MOOSE Integration in the NEAMS Workbench

Robert A. Lefebvre,* Cody J. Permann,† Brandon R. Langley,* Bradley T. Rearden,* Richard C. Martineau†

* Oak Ridge National Laboratory
1 Bethel Valley Road, Oak Ridge, Tennessee 37831 USA
lefebvrera@ornl.gov

† Idaho National Laboratory
P.O. Box 1625, Idaho Falls, Idaho 83415
USA cody.permann@inl.gov

INTRODUCTION

The Multiphysics Object-Oriented Simulation Environment (MOOSE) is a finite-element, multiphysics framework primarily developed by Idaho National Laboratory [1]. The MOOSE framework provides a platform on which physics applications can easily be created. Recent work has been conducted to develop infrastructure to support the MOOSE framework applications within the Nuclear Energy Advanced Modeling and Simulation (NEAMS) Workbench [2]. This paper discusses the status of MOOSE applications’ integration into the NEAMS Workbench and future considerations. This task focused on demonstrating the BISON [3] nuclear fuel performance application.

INTEGRATION OVERVIEW

Integration into the NEAMS Workbench minimally requires the ability for the Workbench to (1) process the application input, (2) run the application with the input, and (3) visualize application data and results. With these three capabilities, the Workbench can effectively assist users in the use of the MOOSE framework application.

MOOSE FRAMEWORK APPLICATION INPUT

At the time of integration, the MOOSE framework used the GetPot [4] input file and command line parser. The GetPot project is an anagram of the Linux “getopt” utility, which is used to process program command line arguments. GetPot extends command line argument processing with a hierarchical input format that describes blocks, sub-blocks, and named parameters to the user via a --dump or --yaml command line option. The --dump option produces the input definition in the GetPot format with certain input information (e.g., named parameter legal value enumerations) in comment fields. The --yaml command line option produces a YAML [6] formatted input definition, or schema. The YAML-formatted input definition provides a more complete input definition for input validation, but it still lacks some important reference information which will be discussed in a later section.

GETPOT INPUT FORMAT

GetPot is licensed under GNU Lesser General Public License 2.1, which allows extensions and redistributions. As such, LibMesh has incorporated the GetPot code and has made minor improvements. The GetPot block can contain comments, nested sub-blocks, and named parameters (Fig. 1).

![Fig. 1. GetPot Block, Sub-Block, and Named Parameter syntax.](image)

The MOOSE framework provides a capable input infrastructure that allows MOOSE applications to communicate an input definition, the available blocks, sub-blocks, and named parameters to the user via a --dump or --yaml command line option. The --dump option produces the input definition in the GetPot format with certain input information (e.g., named parameter legal value enumerations) in comment fields. The --yaml command line option produces a YAML [6] formatted input definition, or schema. The YAML-formatted input definition provides a more complete input definition for input validation, but it still lacks some important reference information which will be discussed in a later section.
However, GetPot input errors were not easily interpreted. The error message did include the fault, but it failed to provide the textual location, line, and column of the fault in the input file. This is not an issue if the fault is unique enough that a simple text search reveals the textual location. However, this does limit programmatic access to the textual location, preventing a user interface from helping facilitate resolution of the issue. During this collaboration the MOOSE Framework migrated to a newly written Hierarchical Input Text (HIT) [7] Parser which addresses many of the identified shortcomings.

WORKBENCH MOOSE INPUT PROCESSOR

The NEAMS Workbench input processor [8] uses lexigraphic analysis—the Lexer—to process input “words”. The Workbench uses the words’ textual location in the input parser—the Parser—for semantic analysis. The result is a parse-tree data structure that represents all of the user-specified data, including hierarchy, regardless of correctness. Once the Lexer and Parser have produced a complete understanding of the user input, the validation engine can be invoked for a more comprehensive validation check. The Lexer and Parser are more complex to implement than the GetPot, but they provide a more robust capability that better facilitates users’ input introspection and validation.

The NEAMS Workbench extracts the textual location of blocks, sub-blocks, and named parameters from the parse-tree, and it presents quick-navigation items which use the textual locations to allow the user to quickly navigate to a section of input. The MOOSE Framework was updated to provide --json and --definition command line options which go beyond the original --yaml in producing a more comprehensive listing of a MOOSE application’s input parameters. The --definition option fulfills the NEAMS Workbench input definition interface [8] and enables Workbench input features for MOOSE Framework applications.

WORKBENCH MOOSE INPUT CREATION

At its most basic level, the Workbench is a text editor that makes input creation natural. However, the Workbench can jump-start the user with added benefits such as autocompletion of input and input introspection. Input autocompletion uses a template construct in which the template contains placeholders for data of interest. For MOOSE input, there is a minimum of three template constructs: block, the sub-block, and the named parameter. These three templates could be presented as examples with text to be replaced by the user, minimizing the user’s required input formatting or required syntax specification. Fig. 2 depicts the simplest MOOSE block input template, which requires the user to replace both parameter and value with the appropriate name and value of the component.

```
[block]
[
]
```

Fig. 2. Simple MOOSE input block template.

```
[./sub_block]
[../]
```

Fig. 3. Simple MOOSE input sub-block template.

```
parameter = value
```

Fig. 4. Simple MOOSE input named parameter template.

The templates shown above aid the uninitiated user in legal syntax of the input, but they do little more. The Workbench combines the input definition, in conjunction with the parse-tree and user’s input cursor, to perform input autocompletion and insert the input field and default value, where available. The component’s input definition communicates all possible input to the Workbench that is available to insert into the input file at the user’s cursor location.

Fig. 5 depicts a MOOSE BISON input with an auto-complete list presented to the user illustrating all available components at line 314. The auto-complete list incorporates the component’s name and description, and Workbench uses the input definition for available components in conjunction with the parse-tree to filter components as appropriate. For example, components that have already been specified, fulfilling the components maximum occurrence validation rule, are not available for additional insertion, as it violates the maximum occurrence validation rule.

```
```

Fig. 5. Workbench BISON input auto-completion.
Significant capabilities have been initiated to support MOOSE application input, but there are still areas to be improved, as well as user feedback and significant input features to be captured.

**WORKBENCH APPLICATION EXECUTION**

A mission of the Workbench is to facilitate transition from conventional tools to high-fidelity tools. Many of these codes involve different means of invocation, and some have multiple means of invocation. To facilitate execution of these applications, a generic runtime environment interface was created. The intent of the runtime is to provide a consistent interface by which the Workbench can interact with each application in all necessary modes of operation (e.g., serial, parallel, and scheduled execution). The runtime is designed to provide a clean, convenient, and consistent means for users to invoke NEAMS toolkit applications via the command line, separate from a graphical user interface that is Workbench.

**MOOSE BISON RUNTIME ENVIRONMENT**

The MOOSE BISON application, bison-opt, requires its invocation to occur in the directory in which the input resides. Fig. 6 depicts a typical invocation of the MOOSE BISON application.

```
/software/neams/moose/bison/bison-opt -i problem.inp
```

Fig. 6. MOOSE BISON-OPT example invocation.

The bison.py runtime environment encapsulates the bison-opt command line options, the job setup, execution, and finalize logic with ~200 lines of Python. Most of these lines are propagating the command line options through the Workbench’s BISON runtime environment for the Workbench to effectively communicate these to the user. Figure 8 depicts an example invocation of the BISON runtime environment. Note the complete path to the problem.inp input file.

```
/software/neams/rte/bison.py -i /path/to/input/dir/problem.inp
```

Fig. 7. MOOSE BISON runtime environment example invocation.

The Workbench can use this interface to allow the user’s invocation of multiple MOOSE application inputs in the same interface, as depicted in Fig. 8.

**VISUALIZATION INTEGRATION STATUS**

The Workbench has the VisIt [9] visualization tool integrated. VisIt supports mesh data visualization including the Exodus file format used by the MOOSE framework for input and results mesh data. Figure 10 depicts visualization of the BISON application input, execution, and input and data mesh VisIt visualization within the NEAMS Workbench.

The Kitware ParaView [10] visualization toolkit is under consideration as an alternative visualization capability.

**SUMMARY**

The MOOSE integration, and demonstration using the BISON fuel performance application, helps streamline the users’ interaction with their input, its execution and visualization, and subsequent results visualization. The MOOSE framework architecture provides a succinct
location from which the NEAMS Workbench can integrate MOOSE applications. While the BISON application was demonstrated, in theory, all MOOSE applications are supported. This initial integration is available in the NEAMS Workbench 1.0 beta release [11].

MOOSE used the LibMesh extension of the GetPot input processor. This input processor lacked the textual location information needed by Workbench to facilitate complex on-demand input validation. The Workbench input processor fulfills these needs and the MOOSE team has since migrated to a new parser that provides better textual location information. Continued collaboration will begin to consolidate these capabilities and will ensure that the user experience remains as consistent as possible.

The Workbench runtime environment greatly facilitates the Workbench in its interactions with MOOSE applications. The current interface has been shown to facilitate local execution of the MOOSE BISON application. Remote high-performance clusters with potential scheduler/queuing interfaces must be designed and integrated to best facilitate large or long-running jobs. This capability is under development in the NEAMS Workbench.

ACKNOWLEDGEMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Energy, under the NEAMS-IPL program under contract number DE-AC05-00OR22725.

REFERENCES