Basic Model Interface Documentation

Release 0.1

Eric Hutton

June 23, 2016

Contents

Release 0.1

Date June 23, 2016

CHAPTER 1

Contents

1.1 bmi.base module

Interface to the basic control functions of a model.

class bmi.base.**BmiBase**

Bases: object

Functions that control model execution.

These BMI functions are critical to plug-and-play modeling because they give a calling component fine-grained control over the model execution.

Methods

finalize()	Perform tear-down tasks for the model.
initialize(filename)	Perform startup tasks for the model.
update()	Advance model state by one time step.
update_frac(time_frac)	Advance model state by a fraction of a time step.
update_until(time)	Advance model state until the given time.

finalize()

Perform tear-down tasks for the model.

Perform all tasks that take place after exiting the model's time loop. This typically includes deallocating memory, closing files and printing reports.

Notes

```
/* C */
int finalize(void *self);
```

initialize(*filename*)

Perform startup tasks for the model.

Perform all tasks that take place before entering the model's time loop, including opening files and initializing the model state. Model inputs are read from a text-based configuration file, specified by *filename*.

Parameters **filename** : str, optional

The path to the model configuration file.

Notes

Models should be refactored, if necessary, to use a configuration file. CSDMS does not impose any constraint on how configuration files are formatted, although YAML is recommended. A template of a model's configuration file with placeholder values is used by the BMI.

```
/* C */
int initialize(void *self, char * filename);
```

update()

Advance model state by one time step.

Perform all tasks that take place within one pass through the model's time loop. This typically includes incrementing all of the model's state variables. If the model's state variables don't change in time, then they can be computed by the *initialize()* method and this method can return with no action.

Notes

```
/* C */
int update(void *self);
```

update_frac(*time_frac*)

Advance model state by a fraction of a time step.

Parameters **time_frac** : float

A fraction of a model time step value.

See also:

update

Notes

```
/* C */
int update_frac(void *self, double time_frac);
```

update_until(*time*)

Advance model state until the given time.

Parameters **time** : float

A model time value.

See also:

update

Notes

```
/* C */
int update_until(void *self, double time);
```

1.2 bmi.info module

Interface that describes a model and it's input and output variables.

class bmi.info.**BmiInfo**
Bases: object

Get metadata about a model.

Methods

get_component_name()	Name of the component.
get_input_var_names()	List of a model's input variables.
get_output_var_names()	List of a model's output variables.

get_component_name()
Name of the component.

Returns str

The name of the component.

Notes

```
/* C */
int get_component_name(void * self, char * name);
```

get_input_var_names()
List of a model's input variables.

Input variable names must be CSDMS Standard Names, also known as *long variable names*.

Returns list of str

The input variables for the model.

Notes

```
/* C */
int get_input_var_name_count(void * self, int * count);
int get_input_var_names(void * self, char ** names);
```

get_output_var_names()
List of a model's output variables.

Output variable names must be CSDMS Standard Names, also known as *long variable names*.

Returns list of str

The output variables for the model.

See also:

get_input_var_names

Notes

```
/* C */
int get_output_var_name_count(void * self, int * count);
int get_output_var_names(void * self, char ** names);
```

1.3 bmi.time module

Interface that describes the time stepping of a model.

class bmi.time.**BmiTime**
Bases: object

Methods that get time information from a model.

Methods

get_current_time()	Current time of the model.
get_end_time()	End time of the model.
get_start_time()	Start time of the model.
get_time_step()	Current time step of the model.
get_time_units()	Time units of the model.

get_current_time()
Current time of the model.

Returns float

The current model time.

See also:

get_start_time

Notes

```
/* C */
int get_current_time(void * self, double * time);
```

get_end_time()
End time of the model.

Returns float

The maximum model time.

See also:

get_start_time

Notes

```
/* C */
int get_end_time(void * self, double * time);
```

get_start_time()
Start time of the model.

Model times should be of type float. The default model start time is 0.

Returns float

The model start time.

Notes

```
/* C */
int get_start_time(void * self, double * time);
```

get_time_step()
Current time step of the model.

The model time step should be of type float. The default time step is 1.0.

Returns float

The time step used in model.

Notes

```
/* C */
int get_time_step(void * self, double * dt);
```

get_time_units()
Time units of the model.

Returns float

The model time unit; e.g., *days* or *s*.

Notes

```
/* C */
int get_time_units(void * self, char * units);
```

1.4 bmi.vars module

Interface that describes a model's input and output variables.

class bmi.vars.**BmiVars**
Bases: object

Methods that get information about input and output variables.

These BMI functions obtain information about a particular input or output variable. They must accommodate any variable that is returned by the BMI functions *get_input_var_names()* or *get_output_var_names()*.

Methods

get_var_grid(var_name)	Get grid identifier for the given variable.
get_var_itemsize(var_name)	Get memory use for each array element in bytes.
get_var_nbytes(var_name)	Get size, in bytes, of the given variable.
get_var_type(var_name)	Get data type of the given variable.
get_var_units(var_name)	Get units of the given variable.

get_var_grid(*var_name*)

Get grid identifier for the given variable.

Parameters **var_name** : str

An input or output variable name, a CSDMS Standard Name.

Returns int

The grid identifier.

See also:

bmi.info.BmiInfo.get_input_var_names Get *var_name* from this method or from *get_output_var_names()*.

Notes

```
/* C */
int get_var_grid(void * self, const char * var_name, int * id);
```

get_var_itemsize(*var_name*)

Get memory use for each array element in bytes.

Parameters **var_name** : str

An input or output variable name, a CSDMS Standard Name.

Returns int

Item size in bytes.

Notes

```
/* C */
int get_var_itemsize(void * self, const char * var_name,
                     int * itemsize);
```

get_var_nbytes(*var_name*)

Get size, in bytes, of the given variable.

Parameters **var_name** : str

An input or output variable name, a CSDMS Standard Name.

Returns int

The size of the variable, counted in bytes.

Notes

```
/* C */
int get_var_nbytes(void * self, const char * var_name,
                   int * nbytes);
```

get_var_type(*var_name*)
Get data type of the given variable.

Parameters **var_name** : str

An input or output variable name, a CSDMS Standard Name.

Returns str

The Python variable type; e.g., `str`, `int`, `float`.

Notes

```
/* C */
int get_var_type(void * self, const char * var_name, char * type);
```

get_var_units(*var_name*)
Get units of the given variable.

Standard unit names, in lower case, should be used, such as `meters` or `seconds`. Standard abbreviations, like `m` for meters, are also supported. For variables with compound units, each unit name is separated by a single space, with exponents other than 1 placed immediately after the name, as in `m s-1` for velocity, `W m-2` for an energy flux, or `km2` for an area.

Parameters **var_name** : str

An input or output variable name, a CSDMS Standard Name.

Returns str

The variable units.

Notes

CSDMS uses the UDUNITS standard from Unidata.

```
/* C */
int get_var_units(void * self, const char * var_name,
                  char * units);
```

1.5 bmi.getter_setter module

Interface for getting and setting a model's internal variables.

class bmi.getter_setter.**BmiGetter**
Bases: object

Get values from a component.

Methods that get variables from a model's state. Often a model's state variables are changing with each time step, so getters are called to get current values.

Methods

get_value(var_name)	Get a copy of values of the given variable.
get_value_at_indices(var_name, indices)	Get values at particular indices.
get_value_ref(var_name)	Get a reference to values of the given variable.

get_value(*var_name*)
Get a copy of values of the given variable.

This is a getter for the model, used to access the model's current state. It returns a *copy* of a model variable, with the return type, size and rank dependent on the variable.

Parameters **var_name** : str

An input or output variable name, a CSDMS Standard Name.

Returns array_like

The value of a model variable.

Notes

```
/* C */
int get_value(void * self, const char * var_name, void * buffer);
```

get_value_at_indices(*var_name*, *indices*)
Get values at particular indices.

Parameters **var_name** : str

An input or output variable name, a CSDMS Standard Name.

indices : array_like

The indices into the variable array.

Returns array_like

Value of the model variable at the given location.

Notes

```
/* C */
int get_value_at_indices(void * self, const char * var_name,
                         void * buffer, int * indices, int len);
```

get_value_ref(*var_name*)
Get a reference to values of the given variable.

This is a getter for the model, used to access the model's current state. It returns a reference to a model variable, with the return type, size and rank dependent on the variable.

Parameters **var_name** : str

An input or output variable name, a CSDMS Standard Name.

Returns array_like

A reference to a model variable.

Notes

```
/* C */
int get_value_ref(void * self, const char * var_name,
                  void ** buffer);
```

class bmi.getter_setter.**BmiSetter**

Bases: object

Set values into a component.

Methods that set variables of a model's state.

Methods

set_value(var_name, src)	Specify a new value for a model variable.
set_value_at_indices(var_name, indices, src)	Specify a new value for a model variable at particular indices.

set_value (*var_name*, *src*)

Specify a new value for a model variable.

This is the setter for the model, used to change the model's current state. It accepts, through *src*, a new value for a model variable, with the type, size and rank of *src* dependent on the variable.

Parameters **var_name** : str

An input or output variable name, a CSDMS Standard Name.

src : array_like

The new value for the specified variable.

Notes

```
/* C */
int set_value(void * self, const char * var_name, void * src);
```

set_value_at_indices (*var_name*, *indices*, *src*)

Specify a new value for a model variable at particular indices.

Parameters **var_name** : str

An input or output variable name, a CSDMS Standard Name.

indices : array_like

The indices into the variable array.

src : array_like

The new value for the specified variable.

Notes

```
/* C */
int set_value_at_indices(void * self, const char * var_name,
                         int * indices, int len, void * src);
```

1.6 bmi.grid_uniform_rectilinear module

Interface that describes uniform rectilinear grids.

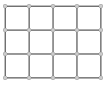
class bmi.grid_uniform_rectilinear.**BmiGridUniformRectilinear**

Bases: bmi.grid.BmiGrid

Methods that describe a uniform rectilinear grid.

In a 2D uniform grid, every grid cell (or element) is a rectangle and all cells have the same dimensions. If the dimensions are equal, then the grid is a tiling of squares.

Each of these functions returns information about each dimension of a grid. The dimensions are ordered with "ij" indexing (as opposed to "xy"). For example, the *get_grid_shape()* function for the example grid would return the array `[4, 5]`. If there were a third dimension, the length of the z dimension would be listed first. This same convention is used in NumPy. Note that the grid shape is the number of nodes in the coordinate directions and not the number of cells or elements. It is possible for grid values to be associated with the nodes or with the cells.



Methods

get_grid_origin(grid_id)	Get coordinates for the origin of the computational grid.
get_grid_rank(grid_id)	Get number of dimensions of the computational grid.
get_grid_shape(grid_id)	Get dimensions of the computational grid.
get_grid_size(grid_id)	Get the total number of elements in the computational grid.
get_grid_spacing(grid_id)	Get distance between nodes of the computational grid.
get_grid_type(grid_id)	Get the grid type as a string.

get_grid_origin(*grid_id*)

Get coordinates for the origin of the computational grid.

Parameters **grid_id** : int

A grid identifier.

Returns array_like

The coordinates of the lower left corner of the grid.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

```
/* C */
int get_grid_origin(void * self, int grid_id, double * origin);
```

get_grid_shape(*grid_id*)
Get dimensions of the computational grid.

Parameters **grid_id** : int

A grid identifier.

Returns array_like

The dimensions of the grid.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

```
/* C */
int get_grid_shape(void * self, int grid_id, int * shape);
```

get_grid_spacing(*grid_id*)
Get distance between nodes of the computational grid.

Parameters **grid_id** : int

A grid identifier.

Returns array_like

The grid spacing.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

```
/* C */
int get_grid_spacing(void * self, int grid_id, double * spacing);
```

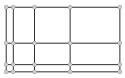
1.7 bmi.grid_rectilinear module

Interface that describes rectilinear grids.

class bmi.grid_rectilinear.**BmiGridRectilinear**
Bases: bmi.grid.BmiGrid

Methods that describe a rectilinear grid.

In a 2D rectilinear grid, every grid cell (or element) is a rectangle but different cells can have different dimensions. All cells in the same row have the same grid spacing in the y direction and all cells in the same column have the same grid spacing in the x direction. Grid spacings can be computed as the difference of successive x or y values.



Methods

`get_grid_rank`(grid_id)	Get number of dimensions of the computational grid.
`get_grid_shape`(grid_id)	Get dimensions of the computational grid.
`get_grid_size`(grid_id)	Get the total number of elements in the computational grid.
`get_grid_type`(grid_id)	Get the grid type as a string.
`get_grid_x`(grid_id)	Get coordinates of grid nodes in the streamwise direction.
`get_grid_y`(grid_id)	Get coordinates of grid nodes in the transverse direction.
`get_grid_z`(grid_id)	Get coordinates of grid nodes in the normal direction.

get_grid_shape(*grid_id*)

Get dimensions of the computational grid.

Parameters **grid_id** : int

A grid identifier.

Returns tuple of int

The dimensions of the grid.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

```
/* C */
int get_grid_shape(void * self, const char * var_name,
                   int * shape);
```

get_grid_x(*grid_id*)

Get coordinates of grid nodes in the streamwise direction.

Parameters **grid_id** : int

A grid identifier.

Returns array_like of float

The positions of the grid nodes.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

```
/* C */
int get_grid_x(void * self, const char * var_name, double * x);
```

get_grid_y(*grid_id*)
Get coordinates of grid nodes in the transverse direction.

Parameters **grid_id** : int

A grid identifier.

Returns array_like of float

The positions of the grid nodes.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

```
/* C */
int get_grid_y(void * self, const char * var_name, double * y);
```

get_grid_z(*grid_id*)
Get coordinates of grid nodes in the normal direction.

Parameters **grid_id** : int

A grid identifier.

Returns array_like of float

The positions of the grid nodes.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

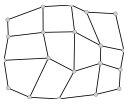
```
/* C */
int get_grid_z(void * self, const char * var_name, double * z);
```

1.8 bmi.grid_structured_quad module

Interface that describes structured quadrilateral grids.

class bmi.grid_structured_quad.**BmiGridStructuredQuad**
Bases: bmi.grid.BmiGrid

Methods that describe a structured grid of quadrilaterals.



Methods

get_grid_rank(grid_id)	Get number of dimensions of the computational grid.
get_grid_shape(grid_id)	Get dimensions of the computational grid.
get_grid_size(grid_id)	Get the total number of elements in the computational grid.
get_grid_type(grid_id)	Get the grid type as a string.
get_grid_x(grid_id)	Get coordinates of grid nodes in the streamwise direction.
get_grid_y(grid_id)	Get coordinates of grid nodes in the transverse direction.
get_grid_z(grid_id)	Get coordinates of grid nodes in the normal direction.

get_grid_shape(*grid_id*)

Get dimensions of the computational grid.

Parameters **grid_id** : int

A grid identifier.

Returns array_like

The dimensions of the grid.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

```
/* C */
int get_grid_shape(void * self, int grid_id, int * shape);
```

get_grid_x(*grid_id*)

Get coordinates of grid nodes in the streamwise direction.

Parameters **grid_id** : int

A grid identifier.

Returns array_like

The positions of the grid nodes.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

```
/* C */
int get_grid_x(void * self, int grid_id, double * x);
```

get_grid_y(*grid_id*)

Get coordinates of grid nodes in the transverse direction.

Parameters **grid_id** : int

A grid identifier.

Returns array_like

The positions of the grid nodes.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

```
/* C */
int get_grid_y(void * self, int grid_id, double * y);
```

get_grid_z(*grid_id*)
Get coordinates of grid nodes in the normal direction.

Parameters **grid_id** : int

A grid identifier.

Returns array_like

The positions of the grid nodes.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

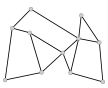
```
/* C */
int get_grid_z(void * self, int grid_id, double * z);
```

1.9 bmi.grid_unstructured module

Interface that describes unstructured grids.

class bmi.grid_unstructured.**BmiGridUnstructured**
Bases: bmi.grid.BmiGrid

Methods that describe an unstructured grid.



Methods

get_grid_connectivity(grid_id)	Get connectivity array of the grid.
get_grid_offset(grid_id)	Get offsets for the grid nodes.
get_grid_rank(grid_id)	Get number of dimensions of the computational grid.
get_grid_size(grid_id)	Get the total number of elements in the computational grid.
get_grid_type(grid_id)	Get the grid type as a string.
get_grid_x(grid_id)	Get coordinates of grid nodes in the streamwise direction.

Continued on next page

Table 1.10 – continued from previous page

get_grid_y(grid_id)	Get coordinates of grid nodes in the transverse direction.
get_grid_z(grid_id)	Get coordinates of grid nodes in the normal direction.

get_grid_connectivity(*grid_id*)

Get connectivity array of the grid.

Parameters **grid_id** : int

A grid identifier.

Returns array_like or int

The graph of connections between the grid nodes.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

```
/* C */
int get_grid_connectivity(void * self, int grid_id,
                          int * connectivity);
```

get_grid_offset(*grid_id*)

Get offsets for the grid nodes.

Parameters **grid_id** : int

A grid identifier.

Returns array_like of int

The offsets for the grid nodes.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

```
/* C */
int get_grid_offset(void * self, int grid_id, int * offset);
```

get_grid_x(*grid_id*)

Get coordinates of grid nodes in the streamwise direction.

Parameters **grid_id** : int

A grid identifier.

Returns array_like

The positions of the grid nodes.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

```
/* C */
int get_grid_x(void * self, int grid_id, double * x);
```

get_grid_y(*grid_id*)

Get coordinates of grid nodes in the transverse direction.

Parameters **grid_id** : int

A grid identifier.

Returns array_like

The positions of the grid nodes.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

```
/* C */
int get_grid_y(void * self, int grid_id, double * y);
```

get_grid_z(*grid_id*)

Get coordinates of grid nodes in the normal direction.

Parameters **grid_id** : int

A grid identifier.

Returns array_like

The positions of the grid nodes.

See also:

bmi.vars.BmiVars.get_var_grid Obtain a *grid_id*.

Notes

```
/* C */
int get_grid_z(void * self, int grid_id, double * z);
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

b

B

F

G

I

S

U