ORNL/TM-2021/2141

NEAMS Workbench MOOSE Integration Update

Robert A. Lefebvre
Brandon R. Langley
L. Paul Miller
Mark L. Baird
Benjamin S. Collins

August 27, 2021

Approved for public release.

OAK RIDGE National Laboratory

ORNL IS MANAGED BY UT-BATTELLE LLC FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2021/2141

Nuclear Energy and Fuel Cycle Division

NEAMS WORKBENCH MOOSE INTEGRATION UPDATE

Robert A. Lefebvre
Brandon R. Langley
L. Paul Miller
Mark L. Baird
Benjamin S. Collins

August 27, 2021

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831-6283
managed by
UT-BATTELLE, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

CSG	constructive solid geometry
GUI	graphic user interface
LSP	Language Server Protocol
MOOSE	Multiphysics Object-Oriented Simulation Environment
NEAMS	Nuclear Energy Advanced Modeling and Simulation
RTE	run time environment
WASP	Workbench analysis sequence processor

ABSTRACT

The Nuclear Energy Advanced Modeling and Simulation (NEAMS) Workbench is a graphical user interface (GUI) that provides a common analysis environment for the accelerated use of the NEAMS toolkit. To improve design and analysis of current and future nuclear energy systems, the NEAMS Workbench provides an integrated development environment for model creation, review, execution, output review, and visualization for integrated tools. In addition to a GUI, the NEAMS Workbench provides the Workbench Analysis Sequence Processor, an open-source tool set that facilitates input-content assistance for supported domain-specific input languages and user-friendly syntaxes. These supported syntaxes enable accelerated development and deployment of enhanced user inputs and workflows. The Multiphysics Object-Oriented Simulation Environment (MOOSE) has been supported in the NEAMS Workbench since 2016 and receives continuous updates. Efforts to improve MOOSE-based tool integration in the NEAMS Workbench are ongoing. This document details the current and planned enhancements of the NEAMS Workbench and the MOOSE framework to improve integration and usability of MOOSE-based applications within the NEAMS Workbench.

1. INTRODUCTION

The Nuclear Energy Advanced Modeling and Simulation (NEAMS) Workbench initiative started in 2016 to provide a common analysis environment for the accelerated use of the NEAMS toolkit [1]. By providing an integrated development environment that facilitates the model creation, review, execution, output review, and visualization for integrated tools, the NEAMS Workbench consolidates code interactions and can shorten model development iterations. The Workbench is composed of three software components: (1) the desktop application or graphical user interface (GUI), (2) the application run time environment (RTE), and (3) the analysis sequence processor. The GUI provides an advanced text editor, local and remote job launch capability, constructive solid geometry (CSG) visualization, 2D data plotting, and an integrated mesh visualization and data analysis toolkit (VisIt or ParaView). The advanced text editor provides input content assistance for accelerated model creation, editing, navigation, and verification. The CSG model visualization enables visual verification of geometry prior to job launch for integrated codes using CSG modeling methods. RTE is a Python scripting interface that normalizes the Workbench interaction with local and remote job launches. Additionally, the RTE provides an extensible workflow environment for specific analysis needs and serves to provide reproducible job execution and an extensible application integration point. Remote job launch enables users to remain on a familiar operating system while leveraging remote computing capabilities. The Workbench analysis sequence processor (WASP) provides an open-source, domain-specific language processor toolkit from which Workbench-integrated code input is lexed, parsed, interpreted, and validated. In addition to language processors, WASP provides tools to assist with input generation and analysis workflows, as needed, for application RTEs and the GUI [2].

Since the inception of the NEAMS Workbench, the vision has been to support multiphysics workflows as provided by the Multiphysics Object-Oriented Simulation Environment (MOOSE) and as needed by the user community. Per user feedback, a focus on integrating applications and existing workflows has been prioritized, with demonstrated successes in neutronics [3,4], thermal hydraulics [5], fuel analysis and performance [6], and associated model uncertainty quantification and optimization [7, 8].

In FY21, the primary goal was to complete integration of MOOSE-based applications into the NEAMS Workbench. Specifically, work was undertaken to (1) ensure that the content assistance provided by the Workbench was consistent with the MOOSE input processor, (2) provide problem mesh visualization and face/block highlighting for select regions, (3) provide solution visualization, (4) ensure support for MOOSE MultiApp inputs, (5) allow mesh overlay of coupled calculations, and (6) include key result

extraction and plotting. The remainder of this document will describe the Workbench components involved, work accomplished to date, and future work.

1.1 WORKBENCH ANALYSIS SEQUENCE PROCESSOR (WASP)

The WASP provides open-source C++ lexers, parsers, and interpreters for language syntaxes that facilitate the NEAMS code's integration into the Workbench. The languages are generic and are incorporated to facilitate the reuse and development of new codes and workflows. The language processors typically produce parse-tree data structures. In addition to the language processors, WASP contains a hierarchical input validation engine and template engine. The validation engine applies a language input schema that describes the structure and component restrictions to validate that a given input conforms to the application requirements. Any input components that do not adhere to the input schema are flagged with a descriptive error message highlighting the location and cause of the problem. The hierarchical template engine facilitates workflows that require input generation and input component autocompletion as delivered by the NEAMS Workbench's content assist capabilities. Lastly, WASP includes command line programs for listing, selecting, converting, and validating supported syntaxes. These aid in integration development, general input tasks, and related workflows.

WASP includes features that help domain researchers quickly design and process application user input and provides a reusable component that reduces research software engineering costs. Furthermore, the user-approachable input syntaxes and NEAMS Workbench content-assist features accelerate deployment and user-adoption. The development of the Economics Dispatch Genetic AlgoRithms code, known as EDGAR, is an example of this researcher-friendly workflow [8].

The formalization of integrated development environments has manifested in Microsoft's Language Server Protocol (LSP) [9]. The LSP provides a language server to communicate input diagnostics and content assistance across program process boundaries. The LSP will reduce maintenance overhead, improve interface stability, and enable the native MOOSE input framework's diagnostics and data structures to be communicated to Workbench. This protocol has been identified as the best path forward for sustainable integration of the MOOSE framework into the NEAMS Workbench, and future integration efforts will pursue this development path.

1.2 LANGUAGE SERVER PROTOCOL (LSP)

The Microsoft LSP provides language-specific content assistance between a single language server and what could be multiple input/editor/development tools. By enabling inter-process communication via a JavaScript Object Notation (JSON) remote procedure call [10], the language protocol enables a one-to-many coupling strategy that is economically attractive for supporting domain-specific languages. An LSP feature overview is outside the scope of this document, but the features are full and matured enough to be incorporated into widely adopted integrated development environments such as Visual Studio Code, the Eclipse integrated development environment, EMACS, and Vim [11]. Because the NEAMS toolkit is under active development, the ability to harness the software development environments used for tool development is an attractive benefit of natively supporting the protocol in the MOOSE framework. Consolidating developer-software and software-input interactions will enable developers to use the same input interaction mechanisms as end-users, thus improving user experience and shortening user-feedback cycles.

The integration of LSP into the NEAMS Workbench has already facilitated support for the MCNP6 code [12]. Per developer iterations between the MOOSE and Workbench teams, the use of a MOOSE-embedded language server is the best direction for current and subsequent integration investments. This

move will avoid development of duplicate software and will ensure that the MOOSE input features and diagnostics are most accurately served to the user.

1.3 RUN TIME ENVIRONMENT (RTE)

The Workbench's RTEs are interface scripts for normalizing the interaction between the Workbench and the individual tool or application. The most basic functionality is the communication and configuration of command line arguments. This enables a tool to have multiple run configurations within Workbench. The RTE enables configuration of remote machines or execution of jobs on the user's local machine. The remote machines require an installation of Workbench and the application but can be run in a scheduled (e.g., PBS, SLURM, IBM LSF) or unscheduled remote environment. These run configurations can be edited as needed or saved as ready-to-select local or remote job-launch configurations.

2. WORKBENCH AND MOOSE APPLICATION INTEGRATION

Development and testing of the MOOSE framework and the NEAMS Workbench has required considerable effort and resources. With both projects under active development, the integration testing has required special attention to prevent feature regression. Additionally, software licensing and export control restrictions have further complicated or delayed integration efforts. Workbench integration testing involves generating the MOOSE application's input schema and subsequently using the generated schema to validate application inputs. Results convey problem areas that need improvement. The following subsections detail the process and improvements implemented during FY21.

2.1 TESTING

Beyond running the typical continuous integration test suites for Workbench, an additional level of application integration testing is regularly performed to identify and prevent regressions in application integration. The primary component of integration is the user input. For each supported application, the application is compiled, and its input schema is generated. This step ensures that the development environment configuration and schema generation are functional. For MOOSE applications, the *--definition* command line argument generates the WASP-formatted schema data. Subsequently, each functional application test input (*functional* because it is expected to not fail) is run through the respective WASP validation program. This process checks that the syntax and semantics are still supported. To date, regressions in integration have subsequently prevented integration within the NEAMS Workbench from reaching 100% input accuracy. For example, a legal MOOSE input component is nonetheless thought to be invalid by Workbench. Most regressions are quickly addressed, but deficiencies remain in the NEAMS Workbench's input accuracy. These problems have led development teams to pursue consolidating and improving input processing and diagnostics by using a language server embedded in the MOOSE framework and inherited by all MOOSE applications.

Some details of MOOSE's input schema generation are unavailable, which has also made the omission of specific validation restrictions necessary. The validation rule that ensures referential components are defined in their respective input locations is one example. For instance, all variables must be defined in a *Variables* or *AuxVariables* block, and all functions must be defined in a *Functions* block. The MOOSE framework does provide information that the schema generator uses to build relative lookup paths for any input context containing a referential component. Workbench could then validate that all references have been properly defined elsewhere in the input. However, it was discovered through integration testing across various MOOSE applications that the MOOSE framework does not verify that a piece of input is defined in its respective definition block until that component *is used in the calculation*. This was then confirmed by the MOOSE development team.

In most cases, this validation rule is beneficial because it warns users about any undefined input or typos that can cause name mismatches. However, the integration testing revealed many examples of legal input with components that were undefined because they were never actually used in the calculation. Workbench's input validation engine cannot identify which pieces of input are used and which are not used by the calculation without building the calculation itself. In this case, the MOOSE framework is necessary for determining a class of input validation using input introspection.

For this specific situation, it was decided that false positive reports would be more disruptive to a user's workflow than unreported invalid cases. Therefore, this entire validation rule was removed from the schema generation for all MOOSE applications. However, a new schema rule was generated to apply these lookup paths to Workbench's autocompletion recommendation lists without using them for input validation. The direct integration of the language server into the MOOSE framework will remedy these validation problems as MOOSE's native input processor will be used for introspection to decide if something is illegal.

2.2 IMPROVEMENTS

In FY21, the MOOSE framework was improved through continuous integration with WASP. This process proactively highlights regression in WASP's support of MOOSE syntax and semantics.

Workbench performs integration testing on the input processing for each MOOSE application in multiple, successive steps. First, Workbench constructs a list that contains all application input files to be tested. Every MOOSE application has a test harness that can be executed with various command line options to perform a suite of regression tests. Most of these tests execute the application on a given input file, and then they check the output for expected results. Workbench's integration testing system uses the test harness of each application as a starting point to gather its list of testable input files.

The *--dry-run* and *--verbose* command line options are provided to the MOOSE application test harness which outputs the path of each directory and input file that would be used for every regression test without actually executing any of the tests. This output is generated as a list of absolute paths to every input file tested by the application. Notably, some input files will contain expected input errors because they are being tested for failures by the application's test harness. These input files must be filtered from Workbench's integration test list because they are not expected to be valid. This filtering is completed by executing MOOSE's *--check-input* option on each input file and only keeping those that report a valid syntax.

The tested MOOSE application is then executed with the *--definition* command line option to generate its WASP-formatted input schema. Next, a WASP utility attempts to parse every input file in the previously constructed list. Any input that does not completely parse without a syntax error fails the integration test. All inputs that successfully parse without error are then validated by another WASP utility that applies the previously generated application input schema. Any input that produces validation errors fails the integration test. Therefore, for an input to pass, its syntax must be successfully parsed, and its data must be regarded as valid by the rules contained within its application input schema.

This style of test iteration lends itself to easy classification of errors and prioritization of fixes for the tested MOOSE applications. All input files that fail the integration test are automatically grouped by their reason for failure. These failure causes usually point directly to the issue that must be addressed. For example, parsing failures must generally be addressed in the WASP parser package, whereas validation failures must be addressed in the MOOSE schema generation code or deeper in the MOOSE input framework. In each case, a ticket is opened to describe the issue, impact, and frequency so that the problem can be fixed quickly. This automated integration testing and subsequent fixes to the framework

have resulted in significant improvements to the Workbench's input processing of MOOSE applications during FY21. A summary of these improvement metrics is presented in Table 1.

Table 1. Summary of improvement metrics

	October 2020			August 2021		
	Inputs passing	**Inputs failing**	**Success rate**	**Inputs passing**	**Inputs failing**	**Success rate***
MOOSE	738	809	47.71%	1,621	53	96.83%
BISON	46	1,096	4.03%	1,322	17	98.73%
SAM	288	43	87.01%	331	62	84.22%
Total	1,072	1,948	**35.50%**	3,274	132	**96.12%**

(*) Code changes enabling August 2021 success rates may not yet be merged into MOOSE or Workbench.

The low success rates for MOOSE and BISON in October 2020 were caused by a change in the JSON library used in the MOOSE framework, in which a slight change in string behavior exposed a deficiency in integration testing. The 132 (or 3.88%) failing inputs were caused by 5 kinds of false validation errors (Table 2). These validations errors were primarily caused by inadequacies in the Workbench processors or missing data in MOOSE's input schema.

Table 2. Validation error types

Type	File count	Failure percent of 132
Invalid input	104	80%
Type failure	72	55%
Wrong value type	15	11%
Unexpected occurrences	10	8%

An *Invalid Input* is defined as an input field that exists in the input but is not listed in the input schema. This occurs in 80% of the failed inputs. *Type Failures* constitute 55% of failures, and *Wrong Value Types* constitute 11% of failures. These failure types are caused primarily by Workbench's current inability to incorporate variable references. As a result, the variable name (a string), not its value, is type checked. Examples of field type checks are minimum or maximum value validation. As the name suggests, *Unexpected Occurrence* failures indicate that the expected occurrence of an input field is not expected. All outstanding integration issues are an argument for consolidating MOOSE input processing under an LSP.

2.3 WORKBENCH MOOSE APPLICATION FEATURES

Although all desired features are not yet implemented, this section highlights the features available as of this writing.

2.3.1 Application Configuration and Local and Remote Job Launch

Workbench supports launching jobs on the user's local machine or on a remote computing resource as scheduled or unscheduled jobs. The user can configure an application to enable convenient execution of application features and have these configurations simultaneously available in the same user session. Workbench enables configurations for small test executions on the user's local machine, larger jobs only requiring a more powerful remote desktop workstation, or the largest jobs involving scheduled compute resources and associated scheduler parameters.

In the case of MOOSE applications, users can launch the application with a select input or simply have the application generate the associated mesh file once the mesh generation parameters have been edited. Figure 1 illustrates a default configuration, *Bison_wr9 1*, and a custom configuration, *Bison_wr9-mesh-only*). In the *Bison_wr9-mesh-only* configuration, the MOOSE application will generate the exodus mesh file, and Workbench will present the file in the *Associated Files* context menu.

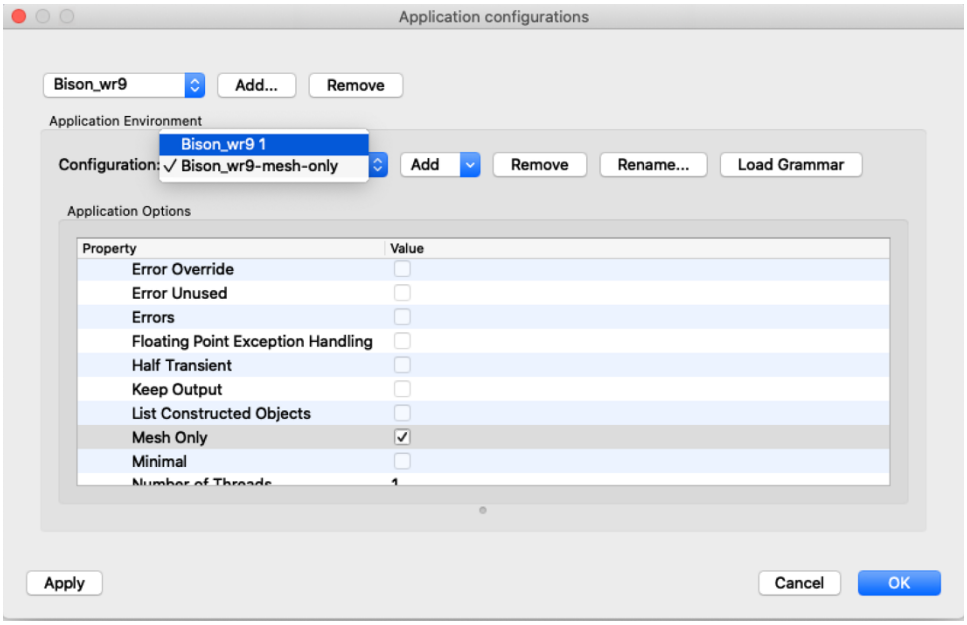

Figure 1. Multiple MOOSE application remote run configurations.

Because active development produces frequent changes to MOOSE applications, including changes to input, Workbench enables users to easily reload the application's "Grammar." The grammar is composed of the input schema, syntax highlighting rules, and input templates. For example, if a developer or user updates the MOOSE application with a new AuxKernel, then clicking the Load Grammar button will query the MOOSE application and obtain the new input field's definition.

2.3.2 Input Content Assistance

The Workbench content assistance incorporates syntax highlighting for improved visual recognition of input and features to assist users in navigating, creating, and editing inputs. Allowing input format and documentation editing are intrinsic features of Workbench, but traditional GUIs often do not allow access to the actual ASCII input and instead present data in widgets. These widgets require additional software maintenance and specific logic to handle documentation styles. For example, does a user comment exist above, after, or below an input field? Workbench focuses features on accelerating a user's interaction with the application's native input, not a widget library prone to subjective user opinion which can create distracting discussion involving what is easiest widget-form to use. This strategy has the added benefit of increasing user-focus on the native input format and its ease-of-use, helping better prioritize user-facing developer activity.

2.3.2.1 Syntax Highlighting

Improved visual presentation of input facilitates more input recognition and less memory recall, thereby reducing navigation times and improving user comprehension [13]. As shown in Figure 2, syntax

highlighting of MOOSE input improves recognition of syntax constructs, parameter bocks, and associated labels and values.

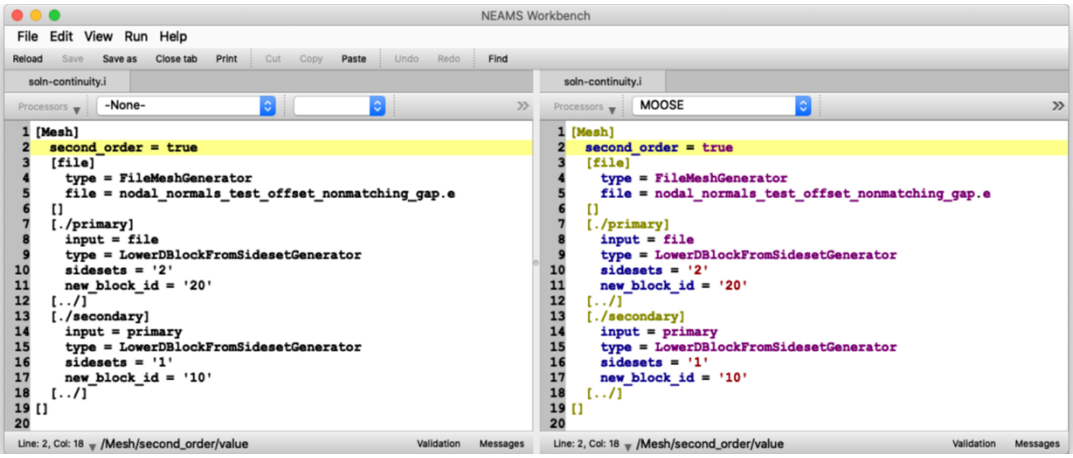

Figure 2. MOOSE input syntax highlighting.

As shown in Figure 3, the user can change the default syntax colors and styles for a given application using Workbench's grammar settings. To accommodate a user's input extension preference, the extension for which one wishes to apply the grammar settings can be customized to include different or additional file extensions.

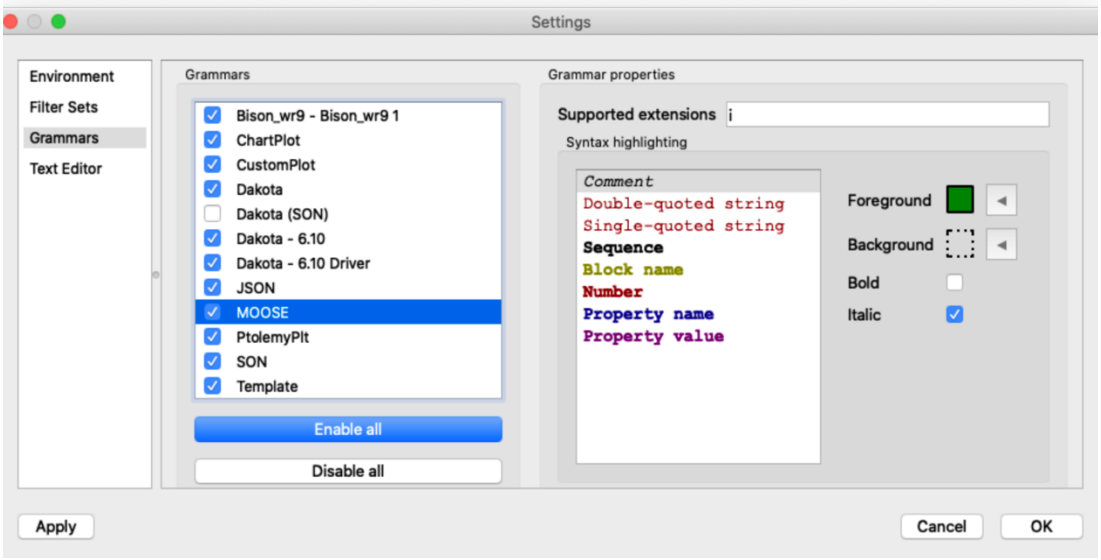

Figure 3. MOOSE syntax color settings.

2.3.2.2 Autocompletion

Because the Workbench processes the application input schema and input syntax, it recognizes the context in which the user's cursor resides. This capability enables Workbench to present a list of available input edit actions to the user (Figure 4). These actions allow a user to preview available components with documentation snippets and to quickly recognize referential data entries (Figure 5), thus accelerating input creation and editing. Additionally, Workbench identifies which input fields are referential, making it easier for the user to quickly navigate to the definition of the reference.

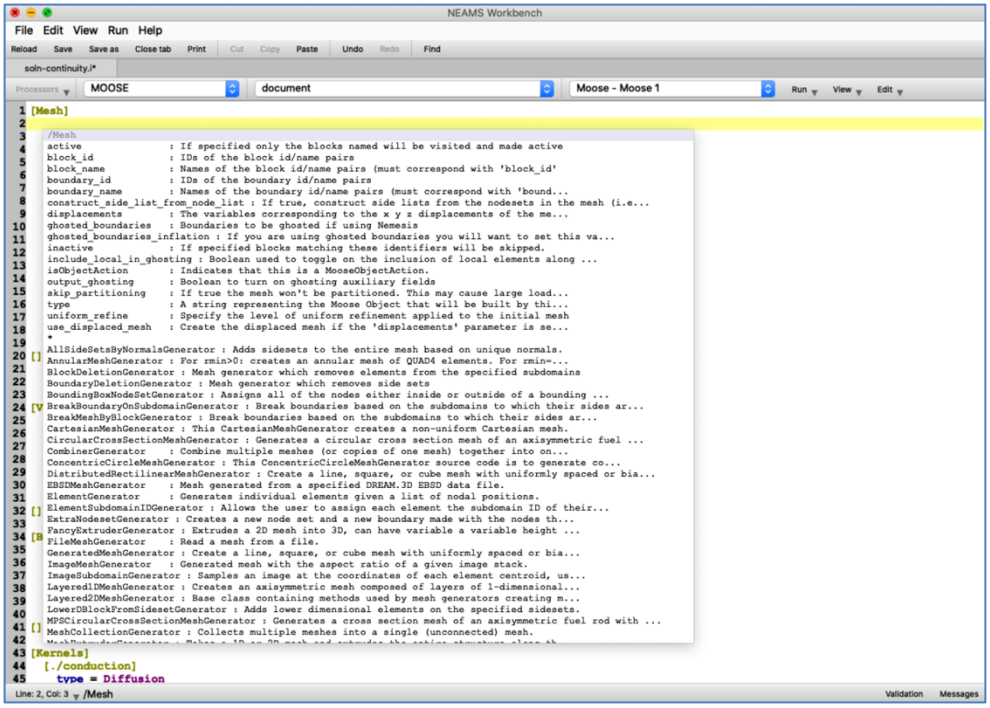

Figure 4. MOOSE application mesh component autocomplete listing with documentation snippets.

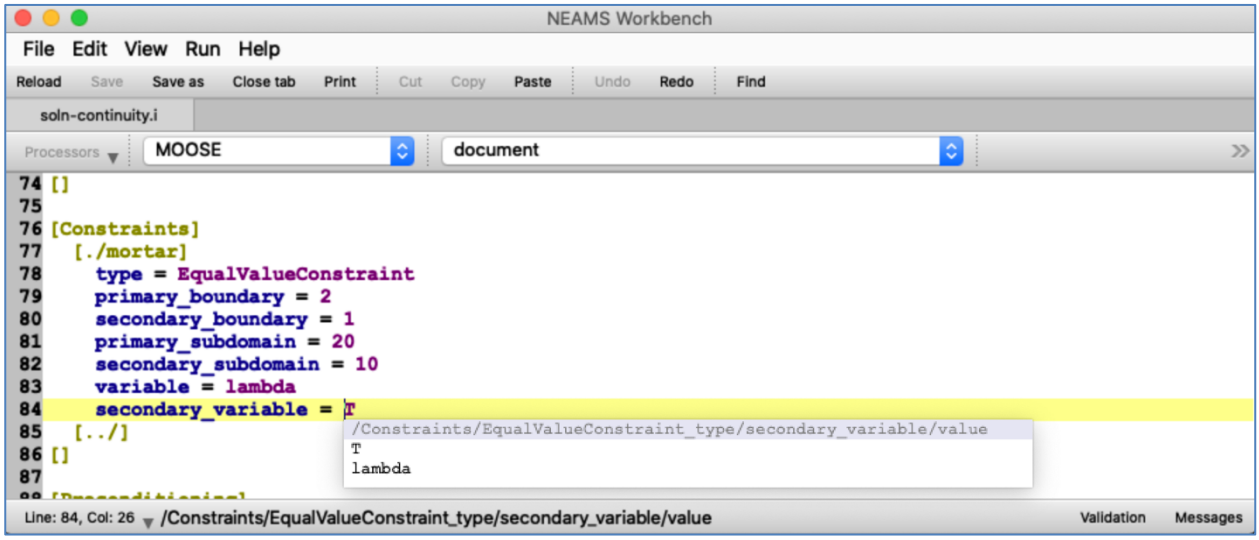

Figure 5. MOOSE application referential input autocomplete listing.

2.3.2.3 Automatic Input Checking

Upon processing an input file, the Workbench applies the application schema to the generated parse tree and produces the list of problems to the user in the editor's *Validation* panel (Figure 6). As the user edits the input, the document is rechecked, and the *Validation* panel's listing is updated.

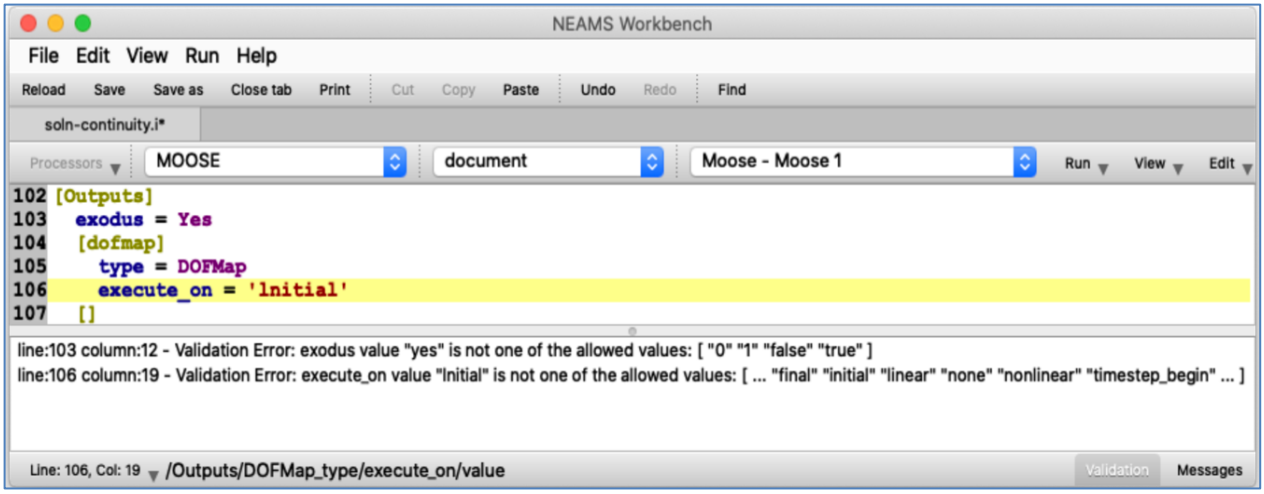

Figure 6. Interactive input validation panel.

Clicking an entry in the *Validation* panel's list will place the user's cursor at the indicated location, enabling the user to quickly examine the problem. The Workbench provides general error messages that effectively communicate many of the problems in an import, but a deficiency is that these messages are not native MOOSE error messages. I.e., the same input error can be described differently when identified by Workbench and MOOSE. The MOOSE framework native diagnostic messages will be accessible to Workbench when the integration of the LSP into the MOOSE framework is complete.

2.3.2.4 Navigation

The Workbench presents the document hierarchy in a navigation panel and an editor-specific document dropdown menu. When the user clicks an entry in the navigation panel or document dropdown, a cursor is placed on the input component. This shows the full input context so that necessary changes to the input can be made more quickly (Figure 7).

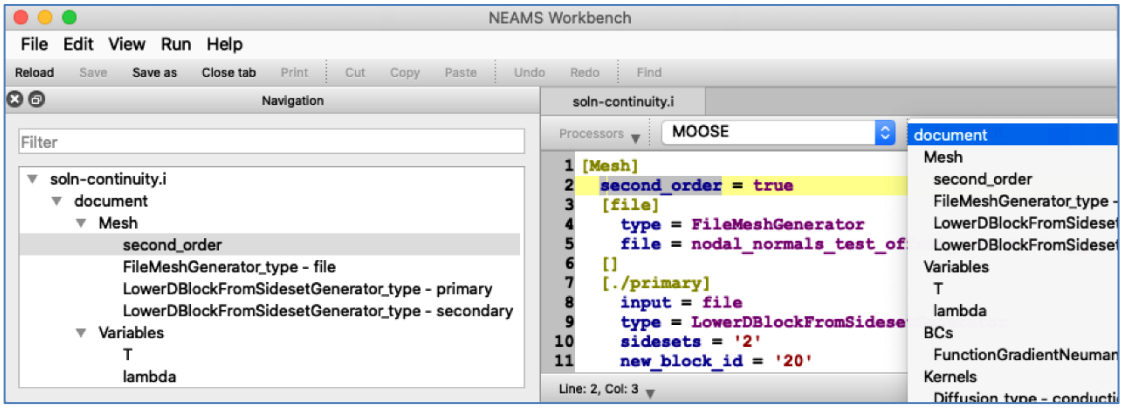

Figure 7. Document navigation panel and editor drop down items.

A right-click on a reference will present a context menu with the option to *Goto definition…* (Figure 8). This takes the user directly to the location where the referenced input is defined, avoiding a time-consuming search for the identifier. Additionally, navigation to the definition avoids possible distractions and confusion caused by ambiguous identifiers in the input document.

Beyond intra-input navigation, the Workbench also provides access to associated files via the *Navigation* panel's *Open associated files* context menu and selection-based file open shortcut (Figure 9). File drag-and-drop into Workbench is also supported.

Figure 8. Select and open file paths in document and output.

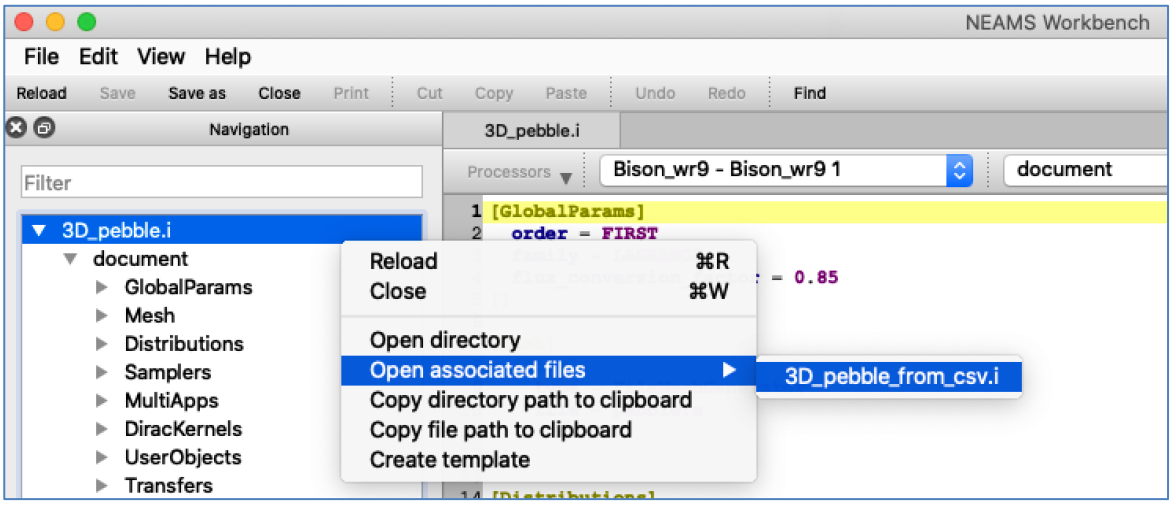

Figure 9. Navigation panel's associated files context item.

Improved Inter-document navigation, especially for MOOSE MultiApp inputs, is expected with the LSP implementation.

2.3.2.5 Edit Accelerators

Because engineers and the Workbench interact with different input formats, having bulk text operations during model development iterations can be useful. Workbench tracks the comment delimiters and provides application-specific comment creation. Additionally, using the **CTRL** + / key sequence will comment or uncomment content based on the lines selected. This enables users to quickly test the presence or absence of input fields.

As with all integrated development environments, **TAB** indents current or selected lines, and **SHIFT** + **TAB** unindents current or selected lines. This enables quick formatting of document blocks for improved arrangement. In addition to the block-wise commenting and indentation, the Workbench supports column text editing using the **ALT** + **MOUSE DRAG** combination to select columns of text.

The Workbench also allows users to embed data into a mathematical formula and evaluate it to the form required by the application. This is common when converting units from engineering to application specifications. This feature can improve model quality by encouraging the user to capture the data conversion formula in a comment, copy and paste the formula into the value field, and evaluate it to the form that the application is requesting.

Parameter sensitivity and uncertainty quantification studies are a best practice in modeling and simulation. The MOOSE framework provides stochastic tools for MOOSE applications; as described above, these tools are available through the application's input fields in the Workbench. Prior work in the NEAMS campaign integrated the DAKOTA toolkit into the NEAMS Workbench [14]. The DAKOTA toolkit, combined with the Workbench's template-editing capability, provides a tool set that enables engineers to conduct uncertainty quantification and optimization tasks. The Workbench's template capabilities enable a user to turn a nominal application input into a template using the *Navigation* panel's *Create Template* context menu item (Figure 10).

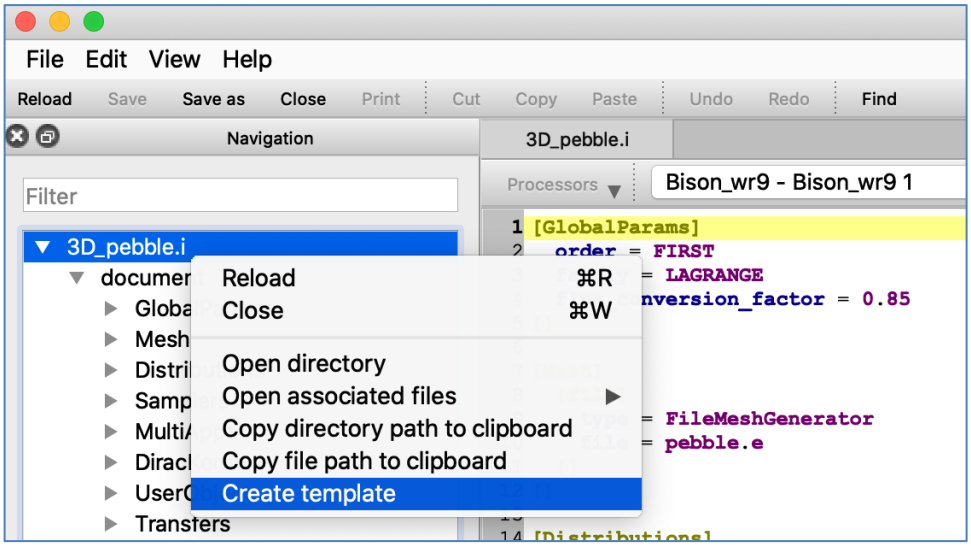

Figure 10. Create template from nominal input document.

Within a template file, any field can be converted to a parameter study attribute by selecting the field and clicking the *Template* context menu item or using the shortcut key sequence **CTRL + T** or **COMMAND + T** (macOS). The nominal text will be replaced with attribute delimiters and a selected placeholder *NAME* (Figure 11). Because the placeholder's name is selected, the user can immediately begin typing the desired attribute name or parameter study expression. The template syntax is supported by the Workbench's open-source Hierarchical Input Template Expansion engine (HALITE), which is a utility within WASP.

```
228  [fueled_region_thermal]
229    type = GraphiteMatrixThermal
230    block = fuel
231    unirradiated_type = A3_27_1800
232    packing_fraction = <NAME>
233    temperature = temperature
234  []
```

Figure 11. Templated placeholder with preselected name ready for editing.

2.4 DATA AND MESH INTERACTION

The NEAMS Workbench supports 2D data plotting and general mesh visualization. General mesh visualization and analysis are enabled through the embedded ParaView data analysis and visualization application [15]. MOOSE's comma-separated value output can be opened in Workbench and processed into interactive 2D plots. These 2D plots are highly configurable and can be saved to image, PDF, and SPF formats. As a native Workbench ASCII file format, SPF preserves the perspective and data in such a

way that shared files still contain the data's full context and remain interactive. For example, when a data feature is zoomed and an SPF is generated, the reopened SPF will present the saved perspective, but the user can still pan, zoom, or edit other attributes of the data presentation.

ParaView's mesh capabilities are outside the scope of this document, but those specifically relevant to the NEAMS Workbench and MOOSE's ExodusII file are highlighted. The NEAMS Workbench was updated to support quickly inspecting the blocks, side sets, and node sets contained in the MOOSE mesh files. Once the user has opened the mesh file, selecting an identifier and choosing *Inspect* from the associated context menu will instruct Workbench to deselect all mesh components except for the one to be inspected (Figure 12).

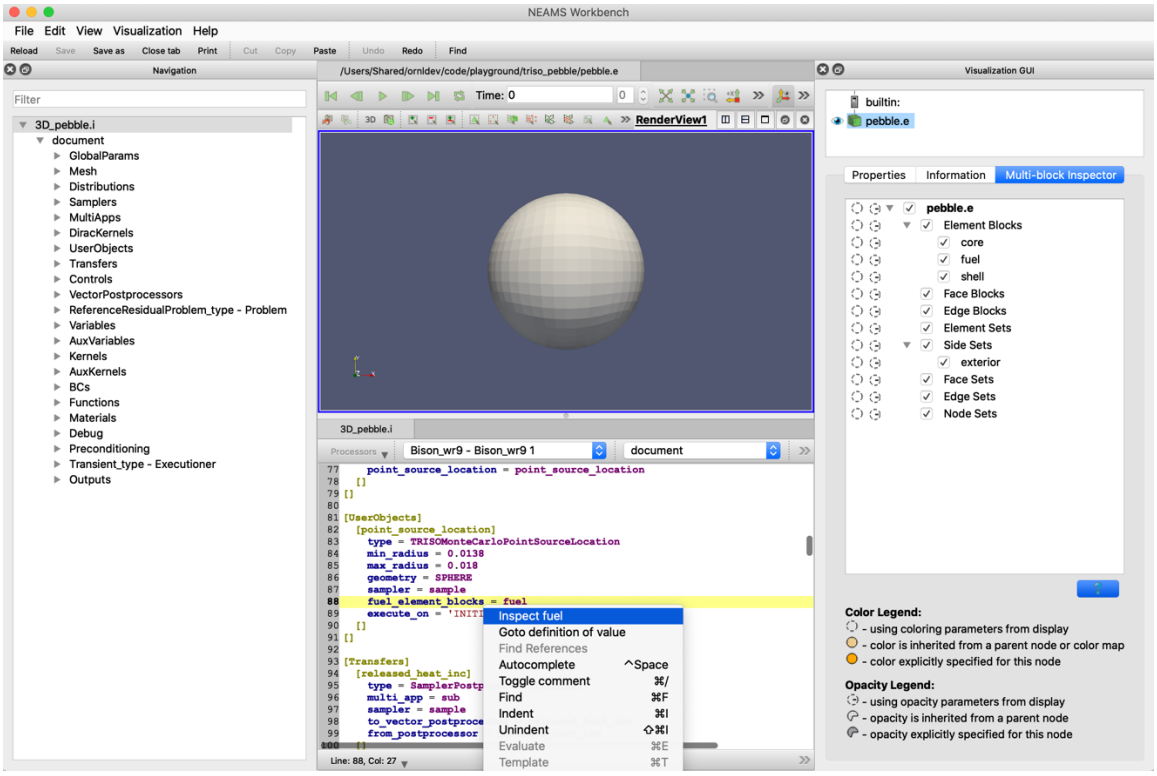

Figure 12. Input originating the mesh inspection.

Because ExodusII mesh files are considered advanced, interacting with the datasets in ParaView is not immediately intuitive and requires configuration before Workbench can inspect these data. There are two important toggles that must be checked. First, the user should ensure that the advanced properties are toggled on and set using the gear icon in the *Visualization* GUI's *Properties* tab (Figure 13).

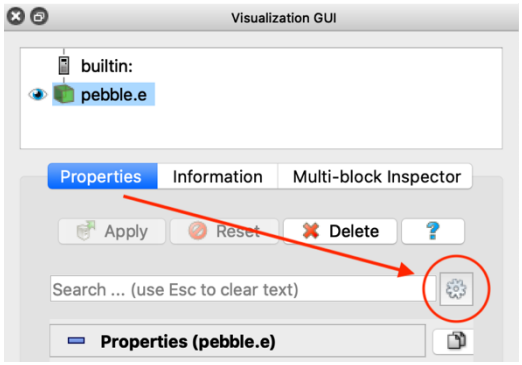

Figure 13. Toggle advanced *Properties* view, required to interact with Exodus sets.

The second important toggle is the *Sets* listing, which becomes visible after the advanced properties are toggled on. If *Sets* are present, then checking the desired box and clicking on the *Apply* button will ensure that the desired *Sets* can be toggled on when inspection is requested via interactions in the input document (Figure 14).

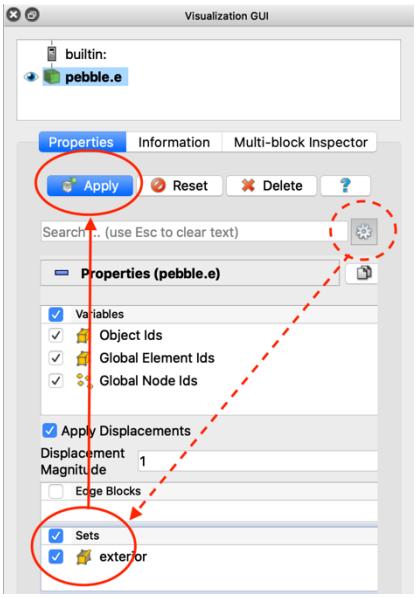

Figure 14. Set activation and application required for inspection from input file.

Improvements to the NEAMS Workbench ParaView interface will continue as user feedback and requests are received and familiarity with the ParaView codebase increases.

3. MAINTENANCE

As with all software, maintenance is an ongoing task that enables current and future development. The NEAMS Workbench had two notable maintenance activities in FY21: (1) the migration of RTE from Python 2 to Python 3 and (2) the GUI framework upgrade from Qt4 to Qt5.

3.1 PYTHON 3

The Workbench RTE uses an offline Miniconda environment to facilitate installation in non-networked locations and allow extensions for proprietary scenarios. The RTE was originally written in Python 2 based on the availability of packages. With Python 2's end of life in January 2020, migration to Python 3 was long overdue, and the Workbench team decided to move the Miniconda environment to Python 3 in FY21. Unfortunately, testing identified gaps in the code coverage for remote job launch capabilities. Therefore, the Python 3 migration has not yet been completed. Although the supported version of Python is not yet being used, the offline Miniconda Python environment insulates the user and Workbench from such changes. Only developers and tool integrators are required to interact with Python 2 code. Python 2 scripts can be written to be compatible with Python 3 to facilitate migration.

3.2 Qt5

The NEAMS Workbench uses the Qt GUI widget toolkit, which provides cross-platform widgets to reduce platform-specific development efforts. Although Qt5 was a significant update from the Qt4 toolkit, the NEAMS Workbench effort required minimal code changes. However, aa sizable effort was required to update the NEAMS Workbench software bundle, fixup, and deployment logic. Additionally, minor changes in the overall Qt widget look and feel have necessitated an ongoing review, with few updates required. With the migration from Qt4 to Qt5, the integrated visualization tool also changed from VisIt to ParaView.

4. CONCLUSIONS

In FY21, the NEAMS Workbench was updated with improved MOOSE application capabilities. Development iterations between the MOOSE and Workbench teams highlighted an improved long-term vision that better facilitates integration and usability. The NEAMS Workbench's MOOSE input interpreter was updated to improve accuracy and the user experience. An overall improvement from 35.50% to 96.12% was achieved in the parsing and validation accuracy of MOOSE input files. The ParaView visualization and data analysis application's integration was also finalized. The Workbench MOOSE user experience improvements include the ability to open MultiApp referenced files and quickly inspect MOOSE mesh components. Lastly, the NEAMS Workbench team conducted two notable software maintenance activities: (1) they commenced the RTE from Python 2 to Python 3, and (2) they upgraded the graphical widget framework from Qt4 to Qt5. Future work will continue to improve integration through the incorporation of the LSP into the MOOSE framework and through additional GUI adaptations to suit MOOSE user needs.

5. ACKNOWLEDGMENTS

This research was sponsored by the DOE NEAMS program. Special thanks to the collaborators who made contributions or provided direction that facilitated improvements. Specifically, thanks to Cody Permann and Brian Alger for their guidance and assistance with integration of the MOOSE framework applications, Andrew Slaughter and Daniel Schwen for language server discussion, and Jason Miller for integration of WASP into the MOOSE framework test suite. Thanks to Bob O'Bara, T. J. Corona, Ben Boeckel, and Utkarsh Ayachit for collaboration on, and contributions to, the ParaView visualization toolkit integration within Workbench. Additional thanks to Kaylee Cunningham for valuable testing and feedback for MOOSE BISON.

6. REFERENCES

1. Lefebvre, R. A., Langley, B. R., and Thompson, A. B. *M3MS-16OR0401086 – Report on NEAMS Workbench Support for MOOSE Applications*. ORNL/TM-2016/572. Oak Ridge, TN: Oak Ridge National Laboratory, 2016. https://doi.org/10.2172/1328333.
2. Lefebvre, R. A., Langley, B. R., Miller, L. P., Delchini, M. G., Baird, M., and Lefebvre, J. P. *NEAMS Workbench Status and Capabilities*. ORNL/TM-2019/1314. Oak Ridge, TN: Oak Ridge National Laboratory, 2019. https://www.doi.org/10.2172/1570117.
3. Stauff, N., Lartaud, P., Jung, Y. S., Lee, C. H., Zeng, K., and Hou, J. *Status of the NEAMS and ARC Neutronic Fast Reactor Tools Integration to the NEAMS Workbench*. ANL/NEAMS-19/1. Argonne, IL: Argonne National Laboratory, 2019. https://doi.org/10.2172/1570009.
4. Stauff, N., Lee, C., Shriwise, P., Miao, Y., Hu, R., Vegendla, P., and Fei, T. *Neutronic Design and Analysis of the Holos-Quad Concept*. ANL/NSE-19/8. Argonne, IL: Argonne National Laboratory, 2019. https://doi.org/10.2172/1524786.
5. Fan, Y., Delchini, M. G., and Lefebvre, R. A. "Verification of Nek4nuc (Nek5000 Integrated in NEAMS Workbench) via Turbulent Pipe Flow Simulation." Presented at the ANS Winter Meeting and Nuclear Technology Expo, Chicago, IL, November 2020. https://www.osti.gov/servlets/purl/1731054
6. Cunningham, K. M., Powers, J. J., and Lefebvre, R. A. *Modeling the IFR-1 Experiment: A BISON Metallic Fuel Benchmark*. ORNL/TM-2019/1270. Oak Ridge, TN: Oak Ridge National Laboratory, 2019. https://doi.org/10.2172/1649544.
7. Zeng, K., Stauff, N. E., Hou, J., and Kim, T. K. "Development of Multi-Objective Core Optimization Framework and Application to Sodium-Cooled Fast Test Reactors." *Progress in Nuclear Energy* 120 (February 2020). https://doi.org/10.1016/j.pnucene.2019.103184.
8. Stauff, N., Maronati, G., Ponciroli, R., Ganda, F., Kim, T., Taiwo, T., Cuadra, A, Todosow, M., Talbot, P., Rabiti, C., Dixon, B., and Kim, S. *Daily Market Analysis Capability and Results*. ANL/NSE-19/5. Argonne, IL: Argonne National Laboratory, 2019. https://doi.org/10.2172/1511150.
9. Microsoft Corporation. "Language Server Protocol Specification – 3.16." Updated December 14, 2020. https://microsoft.github.io/language-server-protocol/specifications/specification-current.
10. "JSON-RPC 2.0 Specification." Updated January 4, 2013. https://www.jsonrpc.org/specification.
11. "Implementations – Tools Supporting the LSP." https://microsoft.github.io/language-server-protocol/implementors/tools.
12. Dominesey, K. A., Kowal, P. J., Eugenio, J. A., and Ji, W. "Scientific Workflows for MCNP6 and PROTEUS within the NEAMS Workbench." *EPJ Web Conferences* 247, 06052 (2021). https://doi.org/10.1051/epjconf/202124706052.
13. Asenov, D., Hilliges, O., and Müller, P. "The Effect of Richer Visualizations on Code Comprehension." Presented at the 2016 CHI Conference on Human Factors in Computing Systems, San Jose, CA, USA, May 7–12, 2016. https://doi.org/10.1145/2858036.2858372
14. Swiler, L. P., Lefebvre, R. A., Langley, B. R., and Thompson, A. B. *Integration of Dakota into the NEAMS Workbench*. SAND2017-7492. Albuquerque, NM: Sandia National Laboratories, 2017. https://doi.org/10.2172/1372616.
15. Ahrens, J., Geveci, B., and Law, C. "ParaView: An End-User Tool for Large Data Visualization." Part 9, Chap. 7 in *Visualization Handbook*. Elsevier, 2005.