The Basic Model Interface 2.0: A standard interface for coupling numerical models and data in the hydrologic sciences

- Eric W. H. Hutton (eric.hutton@colorado.edu)
- Mark D. Piper
- Tian Gan
- Greg E. Tucker

University of Colorado, Boulder, CO USA

Background

The hydrologic modeling and data community has embraced the open source movement as evidenced by the ever increasing number of ***FAIR*** models and datasets available to investigators. Although this has resulted in new science through innovative model application, development, and coupling, the ***idiosyncratic design*** of many of these models and datasets acts as a speed bump that ***slows the time-to-science***.

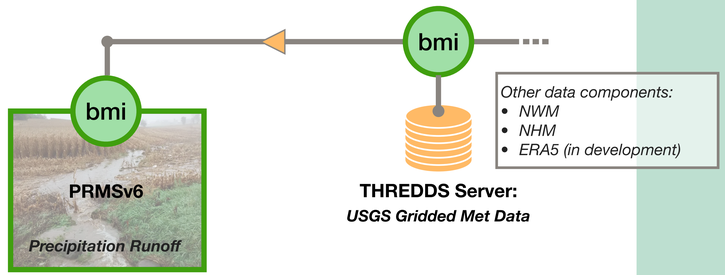
BMI is a library specification to simplify model-to-model and model-to-data coupling

The Basic Model Interface version 2.0 (BMI) specification defines a standardized interface for both models and data. This allows all models and datasets with a BMI to look alike, regardless of their underlying implementation or, in fact, even if they are truly a model or a dataset. With idiosyncratic implementation details obscured, models and data are more easily and quickly picked up and used - if you know how to use one BMI model, you know how to use any BMI model.

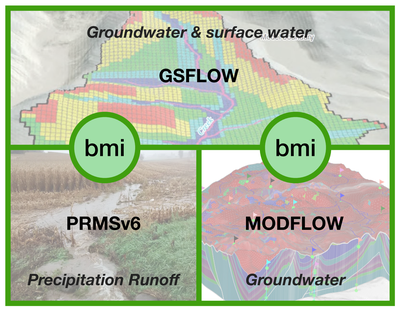
The BMI is not just for models. Data components allow datasets to be interchangeable with models

Although ***BMI*** was initially written as an interface for numerical models, we have extended it to also be able to include ***datasets***.

This allows investigators to be able, in a reproducible way: compare models to one another using a common dataset, validate models to data, ingest data into a model, swap models and data within a workflow.



(***above***) The BMI enabled PRMS (v6) model is driven by a BMI enabled Gridded Met Data component. (***below***) The PRMS model is part of a larger collection of BMI-enabled models.



The **Basic Model Interface** (*BMI*) is a standardized set of functions that allows **coupling** of models to **models** and models to **data**

bmi

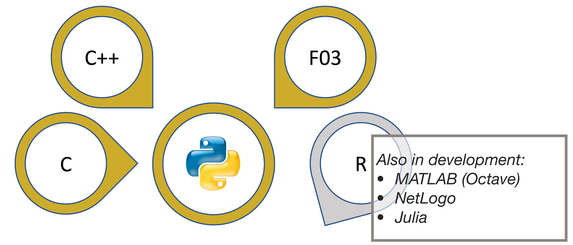
The Python Modeling Toolkit (*pymt*)

pymt is the **Python Modeling Toolkit**. It is an Open Source Python package, developed by the **Community Surface Dynamics Modeling System** (*CSDMS*), that provides the tools needed for coupling models that expose the **Basic Model Interface** (*BMI*).

pymt in three points:

- Tools for coupling models/data of disparate time and space scales and computational grids
- A collection of Earth-surface models, written in a range of languages, brought together in an easy-to-use python interface
- Extensible plug-in framework for adding new models

pymt supports BMIs from a range of languages



The *BMI* specification as *SIDL*.

Below is what the *BMI* specification looks like in the Scientific Interface Definition Language (*SIDL*).

```
package csdms version 2.0b0 {
  interface bmi {

    // Model control functions: Initialize, run, finalize (IRF)
    int initialize(in string config_file);
    int update();
    int update_until(in double time);
    int finalize();

    // Model information functions: Metadata about model's exchange items
    int get_component_name(out string name);
    int get_input_item_count(out int count);
    int get_output_item_count(out int count);
    int get_input_var_names(out array<string, 1> names);
    int get_output_var_names(out array<string, 1> names);

    // Variable functions: Metadata about exchange items
    int get_var_grid(in string name, out int grid);
    int get_var_type(in string name, out string type);
    int get_var_units(in string name, out string units);
    int get_var_itemsize(in string name, out int size);
    int get_var_nbytes(in string name, out int nbytes);
    int get_var_location(in string name, out string location);

    // Variable functions: Metadata about exchange items
    int get_current_time(out double time);
    int get_start_time(out double time);
    int get_end_time(out double time);
    int get_time_units(out string units);
    int get_time_step(out double time_step);

    // Getters: Get variables from a model
    int get_value(in string name, in array<> dest);
    int get_value_ptr(in string name, out array<> dest_ptr);
    int get_value_at_indices(in string name, in array<> dest,
        in array<int, 1> inds);

    // Setters: Set values into a model
    int set_value(in string name, in array<> src);
    int set_value_at_indices(in string name, in array<int, 1> inds,
        in array<> src);

    // Grid information: Description of model grids
    int get_grid_rank(in int grid, out int rank);
    int get_grid_size(in int grid, out int size);
    int get_grid_type(in int grid, out string type);

    // Uniform rectilinear
    int get_grid_shape(in int grid, in array<int, 1> shape);
    int get_grid_spacing(in int grid, in array<double, 1> spacing);
    int get_grid_origin(in int grid, in array<double, 1> origin);

    // Non-uniform rectilinear, curvilinear
    int get_grid_x(in int grid, in array<double, 1> x);
    int get_grid_y(in int grid, in array<double, 1> y);
    int get_grid_z(in int grid, in array<double, 1> z);

    // Unstructured
    int get_grid_node_count(in int grid, out int count);
    int get_grid_edge_count(in int grid, out int count);
    int get_grid_face_count(in int grid, out int count);
    int get_grid_edge_nodes(in int grid, out array<int, 1> edge_nodes);
    int get_grid_face_edges(in int grid, out array<int, 1> face_edges);
    int get_grid_face_nodes(in int grid, out array<int, 1> face_nodes);
    int get_grid_nodes_per_face(in int grid, out array<int, 1> nodes_per_face);
  }
}
```