Mili Python Interface

Table of Contents

Developer’s Guide

Description

First, you need to make sure the file mili_reader_lib.py is in your path. The file is located at

ENTER THIS LATER

This program can be imported and initialized using the following:

```
import mili_reader_lib

mili = mili_reader_lib.Mili()

mili.read(file_name)
```

```
import mili_reader_lib

mili = mili_reader_lib.Mili(file_name)
```

If you are using a parallel file, you would have the `file_name` field be something like `‘parallel/d3samp6.plt’` where all the files for this run are in that directory.

From this point, the mili object contains all the information from the mili file and can be queried.

```
answer = mili.nodes_of_elem(1, “brick)

answer2 = mili.query('nodpos[ux]', None, 4, 'node', 3)
```

Mili Class

Description

The Mili Class has variables that store the pertinent information for a mili file. These variables are filled during the various functions that read in the Mili file and will then be used when querying. The following are variables that a user might want to access if writing a script to work alongside the reader:

state_maps

Type: Array

Description: an array of StateMap objects

directories

Type: Array

Description: an array of Directory objects

params

Type: Dictionary

Description: maps parameter name to a list containing the value, which could be a single integer, a string, or an array

state_variables

Type: Dictionary

Description: maps from:

name: [StateVariable, [subrecords containing this name]]

mesh_object_class_datas

Type: Dictionary

Description: maps from shortname: MeshObjectClassData

labels

Type: Dictionary

Description: maps from label: element id

(superclass, class) : {label: element id}

int_points

Type: Dictionary

Description: maps from:

stress/strain : {es_x : [(integration points), total integration points]}

and from:

es_x : [(integration points), total integration points]

nodes

Type: Array

Description: an array of starting node position coordinates (each entry in the list is a nodes position) with node number i-1 in the ith slot in the array, since the node array in the definition is 1-indexed, and python is 0-indexed

materials

Type: Dictionary
Description: maps from:
material number : {class: [id_s]}
Each class can have its own element with id 2, so this organization is necessary

matname

Type: Dictionary
Description: maps from material name: material number

connectivity

Type: Dictionary
Description: maps from:
class_name : {id : [nodes]}
The nodes make up element id in the class

dim

Type: Integer
Description: the dimension of the simulation

parent_conns

Type: List
Description: the list of connections held by the parent to parallel processes

mili_num

Type: Integer
Description: the identifying number for the Mili process

parallel_mode

Type: Boolean
Description: If this is running in parallel mode or not

error_file

Type: String
Description: the error file name

labeltomili

Type: Dictionary
Description: maps from:
class_name: {label: [mili_nums]}

Reader Functions

__readStateMaps

input:

self: the mili object
f: the file object containing the state maps

output:

offset: the offset in the file after reading the state maps

description:

Reads the state maps one at a time, creating a new StateMap object for each and adding to the mili's state_map array

__readStateVariablesAndParams

input:

self: the mili object
f: the file object containing the state maps
offset: the offset to begin reading at

output:

None

description:

Iterates over the directories matching on directories with a DirectoryType of state variables or parameters. Given a parameter match, add the correct data to self.params. Given a state variable match, create a StateVariable and add the state variable to self.state_variables. When added, an empty array is included, which will later be filled with subrecords containing the state variable.

__readDirectories

input:

self: the mili object
f: the file object containing the state maps
offset: the offset to begin reading the directories

output:

offset: the offset in the file after reading the directories

description:

Reads the directories one at a time, creating a new `Directory` for each and adding to the mili's `self.directories` array.

__readMesh

input:

`self`: the mili object
`f`: the file object containing the state maps

output:

None

description:

Iterates through the directories matching on directories relating to the mesh, such as node and label definitions. Upon a match, updates the appropr0iate object in the mili object.

__readSubrecords

input:

`self`: the mili object
`f`: the file object containing the state maps

output:

None

description:

Iterates through the directories matching on directories with state record data. Given a match, creates a `Subrecord` and adds the subrecord to `self.srec_container`, which contains the subrecords.

read

input:

`self`: the mili object
`file_name`: the name of the problem to be run (e.g. bar1)

output:

None

description:

Calls the other reader functions to build up a Mili object. Capable of handling multiple state map files.

Query Functions

See users guide

Other Functions

__init__

input:

`self`: the mili object
`read_file`: the file name to read

output:

`None`

description:

If `read_file` is passed in, the initiliazer will also call the `read` function.

__del__

input:

`self`: the mili object

output:

`None`

description:

Shuts down any open connections.

modify_state_variable

See users guide

__split_reads

input:

`self`: the mili object
`file_name`: the file name to split reads on
`parallel_read`: whether or not this should be run in parallel

output:

`None`

description:

If `parallel_read`, starts of processes for each parallel Mili file and reads in the data. If not, serially reads each Mili file.

__child_read

input:

`self`: the mili object
`mili`: the child mili object
`conn`: the connection object for the child process to use
`file_name`: the `file_name` for the child to read
`i`: the `mili_num` for the child

output:

None

description:

The process each parallel file is read in. After reading, the child send the results back to the parent and waits for more commands from the parent.

__add_res

input:

`self`: the mili object
`to_add`: the value to add to the result
`res`: the result to combine to

output:

`res`

description:

Uses the fixed hierarchy of querying responses to add to results together

getParams

See users guide

getStateMaps

See users guide

getDirectories

See users guide

getStateVariables

See users guide

getLabels

See users guide

getMaterials

See users guide

__set_string

input:

`self`: the mili object
`subrecord`: the subrecord object

output:

`ret_final`: a string representation of the state variables on the subrecord

description:

Iterates through the state variables on the subrecord, appending to a return string along the way, given the type of the state variable. At the end of the method, `ret_final` contains a single string for every variable in the subrecord that can be used in a call using `struct`.

__parse_name

input:

`self`: the mili object
`name`: the name of the Mili file (the .pltA file)

output:

returns [`vector`, `component`]
`vector`: the name of the vector, if it is a vector
`component`: the name of the component, if there is a vector, or the name of the original state variable if there is no vector.

description:

Checks to see if the name contains an open bracket, indicating this is a vector with a component given. If so, breaks the name down into these two parts. Otherwise, returns the original name in the component slot.

__create_answer

input:

`self`: the mili object
`res`: the result
`names`: the names of the state variables
`materials`: the materials in the result
`labels`: the labels in the result
`class_name`: the class_name of the result
`state_numbers`: the state_numbers in the result
`modify`: whether or not there were modifications
`raw_data`: whether or not to simply output raw data, not `Answer`

output:

if `raw_data:`
    `res:` contains the results in the following structure:
        res[state_number][name][label]
`else:`
    answer: an `Answer` containing the information from `res`

description:

For every state, creates a `StateAnswer`, which contains all the information in `res[state_number]`. Then loops over the names and labels, gathering the information out of `res[state_number][name][label]` and adding the created `Item` from each of these the state answer.

__is_vec_array

input:

`self`: the mili object
`name`: the names of the state variable
`class_name`: the class_name of the state variable

output:

element set name or None

description:

Checks to see if name is an element set and returns the correct element set name given the class_name.

__variable_at_state

input:

self: the mili object
subrecord: the subrecord to search
labels: the labels of the state variable
name: the name of the state variable
vars: the array of state variable values on the subrecord
sup_class: the superclass of the state variable
clas: the class name of the state variable
sub: the number of the subrecord
res: the result
modify: whether or not this is part of a modification
int_points: the integration points

output:

if modify is True, returns [res, indices] where res is a dictionary structure containing the result and indices are the indices in the subrecord's array of variables to modify.
if modify is False, returns the values obtained by searching the subrecord for the specified state variables, labels, etc.

description:

Searches the subrecord for the given input.

__getLabelsFromClassElems

input:

self: the mili object
class_name: the class_name
elems: the elements to find labels for

output:

list of labels pertaining to the elems

description:

Turns labels into element numbers for a class.

__error

input:

self: the mili object
msg: the string to write to the error file or to the screen

output:

None

description:

Writes the msg to the output file or to the screen.

__addDicts

input:

`self`: the mili object
`a`: the dictionary to add
`b`: the dictionary to add to

output:

`b`

description:

Adds to dictionaries of sets together and returns the combined results

__get_children_info

input:

`self`: the mili object
`accumulator`: the string to write to the error file or to the screen
`function`: the function run on the `accumulator` and `query_response` to add them together
`message`: the string that describes what is being accumulated
`data_send`: the data to be sent to the parallel processes
`serial_function`: the serial function run when not in parallel mode for each Mili file

output:

`accumulator`: the accumulated answers from all the children processes.

description:

The parent process calls this when accumulating data from the children processes. It gets the information using either the `message` and `data_send` for parallel processes or the `serial_function` for serial processes.

__elements_of_material

input:

`self`: the mili object
`material`: the name or number of the material

output:

mo_ids: dict from class_name to list of element ids

description:

Given a material number or name, finds the elements of this material.

Other Objects

StateMap

Description

Contains information at given time instance for *every* subrecord. The state records themselves are stored in a separate binary file with a .plt00 ending.

__init__

input:

`self`: the mili object

`file_number`: the file number of this `StateMap`

`file_offset`: the offset for the state of the `StateMap`

`time`: the time in the simulation for this `StateMap`

`state_map_id`: the number of the `StateMap`

output:

None

description:

Initializes the variables of the `StateMap`

Directory

Description

An organization tool for Mili files. These files dictate the layout of the .pltA file.

__init__

input:

`self`: the `Directory` object

`type_idx`: the type of the `Directory`

`modifier_idx1`: type specific information

`modifier_idx2`: type specific information

`string_qty_idx`: number of strings assoicaited with this `Directory`

`offset_idx`: offset to being reading `Directory`

`length_idx`: the length of the `Directory`

output:

None

description:

Initializes the variables of the `Directory`

StateVariable

Description

Objects that are updated at different time instances.

list_size: the number of variables if this is a vector
order: the number of dims
dims: the rank of the state variables in svars
svars: the list of state variables for vectors and vector arrays

__init__

input:

self: the StateVariable object
name: the name of the StateVariable
title: the title of the StateVariable
agg_type: the aggregate type of the StateVariable
data_type: the data type of the StateVariable

output:

None

description:

Initializes the variables of the StateVariable

atom_qty

input:

self: the StateVariable object
state_variables: the mili.state_variable object that can used to map from name to the StateVaribale

output:

qty: the quantity of variables in the StateVariable

description:

Loops over the svars and uses the order and dims to find the total number of state variables (“atoms”) in the StateVariable

Subrecord

Description

A group of StateVariables.

qty_blocks: the number of sections of elements (e.g. 3)
mo_blocks: the sections of elements (e.g. 1-5, 7-9, 11-12)
mo_qty: the total number of elements (e.g. 10)
offset: the offset of the Subrecord
size: the size of the Subrecord

__init__

input:

self: the Subrecord object

name: the name of the Subrecord
class_name: the class name
organization: either result or object ordered
qty_svars: number of state variables in the Subrecord
svar_names: names of the state variables Subrecord

output:

None

description:

Initializes the variables of the Subrecord

SubrecordContainer

Description

A group of Subrecords.

subrecs: the array of subrecords
size: the combined size of the subrecords

MeshObjectClassData

Description

The mesh data needed for the mili file.

__init__

input:

self: the MeshObjectClassData object
short_name: the shorter name of the MeshObjectClassData
long_name: the longer name of the MeshObjectClassData
superclass: the superclass of the class
blocklist: the BlockList for this MeshObjectClassData

output:

None

description:

Initializes the variables of the MeshObjectClassData

add_block

input:

self: the MeshObjectClassData object
start: the start element
stop: the stop element

output:

None

description:

Adds the start, stop section to the BlockList

BlockList

Description

Contains the sections of elements for a MeshObjectClassData.

__init__

input:

self: the BlockList object
obj_qty: the number of elements in blocks
block_qty: the length of blocks
blocks: a list of tuples, where each tuple is a start, stop

output:

None

description:

Initializes the variables of the BlockList

Item

Description

A single piece of an Answer. Any number of its variables may end up unitialized. Often the Item represents a StateVariable.

always_print: whether or not this Item is always printed in an Answer

__init__

input:

self: the Item object
name: the name of the Item
material: the material of the Item
mo_id: the element id of the Item
label: the label of the Item
class_name: the class name
modify: whether or not the Item was modified
value: the value of the Item

output:

None

description:

Initializes the variables of the Item

set

input:

self: the Item object
value: the value of the Item

output:

None

description:

Sets the `value` and also sets `always_print`

__str__

input:

`self`: the `Item` object

output:

`ret`: the string representation of the `Item`

description:

Displays the `Item` in a readable format for outputting the answer to the screen or to a file.

StateAnswer

Description

The representation of all the data for a state, used in a state based `query`.

`items`: a list of `Item` objects
`state_number`: the state number of the `StateAnswer`

__init__

input:

`self`: the `StateAnswer` object

output:

None

description:

Initializes the variables of the `StateAnswer`

__str__

input:

`self`: the `StateAnswer` object

output:

`ret`: the string representation of the `StateAnswer`

description:

Displays the `StateAnswer` in a readable format for outputting the answer to the screen or to a file.

Answer

Description

The return value of a query that contains all information requested.

state_answers: list of StateAnswer objects

__init__

input:

self: the Answer object

output:

None

description:

Initializes the variables of the Answer

set

input:

self: the Answer object
names: the names of the Answer
materials: the material sof the Answer
mo_ids: the element ids of the Answer
labels: the labels of the Answer
class_name: the class name
modify: whether or not the Answer was modified

output:

None

description:

If this Answer is not state based, this function creates a list of Item objects and sets the self.items

__str__

input:

self: the Answer object

output:

ret: the string representation of the Answer

description:

Displays the Answer in a readable format for outputting the Answer to the screen or to a file.

Miscellaneous

Directory Type

Description

A mapping of the string directory types to the integer representation

Superclass

Description

A mapping of the string superclass types to the integer representation

ConnWords

Description

A mapping of the string connection types to the integer representation

DataType

Description

A mapping of the string data types to the integer representation

ExtSize

Description

A mapping of the string data types to the size in bytes

Aggregate Type

Description

A mapping of the string aggregate types to the integer representation

Data Organization

Description

A mapping of the string data organization types to the integer representation

Mili Python Interface Tests

Description

These tests all use the d3samp6.dyn file as the basis for the tests. They cover the basic functionality.

test_invalid_inputs

Testing invalid inputs to the functions don't cause a crash

test_element_number_material

Testing what element numbers associated with a material

test_nodes_material

Testing Testing what nodes are associated with a material

test_nodes_label

Testing what nodes are associated with a label

test_state_varialble

Testing accessing a variable at a given state

test_node_attributes

Testing accessing accessing node attributes -> this is a vector component
Tests both ways of accessing vector components (using brackets vs not)
e.g. nodpos[ux] and ux

test_state_variable_vector

Testing the accessing of a vector, in this case node position

test_modify_state_variable

Testing the modification of a scalar state variable

test_modify_vector

Testing the modification of a vector state variable

test_modify_vector_component

Testing the modification of a vector component

test_state_variable_vector_array

Testing accessing a vector array

test_state_variable_vector_array_component

Testing accessing a vector array component

test_modify_vector_array

Testing modifying a vector array

test_modify_vector_array_component

Test modifying a vector array component

User’s Guide

Description

This program can be imported and initialized using the following:

```
import read

mili = read.Mili()

mili.read(file_name)
```

From this point, the mili object contains all the information from the mili file and can be queried.

```
answer = mili.nodes_of_elem(1, “brick)

answer2 = mili.query(['nodpos[ux]'], ‘node’, None, 3, 4)

directories = mili.getDirectories()
```

Mili Class

Description

The Mili Class has variables that store the pertinent information for a mili file. These variables are filled during the various functions that read in the Mili file and will then be used when querying. If you want to read multiple Mili files, you should create new Mili objects for each of these separate files.

Reader Functions

read

input:

`self`: the mili object
`file_name`: the name of the problem to be run (e.g. bar1)

output:

None

description:

Calls the other reader functions to build up a Mili object. Capable of handling multiple state map files.

```
mili.read(file_name)

# The mili object now has all the contents of the Mili file at file_name
```

Query Functions

query

input:

`self`: the mili object

`names` (required): the names of the state variables. Can either be a string or list of strings

`class_name` (required): the class_name of the result. Must be a string

`material`: the material in the result

if this value is nonzero – looks for all labels matching this `material` and `class_name.` If some labels are also included, includes only the matching labels of input material. Must be a string

`labels`: the labels in the result

if this value is not entered – gets all labels pertaining to the class. Must be a list of ints or a single int

`state_numbers`: the state_numbers in the result. Must be a list of ints or a single int

`modify`: whether or not there were modifications. Must be Boolean

`int_points`: the integration points. Must be a list of ints or a single int

`raw_data`: whether or not to simply output raw data, not `Answer`. Must be Boolean. By default is True.

`res`: the result. This is used in the parallel querying and not by the user.

output:

if `raw_data:`

return raw info – can either be a list or a dictionary containing the info with the format:

`res[state][name][label] = value`

else:

answer: an `Answer` containing the information from the query

description:

Searches for the given state variables at specified states, labels, etc. First, there is code that checks the input. Then parses information from the state file(s) to get the information.

```
stresses = mili.query(‘sx’, ‘brick’, None, [4,6,7], [29])

# Queries the mili object for state variable sx with class brick at state 29 on the labels 4, 6,
and 7. Will return the dictionary res structure described above.

stresses = mili.query('sx', 'brick', 2, None, [37])

# Queries the mili object for state variable sx with class brick at state 37 on all labels of
material 2. Will return the dictionary res structure described above.

node_positions = mili.query(['nodpos'], 'node', None, [4], [3])

# Queries the mili object for node positions at state 3 for label 4. Will return the dictionary
res structure described above.
```

labels_of_material

input:

`self`: the mili object
`material`: the name or number of the material
`raw_data`: whether or not to simply output raw data, not `Answer`. Default value is true.

output:

answer: an `Answer` containing the elements of the specified material

description:

Given a material name or number, finds all the labels of elements that are of this material.

```
element_ids = mili.elements_of_material(‘es_1’)

# Returns the element ids of all the elements of the specified material.
```

nodes_of_material

input:

`self`: the mili object
`material`: the name or number of the material
`class_name`: the class name
`raw_data`: whether or not to simply output raw data, not `Answer.` Default value is true.

output:

answer: an `Answer` containing the nodes of the specified material and class

description:

Given a material name or number and class name, finds all the nodes.

```
nodes = mili.nodes_of_material(‘es_1’, ‘brick’)

# Returns the node numbers of the specified material and class
```

nodes_of_elem

input:

`self`: the mili object
`label`: the label of the element
`class_name`: the class name
`raw_data`: whether or not to simply output raw data, not `Answer.` Default value is true.

output:

answer: an `Answer` containing the nodes of the specified material and class

description:

Given a label and class name, finds all the nodes.

```
nodes = mili.nodes_of_elem(3, ‘brick’)

# Returns the node numbers that make up the specified object
```

Other Functions

modify_state_variable

input:

`self`: the mili object
`state_variable:` the names of the state variable to modify. Must be a string
`class_name`: the class_name of the result. Must be a string
`value`: the value to assign

dictionary `d` with the following organization:

if **scalar** (or single component of a vector):

```
value = d[state_number][state_variable_name][label] = val

    where val is an integer/float/etc.

value = {3 : {'matke' : {1 : 5.5}}}
value = {3 : {'nodpos[uz]' : {4 : 9.0, 5: 9.0}}}
```

if **vector**:

```
value = d[state_number][state_variable_name][label] = val
        where val is an array

val = {3 : {'nodpos' : {4 : [5.0, 6.0, 9.0], 5: [5.0, 6.0, 9.0]}}}
```

if **vector array**:

```
value = d[state_number][state_variable_name][label] = val
        where val is a dictionary that maps:

                val = {sv_name : {integration point: value}}


d = {70 : {'stress' : {5 : {'sz': {2: -2.0}, 'sy': {2: -2.5}, 'sx':  {2:
1.3}, 'szx': {2: -4.8}, 'sxy': {2: 6.9}, 'syz': {2: 1.0}}}}}
```

note: value can contain more than simply the entries to enter – what values are changed depends on the other inputs to this function

labels: the labels that should be modified. Must be an int or list of ints

state_numbers: the state_numbers in the result. Must be a list of ints or a single int

int_points: the integration points. Must be a list of ints or a single int

output:

None

description:

Modifies the state variable at the given state(s) and label(s). Uses the dictionary structure passed in to value after figuring out the proper indexes in each subrecord.

getParams

input:

self: the mili object

output:

params: the mili params object

description:

Getter for params

getStateMaps

input:

self: the mili object

output:

state_maps: the mili state map object

description:

Getter for state maps

getDirectories

input:

`self`: the mili object

output:

`directories`: the mili directories object

description:

Getter for directories

getStateVariables

input:

`self`: the mili object

output:

`state_variables`: the mili params object

description:

Getter for state variables

getLabels

input:

`self`: the mili object

output:

`labels`: the mili params object

description:

Getter for labels

getMaterials

input:

`self`: the mili object

output:

`materials`: the mili materials object

description:

Getter for materials

setErrorFile

input:

`self`: the mili object

output:

`file_name`: the file name for the error output file

description:

Instead of being displayed to the screen, it will be redirected to an output file.