Mili Python Interface – Users Guide

Table of Contents

User's Guide

Description

First, you need to make sure the file mili_reader_lib.py is in your path. On the open side, the file is located at collab/usr/gapps/mdg/bin and on the close side at usr/apps/mdg/bin

There are two options.

Option 1:
Have the following two lines at the beginning of your file. This will need to be done whenever you want another file that uses the library.

```
import sys
sys.path.append('/collab/usr/gapps/mdg/bin') #for open side
```

Option 2:

This option only must be done once. Add the directory location to your $PYTHONPATH
For bash, add the following to .profile (do the equivalent for your shell):

```
PYTHONPATH=/collab/usr/gapps/mdg/bin:${PYTHONPATH-''}
export PYTHONPATH
```

Now, the program can be imported and initialized using the following:

```
import mili_reader_lib_version
mili = mili_reader_lib_version.Mili()
mili.read(file_name)
```

The version number will change over time. The current version is **1.0** and import is:

```
import mili_reader_lib_1_0
```

Equivalently:

Parallel Mili File

```
import mili_reader_lib_version

mili = mili_reader_lib_version.Mili(file_name)
```

If you are using a parallel mili 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.

Parallel Reading/Writing of Parallel Mili File

If you choose to run with `parallel_read` on, which is specified either in the mili constructor call or the read call. If `parallel_read` is `True`, the reader starts a Python threaded process for each mili file and these processes are maintained for reading and writing.

```
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. 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.plt)
`labeltomili`: the data structure storing a dictionary from labels to mili file numbers
`mili_num`: the number of this mili file
`parallel_read`: whether or not to run this job in parallel (default False)

output:

None

description:

Calls the other reader functions to build up a Mili object. Capable of handling multiple state map files. If `parallel_read` is True, a process is started for each parallel mili file. The querying is unchaged by this from the users perspective. If this option is True, the commands to create a mili reader object, read a mili file, and query it should be run in an mxterm or in batch mode, not on a login node.

```
mili.read(file_name, parallel_read=True)

# The mili object now has all the contents of the Mili file at file_name. Will run in parallel and create a process for each mili file
```

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.