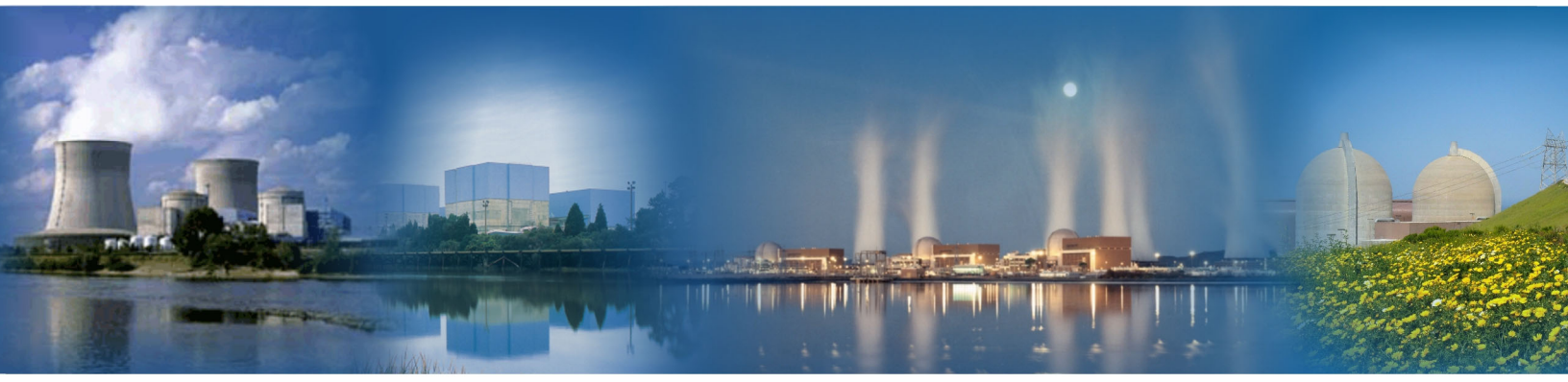
INL/RPT-24-78713

Light Water Reactor Sustainability Program

Upgrade of EMRALD to a Modern JavaScript-based Framework

Steven Prescott
Jared Nielson
John Stewart



June 2024

U.S. Department of Energy
Office of Nuclear Energy

INL/RPT-24-78713

Upgrade of EMRALD to a Modern JavaScript-based Framework

Steven Prescott
Jared Nielson
John Stewart
Idaho National Laboratory

June 2024

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Page intentionally left blank

ABSTRACT

Event Modeling Risk Assessment using Linked Diagrams (EMRALD) is a software tool developed at Idaho National Laboratory for researching the capabilities of dynamic probabilistic risk assessment. It provides a simple interface to represent complex interactions often seen when developing dynamic models. EMRALD can also interface with other applications by modifying inputs, running, and using their results within EMRALD for dynamic and integrated assessment. This report goes over the work performed as part of the Risk-Informed Systems Analysis Pathway under the Light Water Reactor Sustainability program to upgrade the EMRALD software.

Page intentionally left blank

CONTENTS

FIGURES

Page intentionally left blank

ACRONYMS

DPRA	Dynamic probabilistic risk assessment
EMRALD	Event Modeling Risk Assessment using Linked Diagrams
INL	Idaho National Laboratory
JSON	JavaScript object notation
LWRS	Light Water Reactor Sustainability
MAAP	Modular Accident Analysis Program
MUI	Material User Interface
PRA	Probabilistic risk assessment
RISA	Risk-Informed Systems Analysis
UI	User interface

Page intentionally left blank

Upgrade of EMRALD to a Modern JavaScript-based Framework

1. BACKGROUND

Event Modeling Risk Assessment using Linked Diagrams (EMRALD) is a dynamic probabilistic risk assessment (DPRA) tool developed at Idaho National Laboratory (INL) and has been used for many laboratory and academic research projects. EMRALD started out as a Laboratory Directed Research and Development project in 2005 to prove our DPRA methods. In 2016, the Light Water Reactor Sustainability program needed a way to link external event simulations dynamically to risk analysis with the timing of component failures and operator actions. EMRALD was further developed to be used for external hazard analysis research projects [1–4]. Recent projects in LWRS include human reliability projects, dynamic and classical PRA coupling, physical security optimization [5–7]. Several commercial companies have shown interest in EMRALD and integrated it into their tools [8, 9].

In making updates to EMRALD for recent projects, it was determined that the web-based user interface (UI) for modeling was reaching the end of its life cycle. Several of the packages used were no longer supported and would not be receiving any security updates. It was determined that this code needed to be rewritten and upgraded to a more modern framework. This report outlines the work done for upgrading EMRALD to extend its usability lifespan and enable planned research projects.

2. UPGRADE PLANNING

Several months were spent looking at existing features and tools used in the current version of EMRALD and finding and testing newer tools that could achieve equivalent features while minimizing work and maintainability of the new version. The new version will upgrade existing models to the new 3.0 version, but any 3.0 versions will not be compatible with the older UI or solve engine.

2.1 Packages Used

The web-based EMRALD modeler UI was developed using Angular JS, a JavaScript-based web framework from Google [10]. Updates to Angular JS were stopped in 2022. There are new versions of Angular, but there were no upgrade paths from Angular JS to the new Angular framework, and upgrading any project using Angular JS would require a software rewrite.

Other packages were used for many of the features in EMRALD. The diagram and logic tree viewer and editor uses MX_Graph library [11]. The MX_Graph library is also no longer supported and has not been updated since 2020. When reviewing the remaining packages used, none were deemed significant as they all had newer versions or could easily be replaced with similar packages.

2.2 New Package Research

Various front-end technologies were evaluated for the new EMRALD UI. Given that the previous UI was built using AngularJS, upgrading to Angular 17 [12] was initially considered a viable option. However, due to the straightforward nature of EMRALD and the extensive ecosystem of libraries available, React [13] emerged as the preferred choice for the new UI development. React's flexibility, robust community support, and extensive library ecosystem made it particularly well-suited for the EMRALD UI upgrade.

A critical component of the EMRALD application is the diagram tools used for model visualization and construction. To enhance the user experience and streamline the process of building models, the React Flow [14] library was selected. Initially, D3.js [15], a powerful JavaScript library for producing dynamic, interactive data visualizations, was considered because it is used by the EMRALD UI for the Sankey diagrams and remains a popular library. However, during the research phase, React Flow emerged

as the preferred choice for the state diagram editor and logic tree editor as there were easy-to-use components in React Flow making development faster and reducing long-term maintainability. React Flow significantly improves the visual appeal and usability of the UI. The UI's integration with React Flow allows for a more dynamic and responsive interface, providing users with a more intuitive and efficient model-building experience.

Another critically important feature of EMRALD is the capability to manage multiple windows, including the functionality to drag and drop these windows. Additionally, the application requires the ability to drag and drop various items to different locations within the interface and handle the corresponding data transfer seamlessly. To achieve these functionalities, the libraries React-RND [16] and React-DND [17] were selected. React-RND facilitates the resizing and dragging of windows, while React-DND provides a powerful and flexible drag-and-drop framework. Together, these libraries enhance the interactivity and user experience of EMRALD, ensuring that users can efficiently organize their workspace and manage the application's components.

In addition to the core libraries, several other useful tools were employed to craft the new UI for EMRALD, including Material-UI [18], React-Icons [19], and Day.js [20]. Material-UI was chosen for its comprehensive suite of predesigned components that adhere to Google's Material Design guidelines, ensuring a consistent and visually appealing user interface. React-Icons was utilized to provide a vast array of customizable icons, enhancing the visual clarity and user experience of the application. Day.js was incorporated to handle date and time manipulation. Together, these libraries contribute to a polished, user-friendly, and high-performance UI for EMRALD.

3. MODELING UI PROJECT SOURCE LAYOUT

EMRALD operates without a traditional backend or database for data fetching. Instead, users build a JavaScript object notation (JSON) model, a common human- and machine-readable format, that later is used with the EMRALD simulation engine. The new EMRALD UI is designed around a global store or local storage that houses this JSON model. This model is organized into various sections, such as diagrams, actions, and events. To manage these sections efficiently, a React context [21] is created for each subsection, allowing relevant parts of the application to interact with and manipulate the data via their respective contexts. React contexts are a feature of the React library, which allows you to share data across multiple components without having to pass props down manually at every level. Using React context for state management within each subsection of the model ensures that changes in one part of the application can be efficiently propagated to other relevant parts, enhancing the responsiveness and interactivity of the UI.

The new structure, shown in Figure 1, emphasizes a robust component architecture. This architecture is modular with components segmented into distinct parts, including diagrams, forms, windows, and page layouts. Each component is designed to be as simple as possible, leveraging React hooks to manage logic where necessary. This modular approach enhances maintainability by making it straightforward for developers to locate and understand various parts of the application. The clear separation of concerns allows for individual components to be developed, tested, and debugged in isolation, which improves the overall quality of the codebase.

Additionally, the modular architecture promotes reusability, enabling components to be easily reused across different parts of the application. This not only accelerates development but also ensures a consistent user experience throughout the application. This design philosophy results in a more robust, efficient, and user-friendly application, enhancing the overall experience for both developers and end-users.

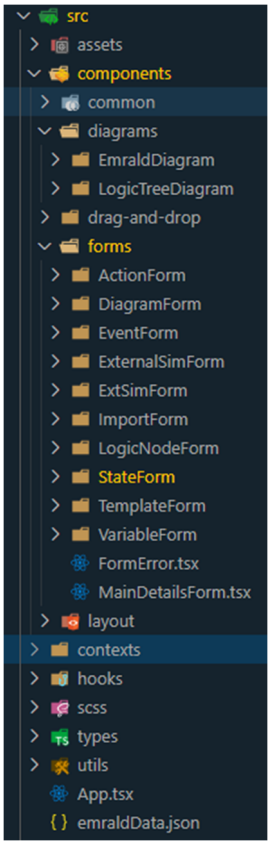


Figure 1. EMRALD project file structure.

4. MODELING UI DEVELOPMENT

One of the goals in the upgrade was to keep the UI experience as close as possible to the original while still making it easier to use and updating the tools. The graphing tools are the most different but maintain the original workflow, and the forms have a very similar layout and feel. This section outlines the main work that was done in the different areas showing the biggest differences or additional features.

4.1 Main User Interface

To enhance the user experience while preserving the familiar feel of the previous EMRALD UI, several new features have been implemented. These enhancements not only eliminate the need for users to relearn the application but also streamline the process of creating models.

The main canvas of the EMRALD application now features a speed-dial button, shown in Figure 2. When hovered over, this button provides users with quick access to create new forms when constructing their model. Previously, users had to right-click on each menu item and then select the new form button, which was a more time-consuming process. The new speed-dial button significantly improves usability by allowing users to quickly and easily locate and create new items, thereby accelerating the model-building process.

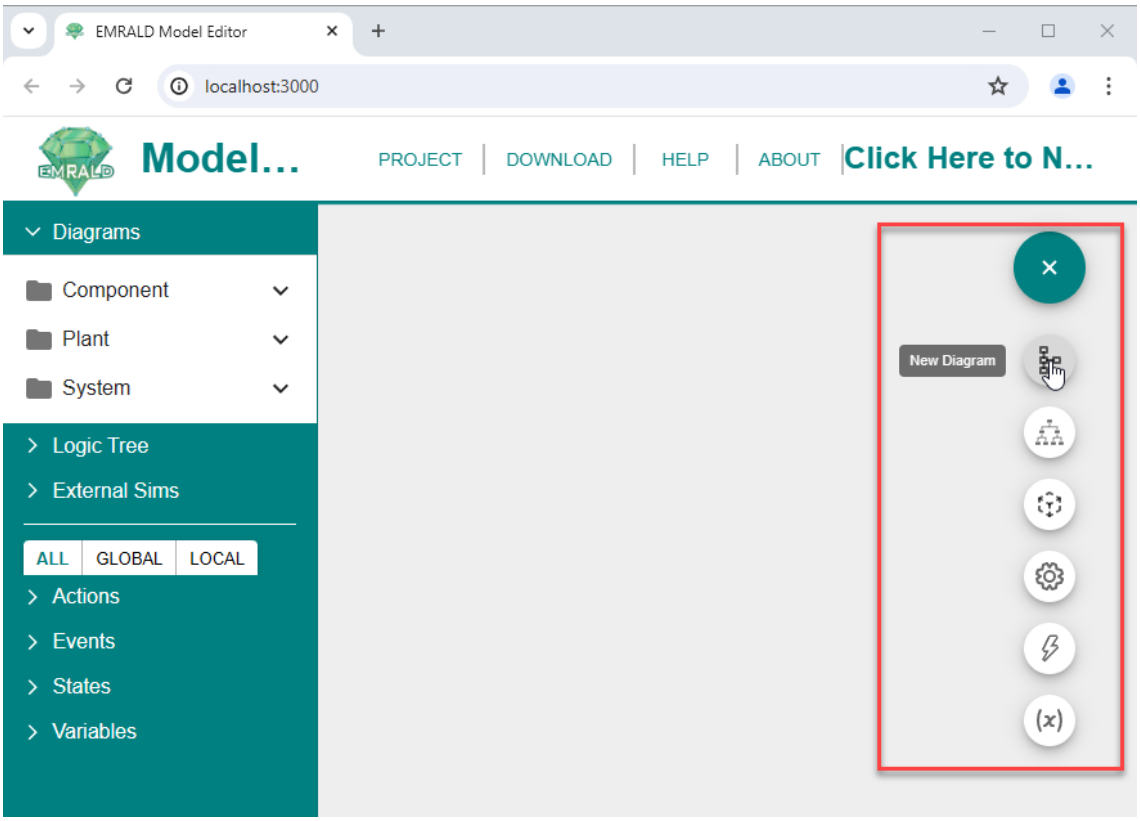


Figure 2. Example of speed-dial button for new forms.

Leveraging the capabilities of the React-RND and React-DND libraries, a new custom window component has been developed. This component grants users' extensive control over the windows on their canvas. These windows can be easily dragged, resized, minimized, and expanded, offering a high degree of flexibility in managing the workspace. Minimized windows, as shown in Figure 3, are conveniently listed at the bottom of the canvas, and a simple click restores them to their previous size, helping users maintain an organized and clutter-free workspace. This functionality enhances the overall user experience by making window management intuitive and efficient.

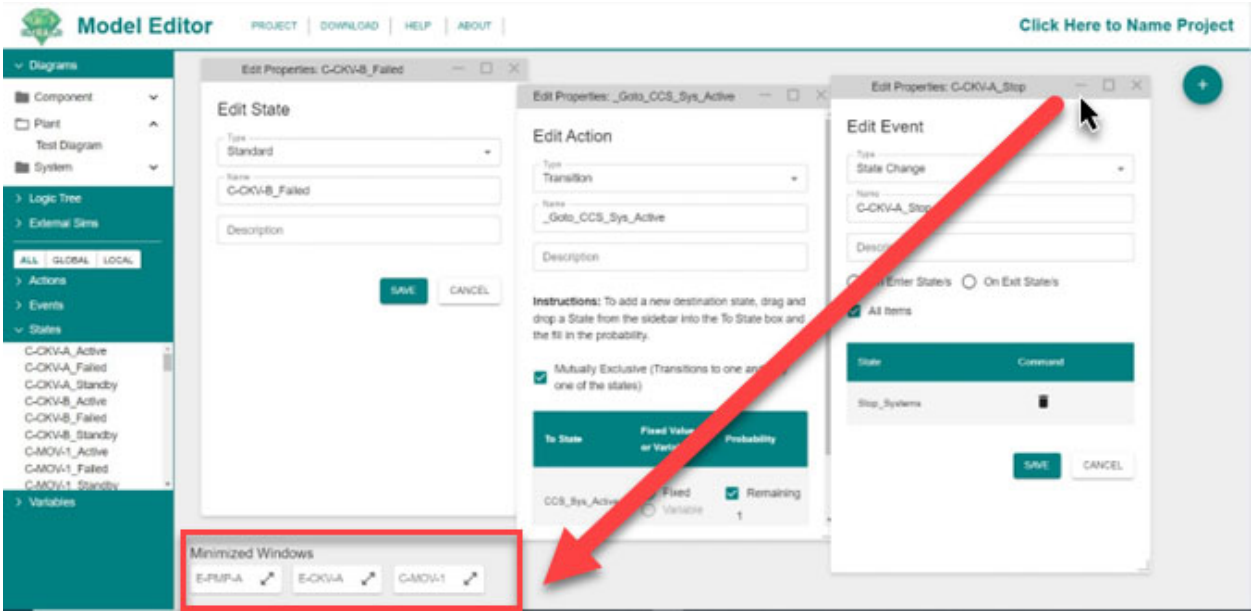


Figure 3. Example of EMRALD window capabilities and minimization.

Additionally, instead of relying on the browser's alert window for entering information such as the model's name, the new UI now utilizes Material-UI dialog windows. These dialogs are centered on the web page, providing a more aesthetically pleasing appearance that aligns with the overall design of the application. This integration, as shown in Figure 4, not only enhances the visual appeal of the UI but also ensures a consistent user experience across different parts of the application.

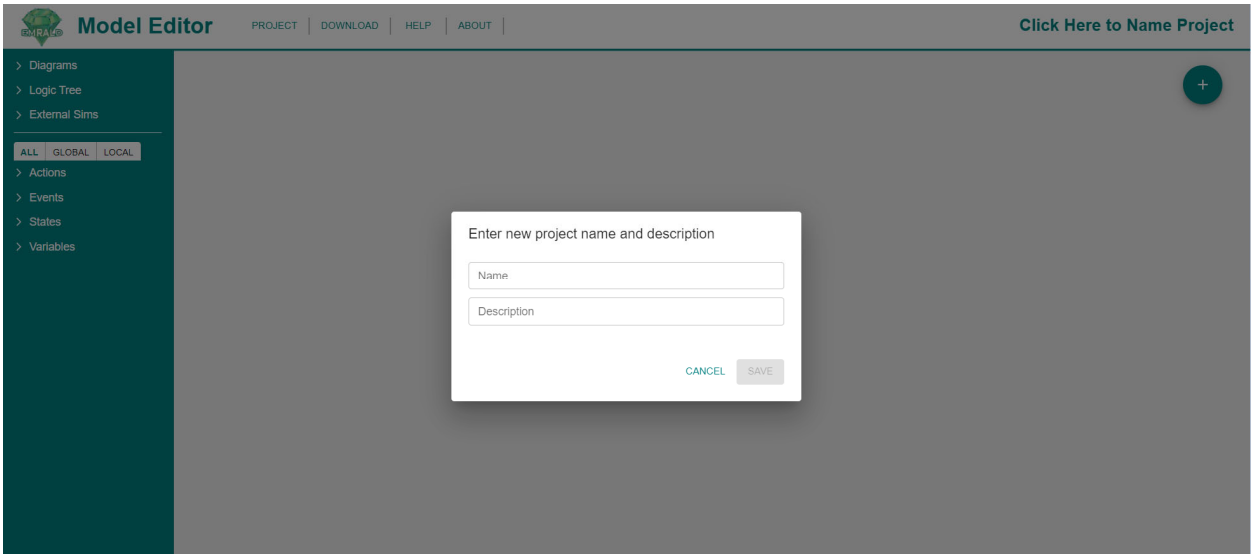


Figure 4. Example of dialog window.

4.2 Diagram Editor

The diagram editor has undergone significant improvements with the integration of the React Flow library, making it much easier and more intuitive to work with. In the previous UI, moving state nodes within the diagram was cumbersome, requiring an initial click before dragging to select the node, and the space to which the node could be moved was limited. In the new UI, users can effortlessly click and drag

nodes to any desired location within the window, significantly enhancing the ease of interaction. Additionally, the diagram's size and scale can expand to accommodate an almost limitless number of nodes, providing greater flexibility in model construction.

The React Flow library introduces new controls, shown in Figure 5, that greatly improve the usability of the diagram editor. Users can now zoom in and out on the diagram, allowing for detailed inspection or an overview of the entire model as needed. A mini-map feature has been added, helping users to easily navigate to nodes that may not be within the current view. This improves the overall navigation and usability, especially in complex diagrams.

Drag-and-drop functionality has been enhanced using the React-DND library, making it clear and easy to see which items can be dropped within the diagram. This improvement ensures a more intuitive and efficient user experience when modifying the model. Additionally, while the previous UI allowed users to move actions up or down within an event, the new UI extends this functionality by enabling both actions and events to be moved up or down. This provides users with greater control over the node configuration and simplifies the process of organizing and adjusting the model.

The handles for connections between state nodes have also been improved. In the older UI, users had to hover over the action item to access the handle, which added confusion and slowed down the process. In the new UI, these handles are easier to locate and drag to other nodes, streamlining the creation and adjustment of connections between state nodes.

Moreover, in the previous version, if a diagram had links to other diagrams, it merely included an icon to indicate this link, without offering further assistance in navigating to the linked diagram. The new UI addresses this limitation by not only featuring these link icons but also allowing users to click on them to open the linked diagram in a new window. This enhancement makes it quick and easy to navigate between related diagrams, significantly improving the user experience.

Overall, these enhancements to the diagram editor, facilitated by React Flow and React-DND, result in a more user-friendly, efficient, and flexible tool for building and managing models in EMRALD.

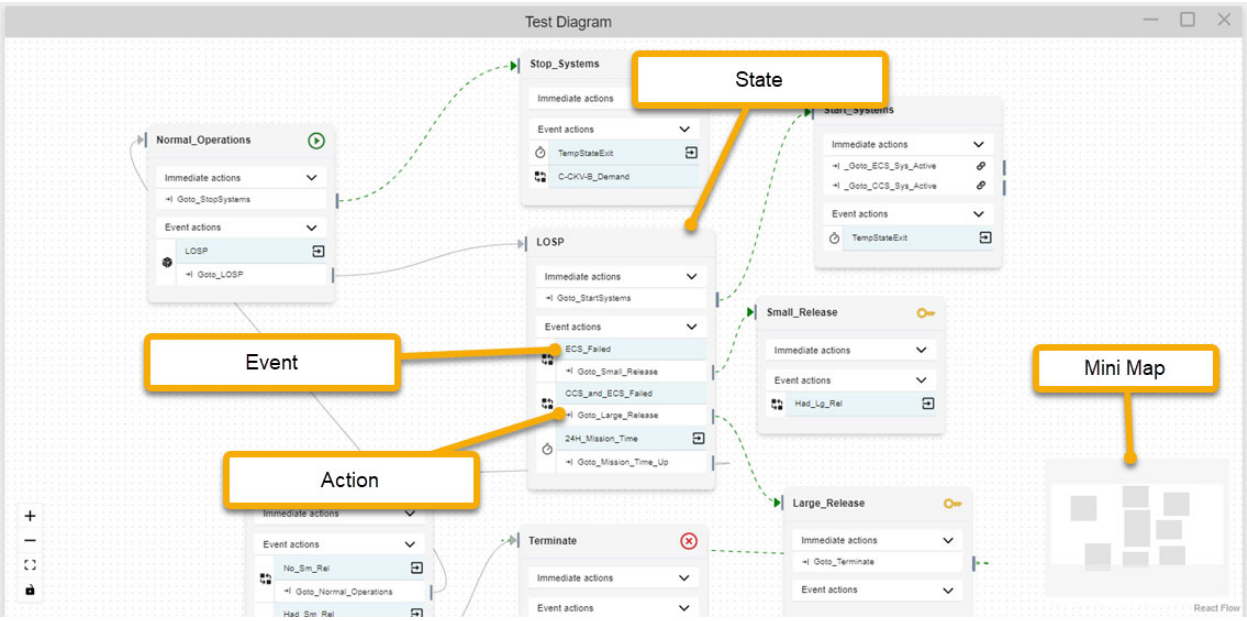


Figure 5. Example of EMRALD diagram.

4.3 Logic Tree Editor

The logic tree diagram in EMRALD has seen significant upgrades with the integration of React Flow, aligning it with the improvements made to the main EMRALD diagrams. These enhancements bring consistency to the UI, making it easier for users to become familiar with and navigate the application.

One of the notable upgrades is the refined appearance of the nodes, shown in Figure 6. Nodes now feature cleaner designs with embedded buttons and more visible gate-type icons, allowing users to understand the diagram at a glance. The improved visual clarity significantly enhances the user experience by making it easier to work with the logic tree.

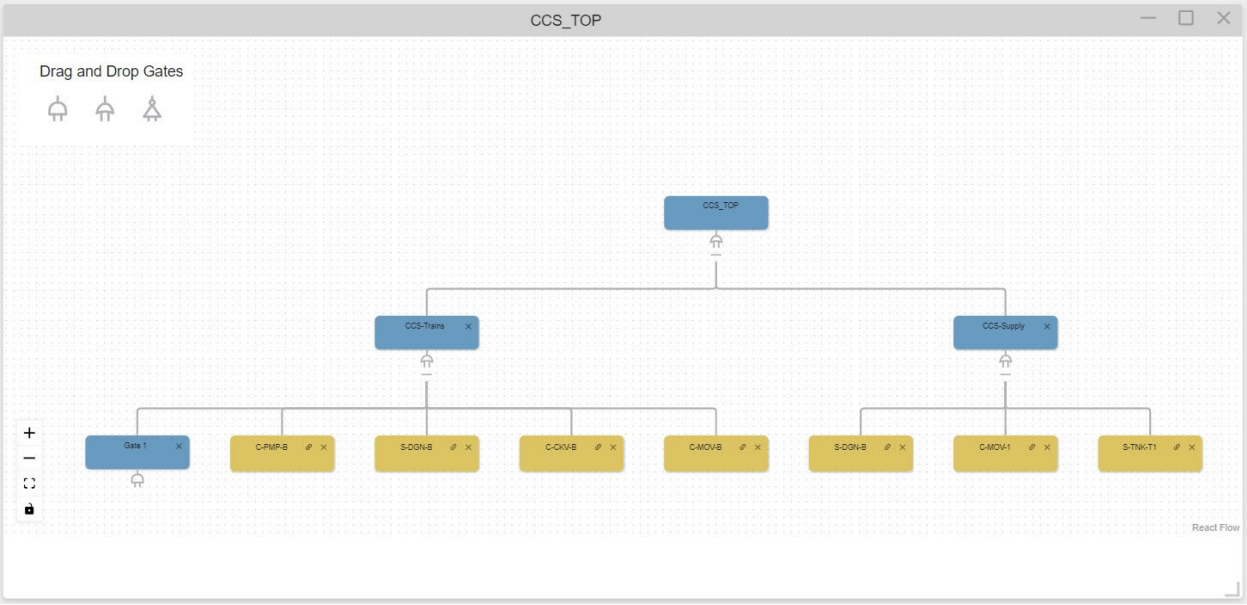


Figure 6. Example of logic tree diagram.

Several powerful features have been added to the logic tree, boosting its functionality and user-friendliness. Users can now double-click on the title of a node to activate a text field, enabling them to quickly and easily update the node's name without needing to open a separate form. An example of this functionality is shown in Figure 7. This same double-click functionality is available for editing node descriptions, making it easier to understand and manage the model as it is being constructed.

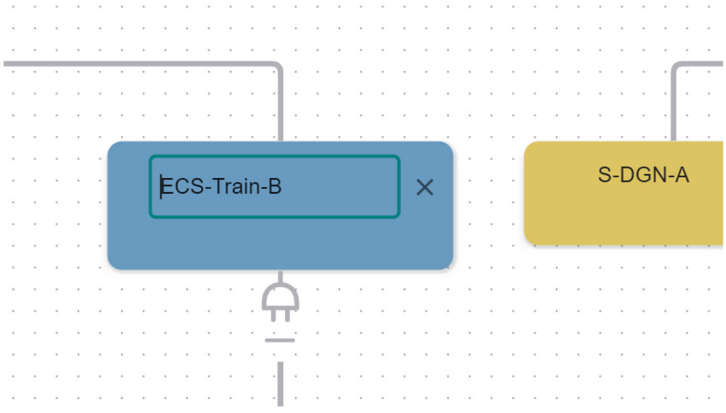


Figure 7. Example of node double-click edit functionality.

A major new feature is the right-click context menu, displayed in Figure 8, which provides a list of actions specific to the node. This menu offers options that were previously unavailable in the older UI, such as copying and pasting nodes within the tree. This capability, combined with the streamlined editing process, significantly accelerates the tree-building workflow. Additionally, the ability to remove a gate node while keeping it within the model for reference by other trees, or to delete the gate entirely, has been introduced. This feature provides greater flexibility and control over the model structure.

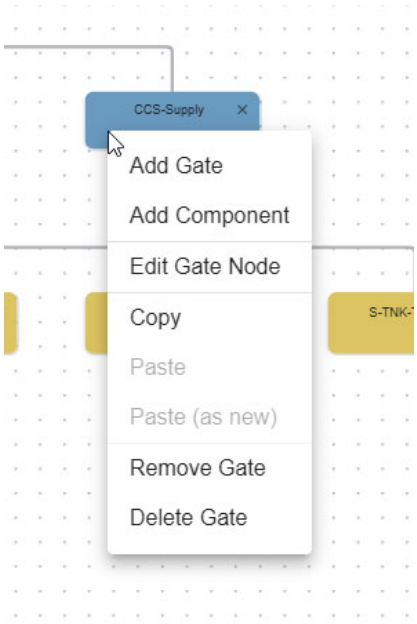


Figure 8. Example of gate node right-click context menu.

Component nodes now have a right-click menu that offers quick access to the edit form, deletion options, and the ability to open the diagram in a new window. Also, if a component node has been edited to deviate from default values, like what is shown in Figure 9, it is highlighted in a darker color and marked with an edit icon in the top left corner. This provides users with a quick visual indicator that the node has been modified, improving the ease of managing and reviewing the model.

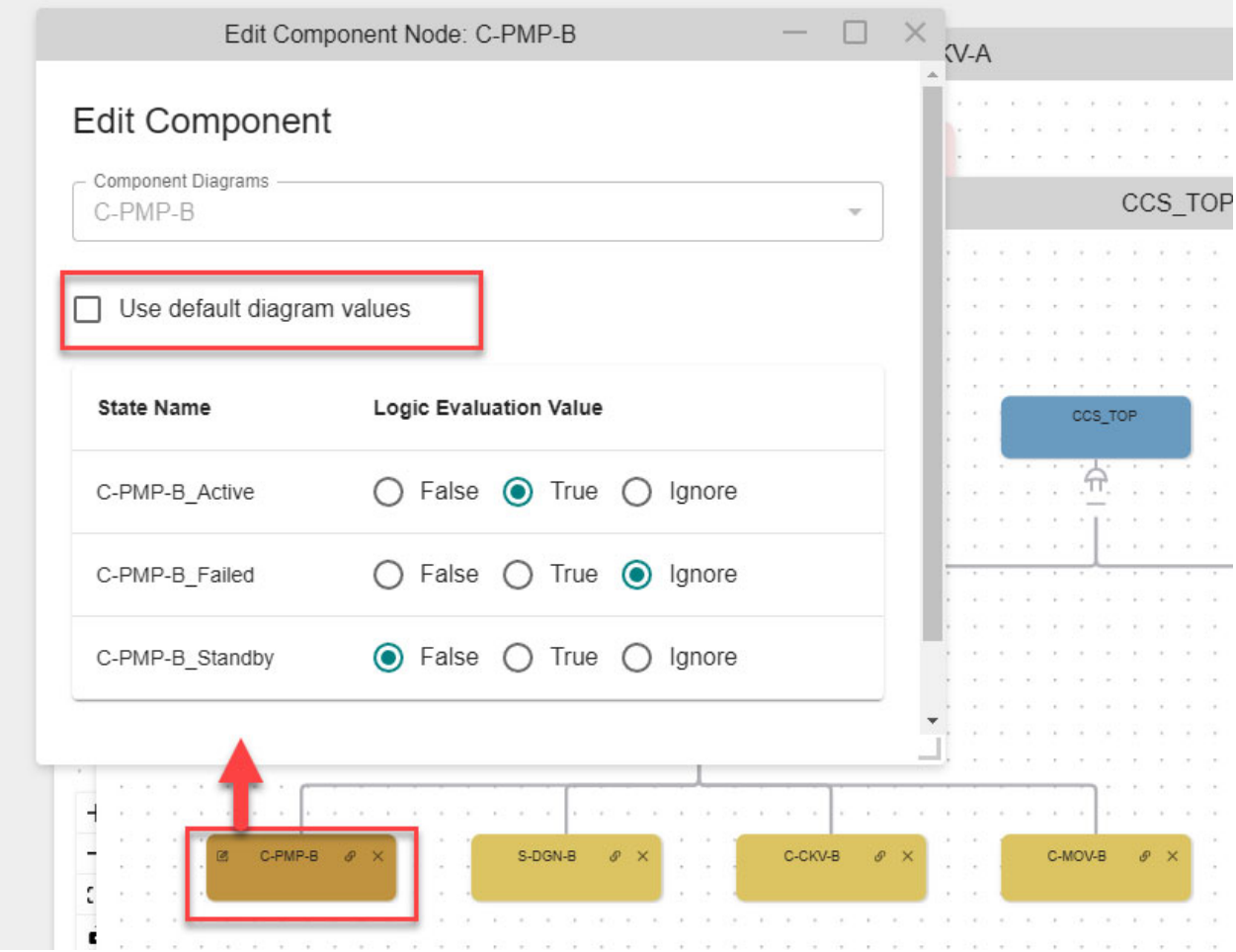


Figure 9. Example of non-default value highlighting.

When dragging gate items onto a node within the tree, the target node is highlighted green, providing clear feedback, as shown in Figure 10, that the new item can be dropped there. This visual aid simplifies the process of adding new items to the tree, reducing the likelihood of errors and enhancing overall efficiency.

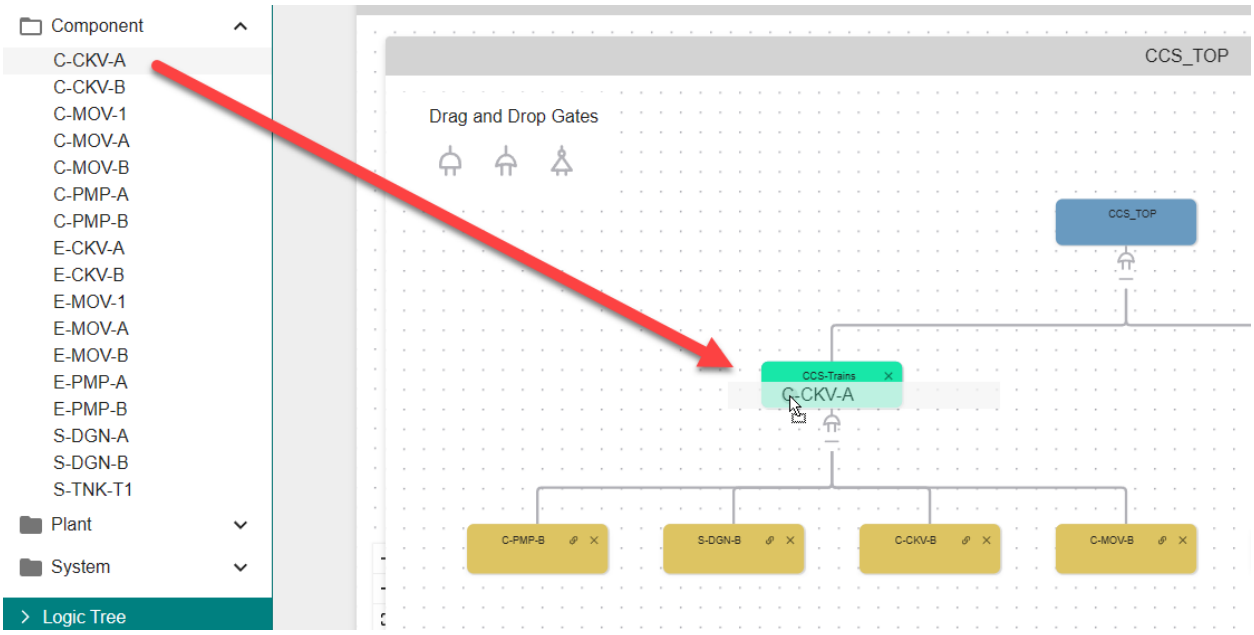


Figure 10. Example of droppable area highlighting.

4.4 Edit Properties Forms

The forms for editing properties have been updated to be consistent with new user interface styling. Several changes were implemented to clarify properties and facilitate the user experience in general. Users can now navigate and utilize the forms more efficiently.

4.4.1 Diagram and State Forms

The diagram form has been updated to now include a new property: the diagram label. In addition to its previous properties of type, name, and description, the diagram label indicates which category the diagram belongs. Previously, this was accomplished by the diagram type, but changes were made to give the user more flexibility in their diagram designs as well as to increase user understanding. The diagram type is now either “Single State” or “Multi State” to indicate whether this type of diagram can be in multiple states at once or not. These small changes to the diagram form—as well as the new user interface styling—help the user diagram effectively.

To edit or create a state form, the following form is used as shown in Figure 11. It can be accessed from an open diagram either by right-clicking on the diagram and choosing “New State” or by right-clicking on a state and choosing “State Properties.” Like the previous UI, the “Default Logic Tree Evaluation Value” will only appear if the assigned diagram is a single state diagram. The new design uses text fields with descriptive labels above and in the empty box to indicate what should be entered, as shown in Figure 11.

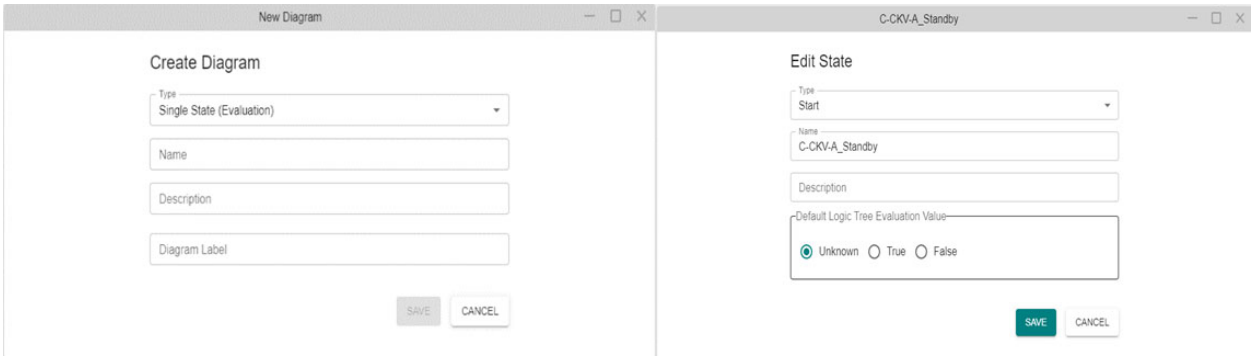


Figure 11. Example of standardized property forms with labels on the fields.

4.4.2 Variable Forms

Variables can be created from the right drop-down menu. The new design follows the same format as the old UI in what information is required but follows the new format concerning how information is entered. Labeled text fields indicate what should be entered for the name and description. All forms of data entry utilize the Material User Interface (MUI) React library. One of the largest updates for the variable forms was this new droppable box, where the user can deposit state items that should be used for accrual variables, as shown in Figure 12.

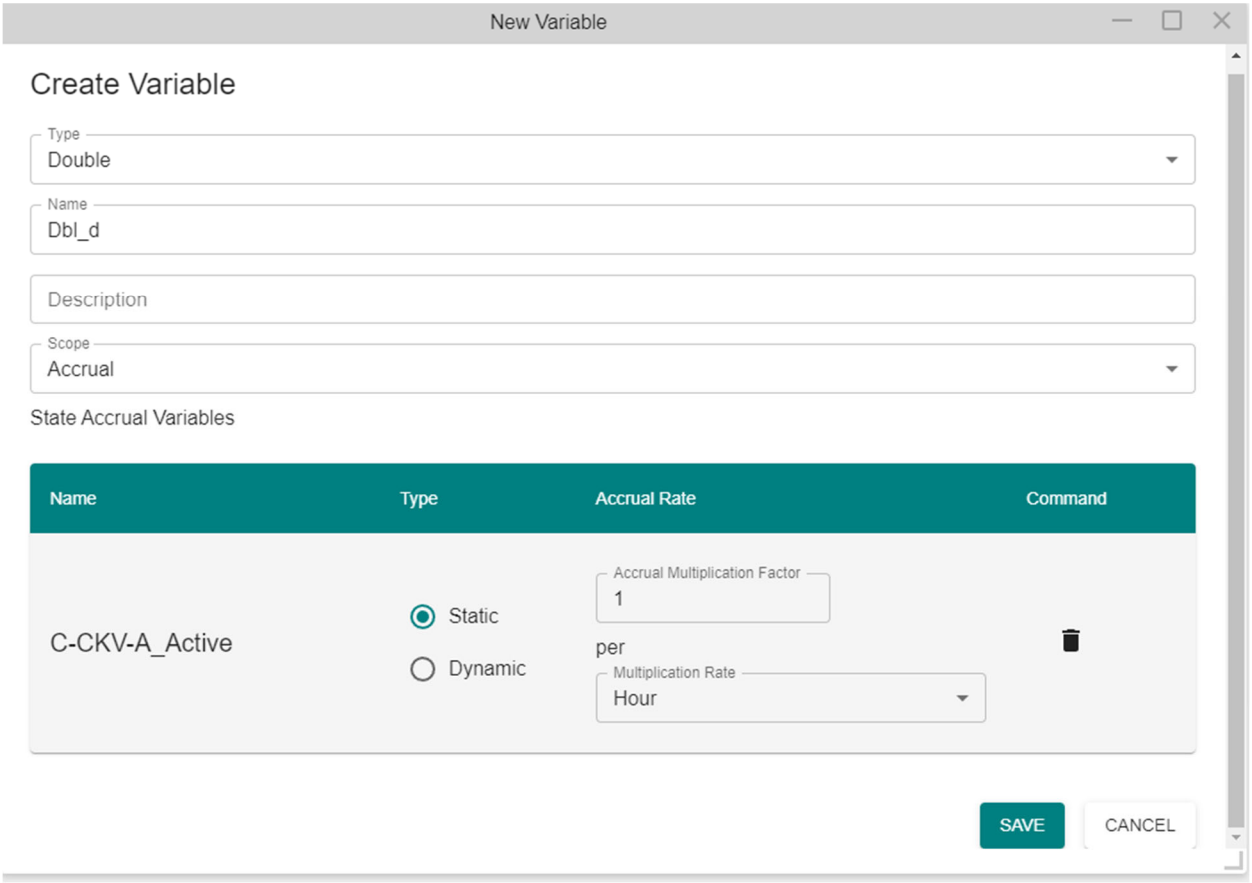


Figure 12. New variable form showing an accrual variable with droppable area for the accrual states.

4.4.3 Event and Action Forms

The event forms—like the variable forms—follow the same outline as the old UI but with some notable changes and updated styling. Besides styling, the most notable change is in the distribution form, shown in Figure 13, where a default rate was added. This allows for a user to set and use a default rate for each variable. This makes the table easier to manage since if the rate is not of concern to the user, then they will not have to bother setting a rate for each variable.

New design work for the action forms includes a modernized droppable box in which the user can drop state items as well as newly styled components for adding code, as shown in Figure 14. The newly styled components should help the user enter information, remove any confusion, and streamline the user experience. Each form has been updated to reflect the design styles of the new user interface resulting in forms that are visually pleasing and easy to work with.

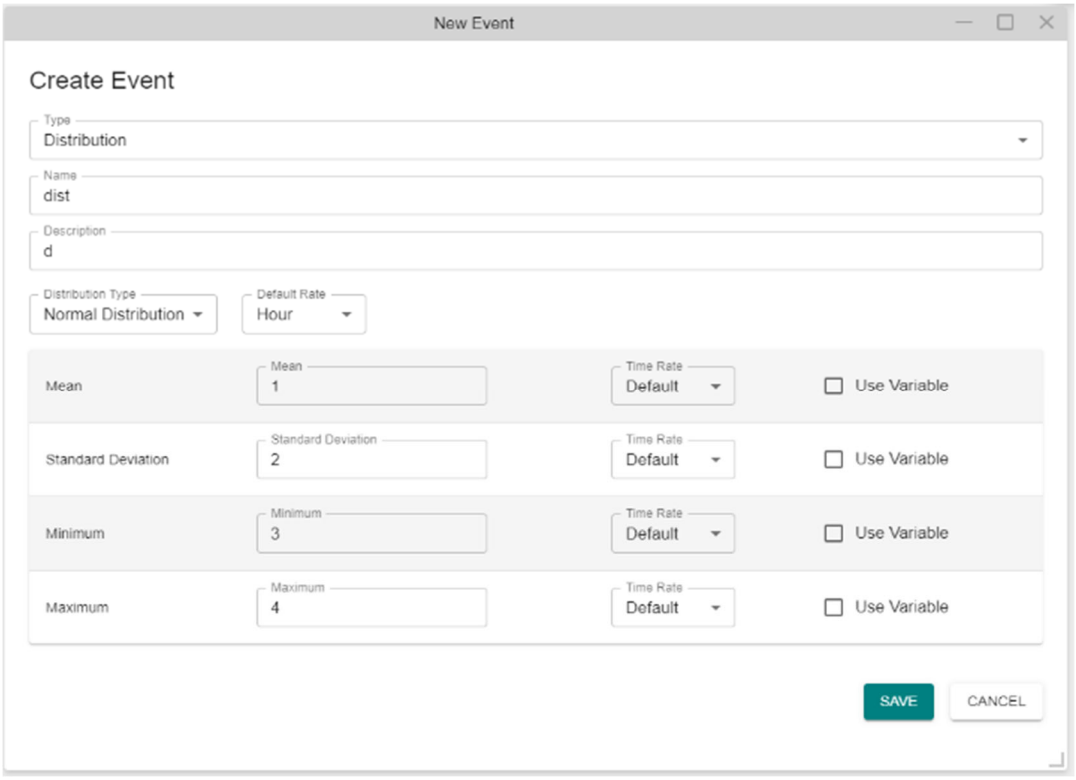


Figure 13. Form showing a distribution event with the parameters for a normal distribution.

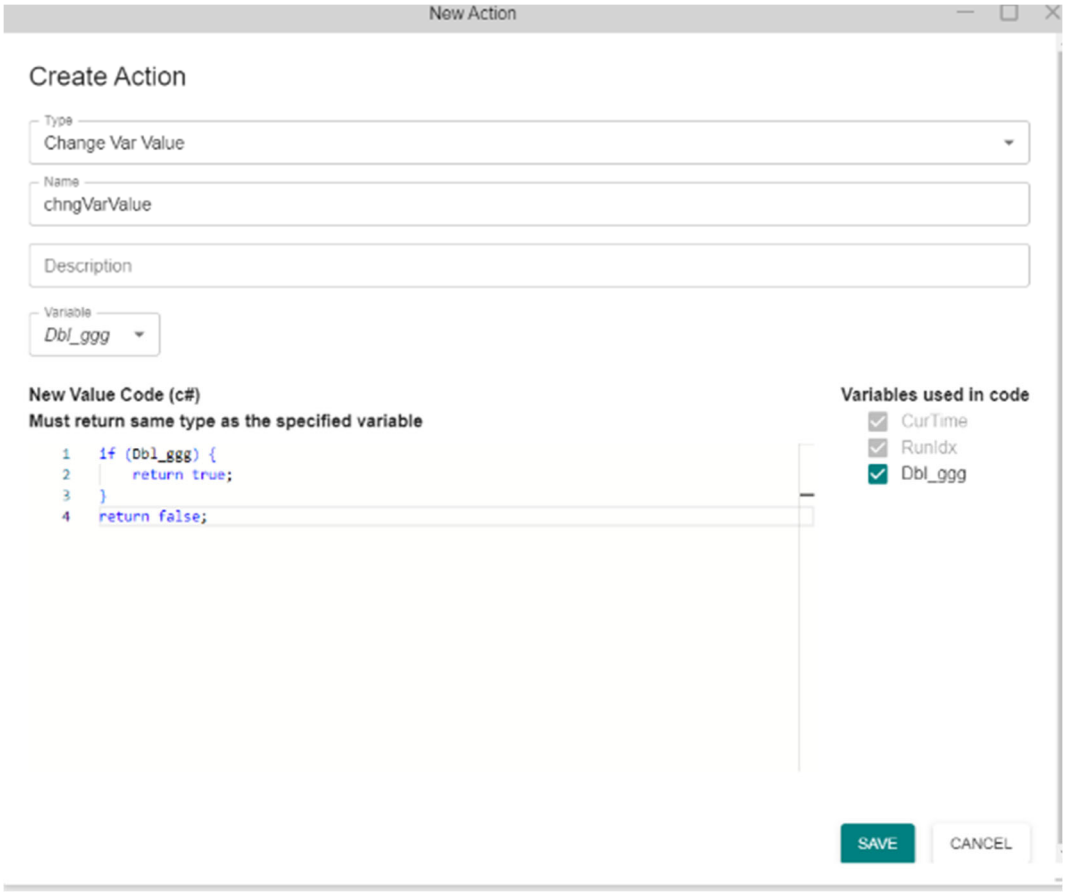


Figure 14. EMRALD form for a change variable value action.

5. EMRALD MODEL USE AND CHANGES

In reviewing the existing EMRALD software, there were several things that would help minimize the effort in rewriting the code. One of those things was how the EMRALD model is structured and processed. For long-term maintainability, a consistent and standardized way of making upgrades and checking for model syntax error was needed. To accomplish this, a schema defining the EMRALD model structure and requirements was needed and could be used in several areas of development.

5.1 Model Schema

The EMRALD model is saved in the JSON format which is both easily processed by most software languages and can still be read, understood, and modified by humans. JSON also has a publicly defined schema language which made it easy to define exactly what is allowed and required for the EMRALD model. A JSON schema was defined early in the development process. This schema was used in several areas: (1) autogenerating and documenting the object types for use in the code, (2) checking the model before saving and find any syntax errors, and (3) implementing the upgrade process described in Section 5.2.1.

5.2 Upgrade Methods

As with all software and tools, it is expected that EMRALD will progress and change over time. This includes the data and format in the EMRALD model. The current model upgrade to a new schema was very limited and not well defined. It also required an upgrade method or old version compatibility in both the model-building UI and the solve engine. Research was done on how to combine this as one code or

tool to minimize work through code reuse and reduce errors. One solution was to require the user to upgrade an existing model in the web UI before opening in the solver, but this seemed less optimal. The second option was to use a package that could execute the JavaScript code in the solver C# code. This second method was chosen.

5.2.1 Web Editor Model Upgrade

To effectively perform model upgrades, a folder with schema logic and standard class structure was set up for each schema change as shown in Figure 15. The class knows both its schema and the previous and implements defined functions to upgrade model pieces. When first opening a project, the upgrade function is called which reads the current version number and runs each schema upgrade that is needed to bring it up to the current version.

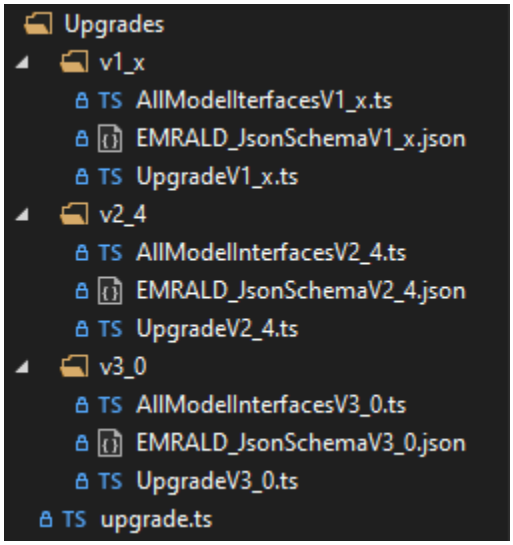


Figure 15. Upgrade folders and files, where a folder and processing files are created for each schema change.

When a new version of the model is needed, a folder with the version is created. Changes are made to the existing schema and saved in the folder, then the schema is used to autogenerate all the TypeScript [23] object types as one file and saved in the folder. The developer then copies and splits out all the different object types and updates the current EMRALD object definitions. Using the definitions from the current and previous version, the developer creates an upgrade class implementing the functions to update the model returning a new JSON model string.

5.2.2 Simulation Engine Model Upgrade

The package ClearScript from Microsoft [22] allows JavaScript to be run inside C#; however, this requires the JavaScript to be included as a file in the solve engine code. The UI code is written in TypeScript, but this translated automatically down to JavaScript for execution as a web application. A script to convert the TypeScript for just the upgrade process was added to the source code and is called whenever the solve engine is compiled and the code has been updated. The upgraded JavaScript source is then copied to the solve engine project. When the user opens an EMRALD model in the solve engine, it executes the upgraded JavaScript function passing in the EMRALD model text and returns and upgraded text.

5.3 Cross References and Item Changes

EMRALD modeling pieces, such as variables, events, actions, diagrams, etc., can be referenced by name by many other pieces of the model. Name vs. ID references are used to make the model human

readable; however, this makes it more difficult to change or alter a name since all the references also need to be updated. The old EMRALD modeling UI had a long piece of code that processed all the model pieces looking for references and making changes if needed. This worked well but was cumbersome to modify and understand by new developers.

A simpler method was implemented in this updated version using JSON path for all referencing and updates [24]. JSON path allows the user to define a simple rule to find items in a JSON file, similar to how a regular expression works. A set of JSON path rules was defined for each of the EMRALD items; these rules specified how to get all the names referenced by the item. For example, a state item can reference diagrams, events, and actions with the JSON path rules shown in Figure 16. A second set of rules was added to get all the items that reference a specific item. For example, a state can be referenced by diagrams, actions, events, logic node, and variables, with JSON path rules shown in Figure 17.

```
//States - items referenced by the specified 'nameRef' State
const InStateRefs: Array<[string, MainItemTypes]> = [
  ["$.StateList[?(@.name == 'nameRef')].diagramName", MainItemTypes.Diagram],
  ["$.StateList[?(@.name == 'nameRef')].events", MainItemTypes.Event],
  ["$.StateList[?(@.name == 'nameRef')].immediateActions", MainItemTypes.Action],
  ["$.StateList[?(@.name == 'nameRef')].eventActions[*].actions", MainItemTypes.Action],
];
```

Figure 16. JSON path in the C# code for looking up items referenced by a state.

```
//States - all the items that reference the specified state 'nameRef'
const UsingStateRefs: Array<[string, MainItemTypes]> = [
  ["$.StateList[?(@.name == 'nameRef')].name", MainItemTypes.State],
  ["$.DiagramList[*].states[?(@ == 'nameRef')]", MainItemTypes.Diagram],
  ["$.ActionList[*].newStates[?(@.toState == 'nameRef')].toState", MainItemTypes.Action],
  ["$.EventList[*].triggerStates[?(@ == 'nameRef')]", MainItemTypes.Event],
  ["$.LogicNodeList[*].compChildren[*].stateValues[?(@.stateName == 'nameRef')].stateName", MainItemTypes.LogicNode],
  ["$.VariableList[*].accrualStatesData[?(@.stateName == 'nameRef')]", MainItemTypes.Variable]
];
```

Figure 17. JSON path in the C# code for looking up items that reference a state.

These rules provide several software use and maintainability benefits. First, they allow the developers to use the same function for updating any item and any references to it. Second, it provides a simple method for getting an item and all the items it uses, which are used to copy a piece of the model or make a template. Finally, if new items are added, it makes it easy to add those changes to the reference and update calls by simply adding a new line to the rules for the applicable items.

6. DOCUMENTATION

With the user interface changes, updates to the documentation website, as shown in Figure 18, were also performed. This section goes over how the documentation is set up and changes that were made. A new documentation piece was added to provide help for anyone desiring to understand and use the EMRALD model JSON format.

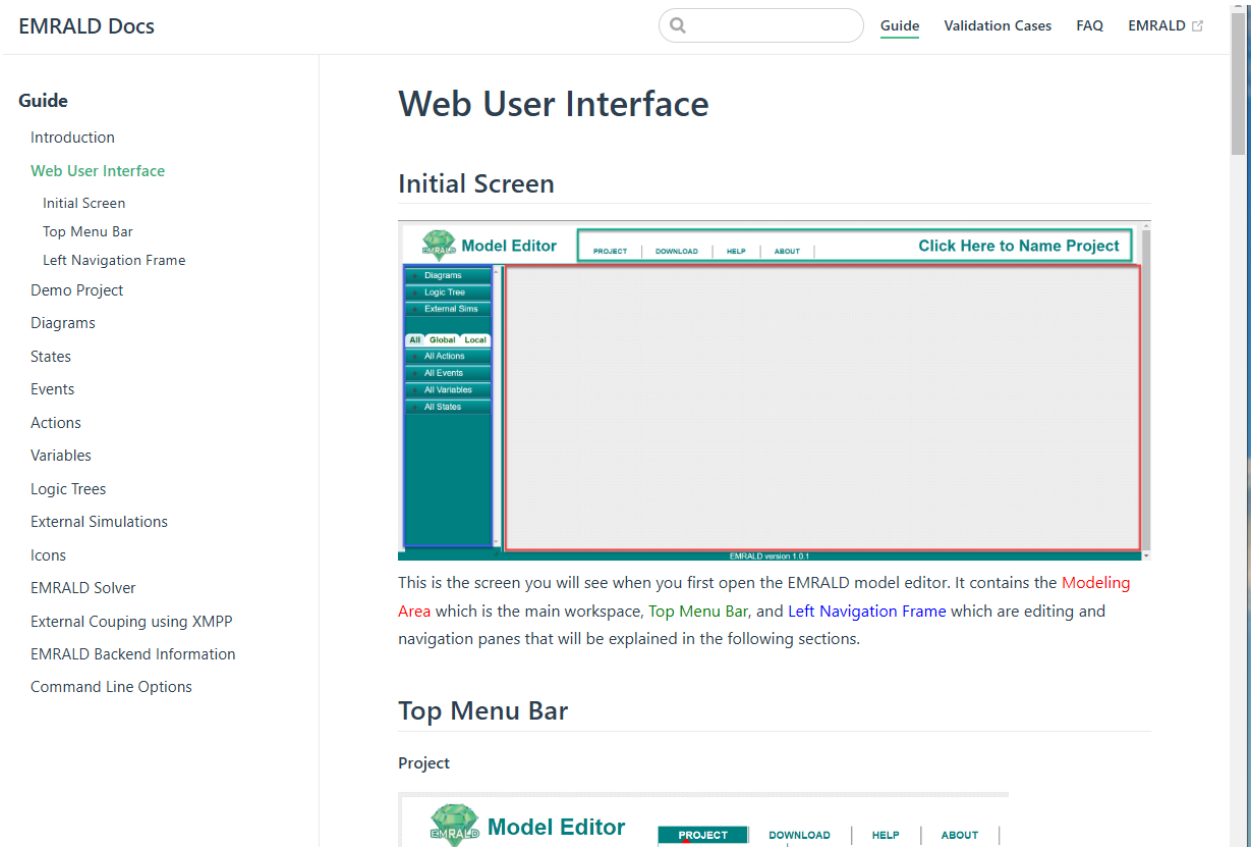


Figure 18. Previous documentation website written in markdown language.

6.1 Markdown Help

The markdown documentation provides a wealth of information for users to better understand and utilize the EMRALD application. These help pages were updated to be consistent with the new UI. This update involved replacing outdated images, rewording text to better match with the new designs, and adding text to explain new features and components. As the help pages are divided into sections, each section needed considerable rework.

6.1.1 Web User Interface

The web UI section of the documentation needed new images to include the new drop-down menu for creating diagrams, logic trees, events, etc. Since this is a new feature in the UI, the help text was modified to include it, and an additional section was added to explain how it works. This additional section explains how hovering over each icon in the drop-down menu displays text indicating what the button will do. When clicked, the icon will open a form to create the desired component.

6.1.2 Diagrams

The method of creating a new diagram has been updated. Instead of right-clicking on the left navigation frame, you need to use the drop-down menu on the right-hand side. In addition, what used to be referred to as the diagram type is now referred to as the diagram label. The diagram type still exists, but the options are now “Single State (Evaluation)” or “Multi State.” These refer to whether the diagram will be a single state or a multi-state diagram. Then, a new field for diagram label was created to implement what was previously referred to as the type. The markdown pages were edited to explain the difference between the type and the label and images of the new interface were provided.

Additional documentation was added to explain templates. When creating a diagram, the user can choose to have the diagram follow a specific template. Documentation in the "Diagram" section will include information on template options as well as an entirely new section explaining how to create new templates.

The subsection "Types of Diagrams" was reworded to instead explain diagram labels and their significance. Instead of having a generic "Other" diagram label, you can type the name of the label you wish your diagram to be part of, and the left navigation frame will update itself to include the category.

6.1.3 States

The new UI caused adjustments to the "State" section of the documentation, including an easier way to create a state. When right-clicking in the diagram, and after selecting the option "New State," instead of a small form just asking for the name of the new state, the user interface will instead pull up the dialogue box for entering all the state details—including name, description, and type. New images are included, and text is updated to reflect this change.

The table listing the state's events has been removed. Since the option to leave a state when an event is triggered is shown on the event form, it does not need to be offered here on the state form as well. Some images and paragraphs in the markdown were removed to clarify these changes.

6.1.4 Events

Several changes to the events forms and methods require the documentation to be updated. Events can no longer be created by right-clicking on the events tab in the left navigation frame. Instead, events are created either by right-clicking the state in the diagram or from the drop-down menu. Then, when entering the event information, the user can select the option to exit the parent state after the event is triggered. (This is available only if the event is accessed from the diagram UI.)

There were several other notable changes to the documentation's "Event" section. Event types can now be changed, but the event data will be cleared with every change. The external event simulation has also changed to allow multiple external event types. Documentation was added to explain each external event type. An event type "distribution" was created, and when selected, the user will be given the option to further specify the type of distribution. The requested information will change depending on that answer.

6.1.5 Actions, Variables, and Other

There have been minor updates to several other pieces of the UI. The "External Simulation Message" action type has been modified, and markdown documentation reflects the changes. When the "Sim Action" property is set to "Open Sim," the model reference, config data, and max simulation runtime variables are no longer required. When the "Sim Action" is "Comp Modify," the variable value is no longer required.

Other small changes to the user interface include how variables must now be created using the right-side drop-down menu. When creating a logic gate, there is no longer a navigation window at the top right corner, and the zoom-in and zoom-out tools have been removed. To add a logic gate to a logic tree, you can drag the provided tool on top of the desired gate—there is no longer an option to click a small "+" icon. When adding basic events to a logic gate, basic events are being reworded as "Evaluation Nodes." Each of these small changes requires the documentation to be updated.

The final few changes that were necessary were in the "External Simulations" and "Icons" sections of the documentation. New external simulations must now be created using the right-side menu. There is no longer another option to create them. In the "Icons" documentation page, the images have been updated in this section to reflect and explain the new icons.

The new user interface has caused much of the markdown documentation to be updated. Almost every section had outdated images and information. Now, each section has been revised to reflect the current state of the application. With the new user interface design and up-to-date documentation, users can better navigate and utilize the EMRALD application.

6.2 EMRALD JSON Model Syntax Information

The user interface is the primary way to create and edit an EMRALD model. However, there are many instances where a user may wish to edit or view the JSON text version of the model. Previously there was no documentation on what fields and requirements; therefore, someone would have to look at examples or review the EMRALD code to understand how items were saved. The creation of the EMRALD JSON model schema that was used to simplify and standardize other areas of development, also provided an easy way to autogenerate much of the help documentation.

Using a tool called jsonschema2md [25], markdown documentation of the EMRALD model schema is generated. This documentation allows the user to click through the schema and see each of the various parts without having to dig through the code base to piece it together.

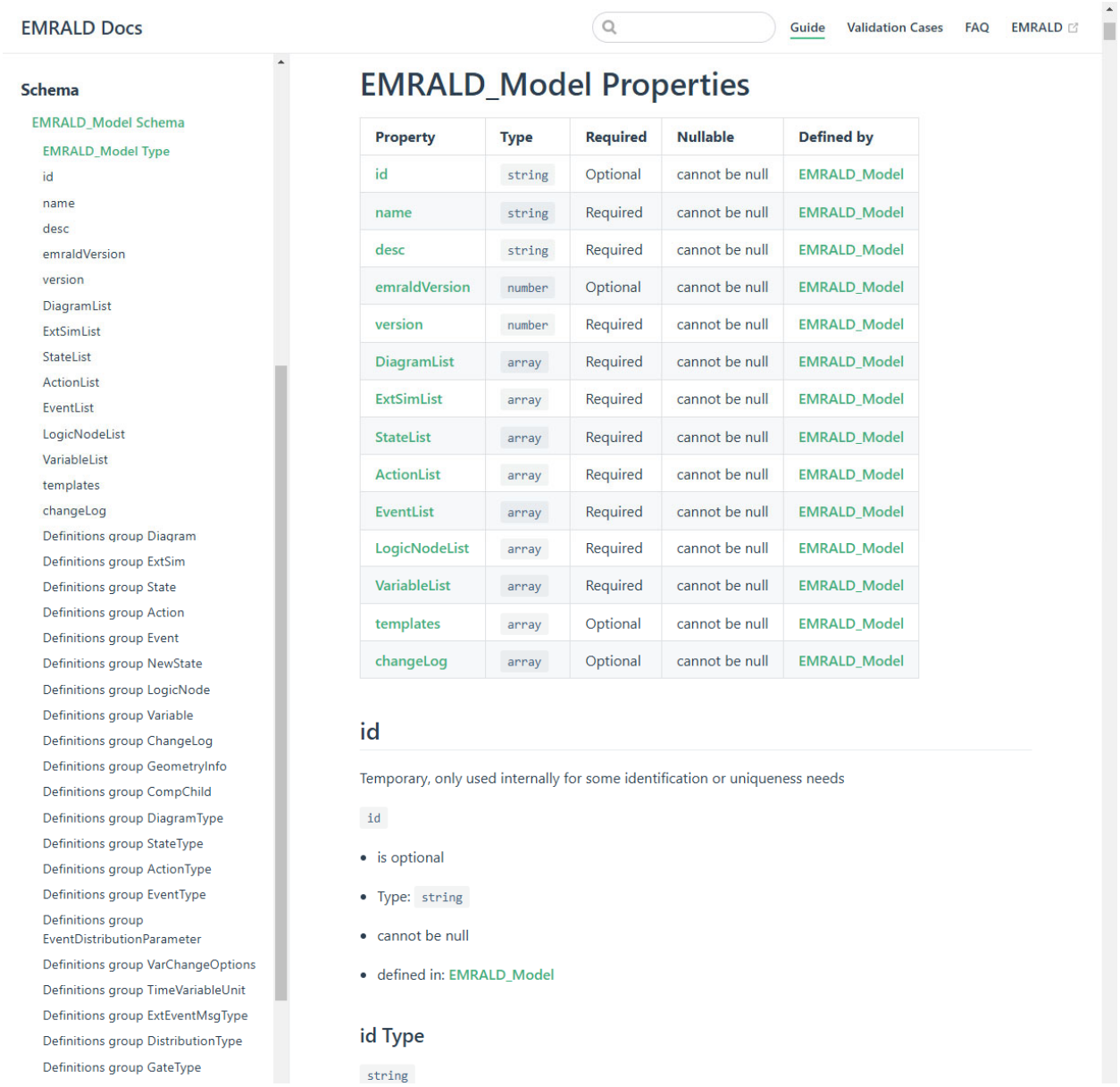


Figure 19. EMRALD model schema added to the documentation. Autogenerated from the JSON schema file.

7. FUTURE WORK

Most of functionality is in the initial Alpha release, but as a web application, EMRALD can have new releases that are seamless to the user. Two current features are still being worked on and will be in a future release. The first of these is the "Use Custom Application" for the Run Application action type shown in Figure 20. Previously, this had the option to run MAAP (Modular Accident Analysis Program) and had a user interface for that. This function is still being developed; however, this will not interfere with the operation of existing models, including those using a custom MAAP application. A custom application form simply offers an easy-to-use interface for generating a standard Run Application event. New models or modifying the MAAP parameters will be difficult until this is complete.

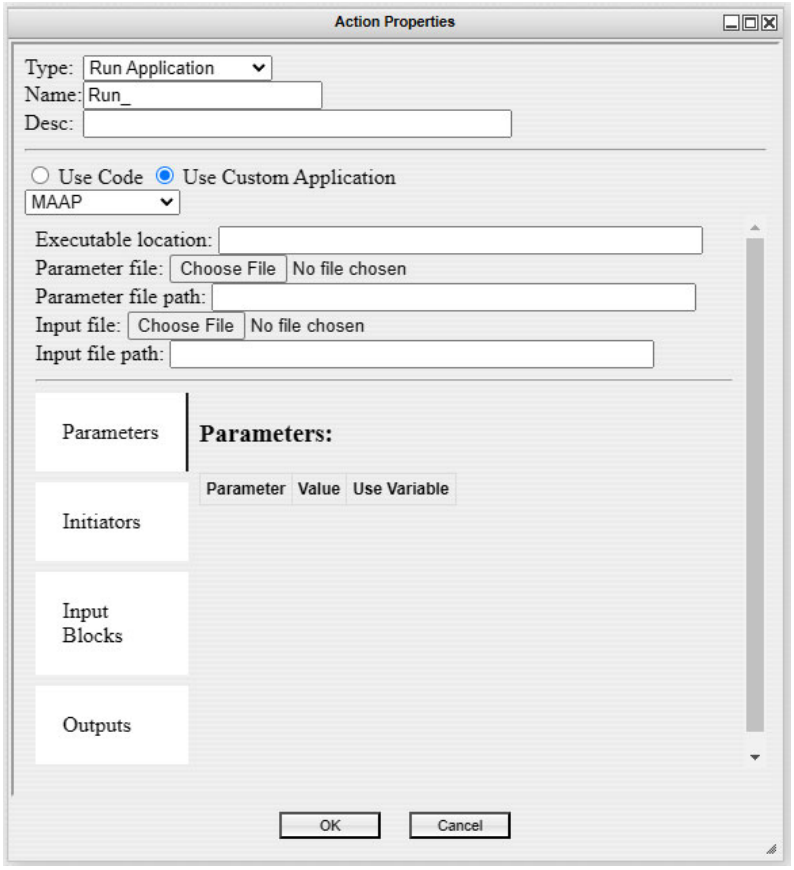


Figure 20. The previous custom application user interface for MAAP.

Also, the filtering option available in the older version as seen in Figure 21 is not complete. This option allowed the user to filter the action, event, variable, and state lists to only contain items that are used by the diagram specified in the filter. A future update will also include this capability while making it more intuitive to the user.

Figure 21. The filter feature of the old version of EMRALD.

Other features identified and that are easy to implement given the new user interface data structure will also be included in near-term releases, including a search and cross reference, where the user searches for items and then sees where that item is used.

8. EMRALD V3 Release

A build pipeline has been set up which pulls the code from the open-source Git repository and then compiles the website. This process allows INL staff to easily review any changes and then test them on a development site before being released to the main website. The new EMRALD website is available at www.emrald3app.inl.gov. Both the old version, located at www.emraldapp.inl.gov, and new version will be available until testing is complete and initial users have had the opportunity to provide feedback. After that, the standard or old website will be updated to the new version. As stated earlier, once upgraded to the 3.0 version, models will not be usable in the old UI or solve engine. It is anticipated that a full transition will be done by the end of 2024.

9. REFERENCES

1. Parisi, C., et al. 2016. Demonstration of External Hazards Analysis. Report No. INL/EXT-16-39353. Idaho Falls, ID: Idaho National Laboratory. https://lwrs.inl.gov/RiskInformed%20Safety%20Margin%20Characterization/Demonstration_of_External_Hazards_Analysis.pdf.

2. Coleman, J., et al. 2016. Multi-Hazard Advanced Seismic Probabilistic Risk Assessment Tools and Applications. Report No. INL/EXT-16-40055. Idaho Falls, ID: Idaho National Laboratory. https://lwrs.inl.gov/RiskInformed%20Safety%20Margin%20Characterization/Multi-Hazard_Advanced_Seismic_Probabilisitc_Risk_assessment_Tools_and_Applications.pdf.
3. Ma, Z., C. L. Smith, and S. R. Prescott. 2022. A Simulation-Based Dynamic Analysis Approach for Modeling Plant Response to Flooding Events. Report No. INL/EXT-17-40928, Rev. 1. Idaho Falls, ID: Idaho National Laboratory. https://lwrs.inl.gov/RiskInformed%20Safety%20Margin%20Characterization/Simulation-basedDynamicApproachFlooding_RISA.pdf.
4. Parisi, C., et al. 2017. Risk-Informed External Hazards Analysis for Seismic and Flooding Phenomena for a Generic PWR. Report No. INL/EXT-17-42666. Idaho Falls, ID: Idaho National Laboratory. https://www.osti.gov/servlets/purl/1376899.
5. Prescott, S., et al. 2023. Plant-Specific Model and Data Analysis using Dynamic Security Modeling and Simulation. Report No. INL/RPT-23-73490, Rev 1. https://lwrs.inl.gov/Physical%20Security/Plant-SpecificModelDataAnalysisDynamicSecurity.pdf.
6. Prescott, S., T. Wood, and M. Ziccarelli. 2022. Dynamic and Classical PRA Coupling using EMRALD and SAPHIRE. Report No. INL/RPT-22-70424. https://lwrs.inl.gov/RiskInformed%20Safety%20Margin%20Characterization/Dynamic_Classical_PRA.pdf.
7. Lew, R., et al. 2023. EMRALD-HUNTER: An Embedded Dynamic Human Reliability Analysis Module for Probabilistic Risk Assessment. Report No. INL/RPT-23-72783. https://lwrs.inl.gov/RiskInformed%20Safety%20Margin%20Characterization/EMRALD-HUNTER.pdf.
8. Office of Technology Transitions. 2020. “Department of Energy Announces $33 Million for 2020 Technology Commercialization Fund Projects.” June 11, 2020. https://www.energy.gov/technologytransitions/articles/department-energy-announces-33-million2020-technology.
9. Small Business Innovation Research. n.d. “Development of EMRALD Services in a Fully-Integrated RISMC Platform.” Accessed August 22, 2023. https://www.sbir.gov/node/1648965.
10. AngularJS. Accessed June 10, 2024. https://angularjs.org.
11. jgraph. “mxGraph.” Accessed June 10, 2024. https://github.com/jgraph/mxgraph-js.
12. Angular. "Angular Documentation." Accessed June 10, 2024. https://v17.angular.io/docs.
13. React. Accessed June 10, 2024. https://react.dev.
14. xyflow team. React Flow. Accessed June 10, 2024. https://reactflow.dev.
15. D3.js. Accessed June 10, 2024. https://d3js.org.
16. npm. “react-rnd.” Accessed June 10, 2024. https://www.npmjs.com/package/react-rnd.
17. React DnD. “Drag and Drop for React.” Accessed June 10, 2024. https://react-dnd.github.io/react-dnd/about.
18. MUI. “Material-UI.” Accessed June 10, 2024. https://mui.com/material-ui.
19. React Icons. Accessed June 10, 2024. https://react-icons.github.io/react-icons.
20. Day.js. Accessed June 10, 2024. https://day.js.org.
21. React. “useContext Hook.” Accessed June 10, 2024. https://react.dev/reference/react/useContext.

22. ClearScript. Accessed June 10, 2024. https://github.com/microsoft/ClearScript.
23. TypeScript. Accessed June 10, 2024. https://www.typescriptlang.org.
24. “JSONPath.” Wikipedia, The Free Encyclopedia. Last modified May 15, 2024. Accessed June 10, 2024. https://en.wikipedia.org/wiki/JSONPath.
25. Python Package Index. “jsonschema2md.” Accessed June 10, 2024. https://pypi.org/project/jsonschema2md.