver/matforcecross.h"
#include "./motion_solver/matforcereact.h"
#include "./file_writer/filewriter.h"
#include "./derived_outputs/outputsbody.h"
#include "./global_objects/solutionset.h"
#include "./global_objects/solution.h"
#include "./global_objects/system.h"
#include "./file_reader/dictcontrol.h"
#include "./file_reader/dictforces.h"
#include "./file_reader/dictbodies.h"
#include "./file_reader/filereader.h"
#include "./global_objects/ofreqcore.h"
#include <string>
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <QtGlobal>
#include "./motion_model/model6dof.h"
```

Include dependency graph for ofreq.cpp:



Functions

- void buildMatBody (int bod, bool useCoeff=true)

  Builds a matrix body object for the body specified by the integer. Uses the motion model identified by the Body object.
- void calcOutput (OutputsBody &OutputIn, FileWriter &WriterIn)

  Calculates derived outputs using the OutputsBody object and then writes those outputs to files.
- void ReadFiles (string runPath)

  Reads in all the input files. Creates the necessary objects for file reading. And connects those objects using Qt slots and signals. Finally proceeds through each file and reads it. All parsing is accomplished by the FileReader object. File interpretation is processed through the Dictionary objects.

- std::string getPath (std::string typePath="exec")

  Finds the path of the critical files for the program.
- int main (int argc, char ∗argv[ ])

  The main function that runs ofreq program.
- std::string getPath (string typePath)

Variables

- vector< matBody > listMatBody
- vector< SolutionSet > listSolutions
- System sysofreq
- oFreqCore Logs
- const std::string EXECNAME = "ofreq"
- const std::string EXECFOLDER = "ofreq_debug"
- const std::string VARFOLDER = "var"
- const std::string LIBFOLDER = "lib"
- const std::string ETCFOLDER = "etc"
- const std::string BINFOLDER = "bin"
- string oFreq_Directory = ""

16.101.1 Function Documentation

16.101.1.1 void buildMatBody (int *bod,* bool *useCoeff* = `true`)

Builds a matrix body object for the body specified by the integer. Uses the motion model identified by the Body object.

Parameters

bod	Which body to use for building the matix body.

Returns

Returns a matBody object, fully provisioned with all necessary data.

Definition at line 352 of file ofreq.cpp.

16.101.1.2 void calcOutput (OutputsBody & *OutputIn,* FileWriter & *WriterIn*)

Calculates derived outputs using the OutputsBody object and then writes those outputs to files.

Parameters

OutputIn	The OutputsBody object that will calculate the derived outputs. All properties for the Outputs-Body object must be set by the time the function is called. Variable passed by reference.
WriterIn	The FileWriter object that will receive the outputs from the OutputsBody object and write those outputs to a file.

Definition at line 520 of file ofreq.cpp.

16.101.1.3 std::string getPath (std::string *typePath* = `"exec"`)

Finds the path of the critical files for the program.

Finds the path of one of four possible folders that are critical to the oSea program. Includes platform dependant code so that this function should work both under Windows or Linux.

Parameters

typePath	String that specifies which path to get. Options are: "exec": Path to the directory of the executable file. NOT the directory the program was called from. "lib": Path to the lib directory that is common to all oSea programs. "var": Path to the var directory that is common to all oSea programs. "etc": Path to the etc directory that is common to all oSea programs. "bin": Path to the binaries directory. Binaries for individual programs are included in sub folders.

Returns

Returns std::string that is the full absolute path to the specified . Returned variable passed by value.

16.101.1.4 std::string getPath (string *typePath*)

Definition at line 626 of file ofreq.cpp.

16.101.1.5 int main (int *argc,* char ∗ *argv[]*)

The main function that runs ofreq program.

The main function that runs ofreq program. It proceeds through in several stages. This briefly outlines them. 0. Initialize a few variables.

1. Read in input files.
2. Interpret / parse input files and use them to build the program objects.
3. Use the motion model to convert input objects into matrix force objects. The particular motion model used changes with each Body object.
4. Setup the motion solver. Feed all the data in.
5. Set the operating frequency and use the motion solver to solve equations of motion.
6. Store results in Solution object.
7. Repeat steps 4 through 6 for each wave direction and wave frequency.
8. Use the results to calculate derived outputs.
9. Write the calculated outputs to output files.

Definition at line 189 of file ofreq.cpp.

16.101.1.6 void ReadFiles (string *runPath*)

Reads in all the input files. Creates the necessary objects for file reading. And connects those objects using Qt slots and signals. Finally proceeds through each file and reads it. All parsing is accomplished by the FileReader object. File interpretation is processed through the Dictionary objects.

Parameters

runPath	String. The path to the root directory of the input files.

See Also

Dictionary

Definition at line 597 of file ofreq.cpp.

16.101.2 Variable Documentation

16.101.2.1 const std::string BINFOLDER = "bin"

Definition at line 110 of file ofreq.cpp.

16.101.2.2 const std::string ETCFOLDER = "etc"

Definition at line 107 of file ofreq.cpp.

16.101.2.3 const std::string EXECFOLDER = "ofreq_debug"

Definition at line 98 of file ofreq.cpp.

16.101.2.4 const std::string EXECNAME = "ofreq"

Definition at line 95 of file ofreq.cpp.

16.101.2.5 const std::string LIBFOLDER = "lib"

Definition at line 104 of file ofreq.cpp.

16.101.2.6 vector<matBody> listMatBody

Definition at line 83 of file ofreq.cpp.

16.101.2.7 vector<SolutionSet> listSolutions

Definition at line 86 of file ofreq.cpp.

16.101.2.8 oFreqCore Logs

Definition at line 92 of file ofreq.cpp.

16.101.2.9 string oFreq_Directory = ""

Definition at line 169 of file ofreq.cpp.

16.101.2.10 System sysofreq

Definition at line 89 of file ofreq.cpp.

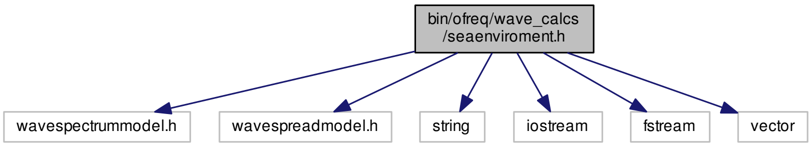
16.101.2.11 const std::string VARFOLDER = "var"

Definition at line 101 of file ofreq.cpp.

16.102 bin/ofreq/wave_calcs/seaenviroment.h File Reference

```
#include "wavespectrummodel.h"
#include "wavespreadmodel.h"
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
```

Include dependency graph for seaenviroment.h:



Classes

- class SeaEnviroment

Chapter 17

Example Documentation

17.1 ExampleTextHeader.txt

Index