MOUSE Documentation

Release 0.3

M. Yetisir

Dec 30, 2016

CONTENTS:

CHAPTER

ONE

MODULES PACKAGE

1.1 Subpackages

1.1.1 Modules.HODS package

Submodules

Modules.HODS.HODS module

class `Modules.HODS.HODS.`**`DataSet`**(*fileName*)

Bases: `object`

`blocksWithContacts`(*blocks*, *contacts*)

`blocksWithCorners`(*blocks*, *corners*)

`contactsBetweenBlocks`(*blocks1*, *blocks2*)

`contactsOnBlocks`(*blocks*)

`cornerX`(*corners*, *time*)

`cornerY`(*corners*, *time*)

`cornersOnBlocks`(*blocks*)

`cornersOnContacts`(*contacts*)

`limits`()

`parseDataFile`(*fileName*)

`zoneS11`(*zones*, *time*)

`zoneS12`(*zones*, *time*)

`zoneS22`(*zones*, *time*)

`zoneS33`(*zones*, *time*)

`zonesInBlocks`(*blocks*)

class `Modules.HODS.HODS.`**`Homogenize`**(*centre*, *radius*, *fileName*)

Bases: *`Modules.HODS.HODS.DataSet`*

`blocksInsideBoundary`()

`blocksOnBoundary`()

`blocksOutsideBoundary`()

calculateHomogenizationParameters()

contactsInsideBoundary()

contactsOutsideBoundary()

cornersInsideBoundary()

cornersOutsideBoundary()

duplicateCorners(*corners*, *blocks*)

orderBlocks(*blocks*, *relaventContacts*)

orderCorners(*orderedBlocks*, *corners*)

singleElementCorners()

strain()

stress()

time()

class `Modules.HODS.HODS.`**common**

Bases: `object`

angle(*x1*, *y1*, *x2*, *y2*)

area(*p*)

listIntersection(*a*, *b*)

segments(*p*)

triangleArea(*gp*)

Module contents

1.2 Submodules

1.3 Modules.Module_ABAQUS module

class `Modules.Module_ABAQUS.`**Module_ABAQUS**(*baseName*)

Bases: *Modules.Base.ContinuumModuleBaseClass*

createArgumentParser()

formatOutput()

parseArguments()

parseInput()

run()

setParameters(*revCentreX=None*, *revCentreY=None*, *revRadius=None*)

`Modules.Module_ABAQUS.`**importModelData**(*modelName*)

`Modules.Module_ABAQUS.`**parserHandler**(*args*)

`Modules.Module_ABAQUS.`**populateArgumentParser**(*parser*)

1.4 Modules.Module_HODS module

class `Modules.Module_HODS.`**`Module_HODS`**(*baseName*)

Bases: *`Modules.Base.HomogenizationModuleBaseClass`*

`createArgumentParser`()

`formatOutput`()

`parseArguments`(*args*)

`parseInput`()

`run`()

`setParameters`(*args*)

`Modules.Module_HODS.`**`importModelData`**(*modelName*)

`Modules.Module_HODS.`**`parserHandler`**(*args*)

`Modules.Module_HODS.`**`populateArgumentParser`**(*parser*)

1.5 Modules.Module_OSTRICH module

class `Modules.Module_OSTRICH.`**`Module_OSTRICH`**(*baseName*)

Bases: *`Modules.Base.ParameterEstimationModuleBaseClass`*

`createArgumentParser`()

`formatOutput`()

`getBoundaryDisplacements`()

`getBoundaryStresses`()

`getModelConstants`()

`getModelParameters`()

`getOstrichParameters`(*frontBias=1*)

`parseArguments`(*args*)

`parseInput`()

`run`()

`setParameters`(*args*)

`Modules.Module_OSTRICH.`**`fillTemplate`**(*template*, *parameters*, *file*)

`Modules.Module_OSTRICH.`**`getVelocityString`**(*velTable*)

`Modules.Module_OSTRICH.`**`importMaterialData`**(*materialName*)

`Modules.Module_OSTRICH.`**`importModelData`**(*modelName*)

`Modules.Module_OSTRICH.`**`parserHandler`**(*args*)

`Modules.Module_OSTRICH.`**`populateArgumentParser`**(*parser*)

1.6 Modules.Module_UDEC module

class `Modules.Module_UDEC.`**`Module_UDEC`**(*baseName*)

Bases: *`Modules.Base.DemModuleBaseClass`*

`createArgumentParser`()

`createInputFiles`()

`formatOutput`()

`getUDECParameters`()

`getVelocityString`(*velTable*)

`inputFileName`(*data*)

`loadData`()

`outputFileName`()

`parseArguments`()

`parseInput`()

`run`()

`setParameters`()

`Modules.Module_UDEC.`**`compileFiles`**(*simulations*, *files*, *rawPath*, *compiledPath*)

`Modules.Module_UDEC.`**`fileList`**(*path*)

`Modules.Module_UDEC.`**`importModelData`**(*modelName*)

`Modules.Module_UDEC.`**`parseDataFile`**(*fileName*)

`Modules.Module_UDEC.`**`parserHandler`**(*args*)

`Modules.Module_UDEC.`**`populateArgumentParser`**(*parser*)

`Modules.Module_UDEC.`**`simulationFiles`**(*files*, *rawPath*, *compiledPath*)

1.7 Modules.Base module

class `Modules.Base.`**`ContinuumModuleBaseClass`**(*program*, *parameters*, *baseName*)

Bases: *`Modules.Base.ModuleBaseClass`*

Creates a base class for the continuum model modules containing common methods and attributes

A base continuum model module class is implemented here, inheriting from the module base class to provide a framework containing required methods and attributes for the continuum model modules to inherit. The module class contains methods pertaining to I/O routines associated with the module so that each module that is written behaves in a consistent manner and to avoid reimplementation of certain methods.

`type`

str – Type of module

`inputFileName`()

Returns full path of input binary data

Returns full path of input binary data

Return type str

`outputFileName``()`
Returns full path of output binary data

Returns full path of output binary data

Return type str

class `Modules.Base.`**`DemModuleBaseClass`**`(`*program*, *parameters*, *baseName*`)`
Bases: *`Modules.Base.ModuleBaseClass`*

Creates a base class for the DEM modules containing common methods and attributes

A base dem module class is implemented here, inheriting from the module base class to provide a framework containing required methods and attributes for the DEM modules to inherit. The module class contains methods pertaining to I/O routines associated with the module so that each module that is written behaves in a consistent manner and to avoid reimplementation of certain methods.

`type`
str – Type of module

`inputFileName``()`
Returns full path of input binary data

Returns full path of input binary data

Return type str

`outputFileName``()`
Returns full path of output binary data

Returns full path of output binary data

Return type str

class `Modules.Base.`**`HomogenizationModuleBaseClass`**`(`*program*, *parameters*, *baseName*`)`
Bases: *`Modules.Base.ModuleBaseClass`*

Creates a base class for the homogenization modules containing common methods and attributes

A base homogenization module class is implemented here, inheriting from the module base class to provide a framework containing required methods and attributes for the homogenization modules to inherit. The module class contains methods pertaining to I/O routines associated with the module so that each module that is written behaves in a consistent manner and to avoid reimplementation of certain methods.

`type`
str – Type of module

`inputFileName``()`
Gets full path of input binary data

Returns full path of input binary data

Return type str

`outputFileName``()`
Returns full path of output binary data

Returns full path of output binary data

Return type str

class `Modules.Base.`**`ModuleBaseClass`**`(`*program*, *baseName*, *parameters={}*, *suppressText=False*, *suppressErrors=True*`)`
Bases: `object`

Creates a base class containing common module methods and attributes

A base module class is implemented here to provide a framework containing required methods and attributes for the MOUSE modules to inherit. The module class contains methods pertaining to I/O routines associated with the module so that each module that is written behaves in a consistent manner and to avoid reimplementation of certain methods.

`program`
str – String containing name of module software executable file.

`parameters`
dict – Dictionary of command line parameters as keys and corresponding arguments as entries

`suppressText`
bool – Suppreses text output from modules if True

`suppressErrors`
bool – Suppress error output from modules if True

`baseName`
str – Name of model input file

`binaryDirectory`
str – Directory in which MOUSE binary data is located

`textDirectory`
str – Directory in which MOUSE text data is located

`inputDirectory`
str – Directory in which MOUSE input data is located

`outputDirectory`
str – Directory in which MOUSE output data is located

`clearScreen``()`
Clears all text from the console.

Returns: None: Clears all text from the console

`commandLineArguments``()`
converts the parameters dictionary to a string which can be passed to the command line when running the specified program.

Returns string to be passed to command line

Return type str

`loadData``()`
Loads module data from binary using the pickle serialization module

Parameters **`data`** (*`any`*) – Module data to be serialized and stored in file

Returns serialized data in binary file in specified binaryDirectory

Return type None

`printDone``()`
Prints ‘Done’ to console.

Note: It is recommended that this method be used in conjunction with printStatus()

Parameters **`status`** (*`str`*) – status to be printed to console

Returns ‘Done’ printed to the console

Return type None

printErrors(*error*)

Prints error to console if not suppressed

Note: All errors caught should be routed through this function. Using this function allows for easy suppression and piping of output.

Parameters **error** (`str`) – error to be printed to console

Returns error printed to the console

Return type None

printSection(*section*)

Prints a section name to console.

Sections are displayed alligned to the left side of the console.

Parameters **section** (`str`) – section name to be printed to console

Returns section name printed to the console

Return type None

printStatus(*status*)

Prints a status to console.

Statuses are displayed proceeding a tab and are follwed by ellipses with no new line character at the end of the print line.

Note: The no new line character at the end of the print line allows the printDone() method to print 'Done' at the end of the ellipses after some arbitrary code execution. It is recommended that these two methods always be used together

Parameters **status** (`str`) – status to be printed to console

Returns status printed to the console

Return type None

printText(*text*, *end='\n'*)

Prints text to console if not suppressed

Note: All text printed to the console should be routed through this function rather than using the built-in print() function. Using this function allows for easy suppression and piping of output.

Todo

If text suppression is on, route output to file.

Parameters

- **text** (`str`) – text to be printed to console
- **end** (`str, optional`) – character to be appended to end of print line

Returns text printed to the console

Return type None

printTitle(*title*)
Prints a title to console.

Titles are displayed with horizontal lines printed above and below the text and are alligned with the left side of the console.

Parameters **title** (*str*) – title to be printed to console

Returns title printed to the console

Return type None

run()
runs specified program with specified parameters

Returns runs specified program with specified parameters

Return type None

saveData(*data*)
Saves module data as binary using the pickle serialization module

Parameters **data** (*any*) – Module data to be serialized and stored in file

Returns serialized data in binary file in specified binaryDirectory

Return type None

updateParameters(*parameters*)
Updates the parameter attribute so that the modul can be run with a different parameter set without being re-instantiated

Parameters **parameters** (*dict*) – dictionary of new parameters

Returns updates the parameter attribute

Return type None

class Modules.Base.**ParameterEstimationModuleBaseClass**(*program*, *parameters*, *baseName*)
Bases: *Modules.Base.ModuleBaseClass*

Creates a base class for the parameter estimation modules containing common methods and attributes

A base parameter estimation module class is implemented here, inheriting from the module base class to provide a framework containing required methods and attributes for the parameter estimation modules to inherit. The module class contains methods pertaining to I/O routines associated with the module so that each module that is written behaves in a consistent manner and to avoid reimplementation of certain methods.

type
str – Type of module

inputFileName()
Returns full path of input binary data

Returns full path of input binary data

Return type str

outputFileName()
Returns full path of output binary data

Returns full path of output binary data

Return type str

1.8 Module contents

CHAPTER TWO

MOUSE MODULE

class `MOUSE.`**`SplashScreen`**(*boxWidth=55*, *textWidth=70*, *padWidth=15*)

Bases: `object`

Creates the splash screen and interface for MOUSE

This class allows for the generation of an introduction screen for MOUSE. Here, a collection of printing methods are created in order to provide an environment for creating a consistent splash screen and interface.

`boxWidth`

int – Character width of text box for splash screen

`textWidth`

int – Character width of text area for splash screen

`padWidth`

int – Character width of text area for padding on splash screen

`printBoxLine`()

Prints a horizontal line for the box in the centre of the console

Returns Prints a centred horizontal dashed line of length self.boxWidth on the console

Return type None

`printCentre`(*text*)

Prints text in the centre of the console

Parameters **`text`** (*`str`*) – text to be printed in the centre of the console

Returns Prints str to the centre of the console

Return type None

`printFullLine`()

Prints a horizontal line across the text width of the console

Returns Prints a horizontal dashed line of length self.textWidth on the console

Return type None

`printInBox`(*text*)

Prints text in the centre of the box

Parameters **`text`** (*`str`*) – text to be printed in the centre of the box

Returns Prints a horizontal dashed line of length self.textWidth on the console

Return type None

`printModule`(*module*, *status*)

Prints module and installation status in the splash box

Parameters

- **`module`** (*`str`*) – name of the module
- **`status`** (*`str`*) – module status [installed, available, unavailable]

Returns Prints the module name with the status in the splash box

Return type None

`printSplash``()`

Clears the console and prints the splash screen to the console

Returns Splash screen printed on console

Return type None

Todo

Import Modules and stuses from module files rather than hard coding them into this method

---

`MOUSE.`**`createMainParser`**`()`

Creates an argparse parser object for MOUSE

Note: This does not populate the parser with arguments

Returns the main argument parser for MOUSE

Return type argparse.ArgumentParser

`MOUSE.`**`importModuleParsers`**`(`*parser*`)`

Imports argparse subparsers for each MOUSE Module

Todo

Scan subparsers from module files and import in order to remove hard-coded dependance

---

Note: Currenlty subparser imports are hard-coded in

Parameters **`parser`** (*`argparse.ArgumentParser`*) – the main argument parser for MOUSE

Returns the main argument parser for MOUSE now populated with all required subparsers form modules.

Return type argparse.ArgumentParser

CHAPTER THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX