Defining BMI Extensions

This is a first, rough outline-level stab at some text that defines what an Extension is and how Extensions are proposed, reviewed, and ultimately added to the overall BMI Standard.

What are BMI Extensions?

The Basic Model Interface (BMI) is meant to be a lightweight interface standard for numerical model codes, containing the minimum "common denominator" among operation, query, and variable-modification functions. Lightweight design makes it easy for developers to implement BMI in their codes. At the same time, there are many use cases that call for standardization of more specialized capabilities. Examples might include handling of geospatial information, support for particular kinds of numerical solvers, functions for parallel computing, or capabilities related to particular grid types (such as the "sigma coordinates" used in some atmosphere and ocean models). In order to keep the Core BMI lightweight while also accommodating these and other needs, BMI uses Extensions. An Extension is a group of functions that is:

- Standardized but not part of the Core BMI, so it is effectively optional: a code can comply with Core BMI without needed to comply with any particular BMI Extension
- Versioned separately; for example, a code might be compatible with Core BMI 2.1 and Extension "ABC" version 0.5.9.
- *Defined* in language-agnostic form (currently using Scientific Interface Definition Language (SIDL); see **ADD REF**)
- *Specified* in particular languages, just as with Core BMI
- Fully compatible with Core BMI and other formally defined Extensions; that is, a new Extension should not conflict with any previous part of BMI or its Extensions (...maybe except major release?). Extensions should therefore not duplicate existing names.
- Defined using the same syntactical and function-signature patterns as core BMI
- Restricted to the same data types that core BMI handles [ADD FURTHER CLARIFICATION SOMEWHERE]

What is the design philosophy behind BMI Extensions?

Some principles of BMI Extensions:

- Extension design should follow the same philosophy as BMI design generally; for example, the "Hollywood principle" (see ***ADD LINK/REF***)
- Extensions are preferred over "workarounds"
- In general, operations that are needed for only a subset of models or model types (such as geospatial capabilities or dynamic gridding) should be handled by an Extension

- Extensions should be compatible with any commonly used scientific programming language; that is, their design should not preclude implementation in any commonly used scientific programming language
- Extension developers should be mindful of naming and namespaces; for example, an Extension that handles cooking and kitchens might define function names starting with `cook_` or `get_kitchen_` [**THERE'S SOME VARIANCE OF OPINION HERE; PROJECT NAMES ARE POTENTIALLY PROBLEMATIC**]; in some cases, good practice would be to extend the names of existing functions, such as `get_var_` and `set_var_` as long as the remainder of the function name is sufficiently specialized that it is unlikely to conflict with future names.
- IS THERE VALUE TO PREPENDING SOMETHING TO **ALL** EXTENSION FUNCTIONS, EG `bmix_` so developers and readers of code can immediately tell whether a given function is core or extension? What alternative methods could there be to distinguish?
- GENERAL UNRESOLVED ISSUE OF RESTRICTIONS ON FUNCTION RETURNS, OR LACK THEREOF
- REQUIREMENTS VS RECOMMENDATIONS...

Who can design and propose a BMI Extension?

Anyone! BMI is an open community standard. The process for proposing a new Extension is described below.

How are BMI Extensions proposed, evaluated, and ultimately added to the overall BMI Standard?

Extension definitions are proposed, evaluated, and added using the same procedures for BMI as a whole, which are outlined in the BMI Governance Document.

A complete proposal for an extension should be implemented as a Pull Request. It should include a specification in at least one language. It should include at least one example, such as the heat example included in the Core BMI documentation [**ADD LINK**].

(summarize what these are)

What in Core BMI needs to be modified to accommodate the existence of extensions?

Core BMI will need a function to query what extensions and versions are supported.

How and where are Extensions listed and documented?

What are some example Extensions?

The CSDMS RSEs have proposed a pair of examples for what an extension might look like.

- Python: https://github.com/csdms/bmi-example-python/pull/31
- Fortran: https://github.com/csdms/bmi-example-fortran/pull/20

Nels Frazier has made C extensions: https://github.com/hellkite500/bmi-c-extensions

What are some guidelines for frameworks to implement support for Extensions?

- Frameworks will need some way to detect core vs. extension
- ...and a way to find out what extensions a particular model supports (as well as versions)

[**include at least one example of how csdms would do it**]

ISSUES TO CONSIDER

- Can there be multiple extensions to do the same thing? Probably not, if we want "fully compatible" - i.e., two extensions doing the same thing would not be compatible with one another
- ...