Extensions to the Basic Model Interface and Serialization of a Fortran Model's State Variables for Load Balancing and Checkpointing

Scott D. Peckham[1,4], Zhengtao Cui[1], Jessica L. Garrett[1,3], Keith S. Jennings[1,3], Luciana Kindl da Cunha[1,6], Shengting Cui[1,2], Robert L. Bartel[1], Nels J. Frazier[1,2], Donald W. Johnson[1], Fred L. Ogden[1], Trey C. Flowers[1]

(1) NOAA National Water Center, (2) ERT Inc., (3) Lynker Technologies, (4) University of Colorado, Boulder, (5) University of Alabama, (6) West Consultants, Inc.

NextGen Water Resources Modeling Framework

In coordination with federal water prediction partners, NOAA's Office of Water Prediction (OWP) leads development of a model coupling framework called the Next Generation Water Resources Modeling Framework (Nextgen). This framework uses a non-invasive, community-standard API for computational models called the Basic Model Interface (BMI). Nextgen uses an Adapter-Mediator pattern to access model functions using calls to BMI functions. BMI function calls provide fine-grained control (initialize, update, finalize), variable getters and setters, and functions to retrieve information about a model's input and output variables (e.g. grid, rank, and units). This information allows the framework to automatically call mediators, when needed, to enable passing values of variables between models (e.g. for regridding, time interpolation and unit conversion). BMI supports models written in C, C++, Fortran and Python.

Extensions to the Basic Model Interface (BMI)

In previous work, we introduced a variable's **role** (e.g., input, output, file info, calibration parameter, etc.) as a new attribute and several new functions to BMI to support tasks such as model **calibration**, and **serialization/deserialization** of all of a model's state variables.

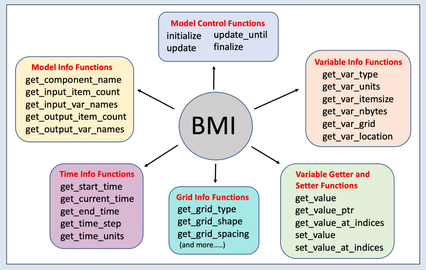
A new role-based get_var_count() generalizes the BMI v. 2.0 functions: get_input_item_count() and get_output_item_count().

Similarly, a new get_var_names() generalizes the BMI v. 2.0 functions: get_input_var_names() and get_output_var_names().

Variable Information Functions (see orange box in figure to the right) now return info for ***any*** model variable, not just input and output variables.

Two new functions have been added to get the ***role*** and array ***length*** (i.e., the number of array elements) for any model variable.

These proposed extensions have been submitted to the newly-formed **BMI Council**.



Serialization for Checkpointing & Load Balancing

Our extensions to BMI support both **load balancing** --- stopping a model, moving it to a different computational node and restarting it --- and **checkpointing**, or restarting a model from a checkpoint after a hardware or power failure. Both will be important when Nextgen is running many model instances on an HPC system.

We previously developed a general framework utility in C that makes use of these new BMI functions to do the work of serialization and deserialization. However, using this C utility to serialize and deserialize a general Fortran model's state requires dealing with low-level differences between C and Fortran, such as (1) how pointers are represented, (2) row-major vs. column-major order of multi-dimensional arrays, and (3) how integer data types are handled. In this work we explored options and design tradeoffs for dealing with these C-Fortran interoperability issues, such as using the ISO_C_BINDING module in Fortran or writing a separate framework utility in Fortran to serialize Fortran models.

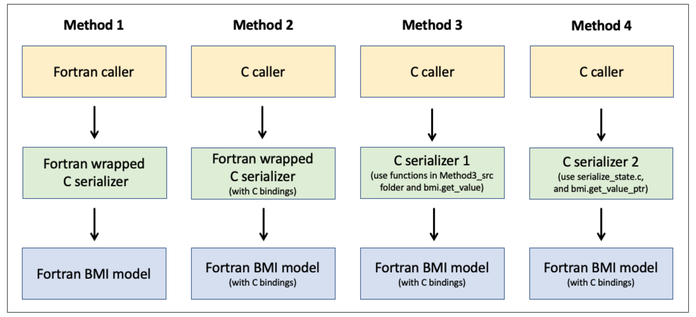


Figure 1. Summary of the 4 serialization methods tested. All 4 methods use the msgpack-c library to serialize and deserialize. The first 3 methods use bmi.get_value_*() vs. bmi.get_value_ptr_*(). "Caller" is a main program for testing that represents the framework. "Fortran BMI model" is bmi_sine.f90 wrapper for sine.f90.

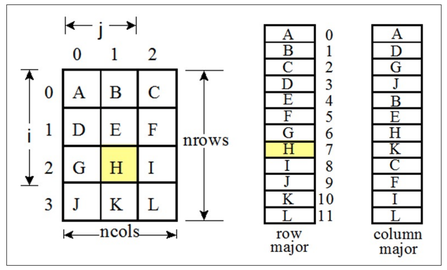


Figure 2. Row-major vs. column major ordering of the elements in a 2D array, A, that has 4 rows and 3 columns.

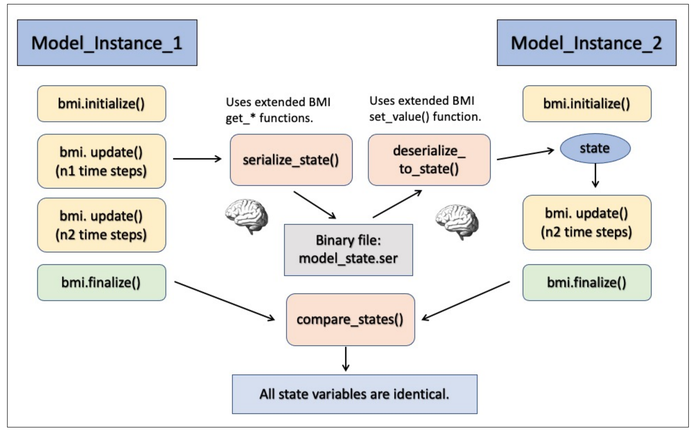


Figure 3. This figure illustrates the testing methodology for the transference of model state between two instances of the same model by using serialization and deserialization.

C – Fortran Language Interoperability Issues

There are significant differences between the C and Fortran languages that present challenges when code written in these 2 languages must be interoperable. The most important differences have to do with:

(1) Pointers, (2) Multi-dimensional arrays (row-major vs. column-major ordering and flattening, see **Figure 2**), (3) Fortran's Integer and Float "kinds", (4) Strings, and (5) Pass-by-value vs. pass-by-reference.

The **Fortran 2003 standard** introduced several new language features meant to provide better support for both C-Fortran interoperability and object-oriented programming. For C-Fortran interoperability, the main additions were the intrinsic **ISO_C_BINDING module** (with 6 intrinsic procedures such as C_LOC and C_F_POINTER) and the **bind(C)** attribute. For object-oriented programming, the main changes were the **class** statement, "type-bound" procedures, and the **extends** attribute (for "extending" a derived type). These last two features correspond to member functions and inheritance in the jargon of object-oriented programming.

The Four Fortran Model Serialization Strategies Tested

Method 1: **Fortran caller and Fortran + C serializer.** To serialize, Fortran code calls the BMI get_value() function directly to get values of the model's state variables. These are then passed to C code that performs the actual serialization using msgpack-c library. (Figure 1)

Method 2: C caller and Fortran + C serializer. Very similar to Method 1, except the caller is written in C (to emulate Nextgen framework). An extra interoperability layer between the caller and the Fortran+C serializer (using ISO_C_BINDING module) is therefore necessary.

Method 3: C caller and C serializer #1. Caller and serializer are both written in C, and C-Fortran interoperability (using ISO_C_BINDING module) is needed for these to interoperate with the BMI-enabled Fortran model. BMI get_value() is used to get state variable values.

Method 4: C caller and C serializer #2. Very similar to Method 3, except the BMI function get_value_ptr() (instead of get_value()) is used to get the values of the state variables.

Implementing bmi.get_value_ptr() in Fortran 2003

The following method is very easy to implement and does not require rewriting the Fortran model so that all variables have the POINTER or TARGET attribute.

Step 1. *Apply the TARGET attribute to the entire BMI object instance.*

NOTE: The target attribute applied to an instance of a class or derived type (i.e., BMI object "this") propagates down to its members, so we can later get C pointers to them with C_LOC. If an object does not have the TARGET attribute or has not been allocated (using an ALLOCATE statement), ***no part of it can be accessed by a pointer***.

Step 2. *Use the "c_loc and c_f_pointer pattern"*

C_LOC() and C_F_POINTER() are procedures in the intrinsic ISO_C_BINDING module, in Fortran 2003. **C_LOC(X)** returns the C address of the argument, X. X must have either the POINTER or TARGET attribute. **C_F_POINTER(C_PTR, F_PTR [, SHAPE])** assigns the target of the C pointer *CPTR* to the Fortran pointer *FPTR* and specifies its shape.