Documentation for the Simulation Editor of the FRAMEwork System (FRAMES)

PNWD-3510

With Contributions From

B. L. Hoopes
M. A. Pelton
K. J. Castleton
G. M. Gelston
G. Whelan

Documentation for the Simulation Editor of the FRAMEwork System (FRAMES)

Documentation for the Simulation Editor of the FRAMEwork System (FRAMES)

Introduction

This documentation provides information on a component of Version 2.x of the FRAMEwork System (FRAMES), which is a software platform that allows for the linking of various modules into complete assessment systems (Whelan et al. 1997. PNNL-11748). Documentation includes requirements, design and specifications or formulations, and quality assurance and testing.

Portions of this documentation may have been previously issued in reports from the Pacific Northwest National Laboratory (PNNL), operated by Battelle for the U.S. Department of Energy. All PNNL reports are issued a tracking number. Numbers on the title page of this documentation indicate these previous reports.

This documentation can be used by software engineers and testers to ensure that each component functions properly. The information can also be used by analysts and managers to better understand the component's use within FRAMES.

Documentation for the Simulation Editor of the FRAMEwork System (FRAMES)

Requirements of the Simulation Editor

The FRAMEwork System (FRAMES) includes a suite of editors designed to manage, view, and set up the underlying infrastructure as well as run a simulation. The five editors, each with its own documentation, include the following:

- Conversion Editor, which manages measures and units used by dictionaries and datasets to automatically convert between different mediums
- Dictionary Editor, which creates and edits dictionaries used by FRAMES to categorize data
- Module Editor, which creates and manages the modules that manipulate data by specifying dictionaries
- Domain Editor, which organizes a palette of modules from which to select for a particular simulation
- Simulation Editor, which sets up and runs a particular simulation.

This section summarizes requirements for the Simulation Editor. Those interested in setting up a risk scenario and analyzing data should refer to the start up documentation for additional information. Those interested in importing a module into FRAMES should refer to the importation documentation for additional information.

The Simulation Editor has the following general requirements:

1. Allow the user to create, access, and save simulation (*.sim) files.
2. Allow the user to pictorially develop the conceptual site model (CSM) for the scenario.
3. Provide online help on how the Simulation Editor operates.
4. Allow the user to select from icons representing the available model types (e.g., constituents, source, vadose, aquifer, surface water, air, overland, intake, exposure, impacts, and sensitivity/uncertainty) to develop the conceptual picture of the problem.
5. Provide the user with a signal light indicating the status of each module associated with the scenario.
6. Provide data viewers to allow users to view text and graphical information produced by any module that adheres to FRAMES data file specifications.
7. Provide error messages or prevent the user from linking modules inappropriately or when module selections occur.
8. Display module-provided error messages to the user when operating modules inappropriately.
9. Provide a drag and drop system to add or remove connections between the modules in the diagram.
10. Follow module descriptions to allow only appropriate linkages between modules to develop a sull simulation.
11. Allow the user to delete a module icon by selecting "Delete" from its right-click context menu.
12. Provide a "Go" button to run all modules associated with a simulation.
13. Allow the user to customize the description of the simulation by changing the label for each module.
14. Allow the user to close a simulation file with or without saving changes.
15. Allow the user to print the CSM, module descriptions, and user inputs.
16. Have the look and feel of Windows.
17. Display the available models as icons organized by class and group (e.g., module type) within a domain as described in each module description file. Allow the user to change icons.
18. Allow the user to define the user interface font, colors, and logo.
19. Read a set of domains from the Startup.ini file and display it to the user in the module pallette.
20. Allow for execution of the simulation as a FRAMES component.
21. Implement the ability to lock the models, the connections, or both.
22. Allow the user the ability to add tiered icons to the icon pallette. Perform some testing and update any documentation associated with this updated functionality.
23. Allow the user to enter some information about a model that will easily be viewable in the user's work space. Update the User Interface so the user has the option of creating a sticky note for each module. The sticky note will provide the user with the ability to enter desired text and document a personal note.

Title Page
Legal Notice
Table of Contents
Introduction
Requirements
Design
Quality Assurance and Testing

Design of the Simulation Editor

The FRAMEwork System (FRAMES) includes a suite of editors designed to manage, view, and set up its underlying infrastructure on a single system as well as run a simulation. A full description of the design parameters can be found in Whelan et al. 2002, PNWD-3145.

This section describes the design of the Simulation Editor. Related design information on the other editors can be found in

- Design of the Conversion Editor
- Design of the Dictionary Editor
- Design of the Module Editor
- Design of the Domain Editor.

Simulation Editor General Design

The design of the Simulation Editor is simple but powerful. The user interface design was derived from two previous software systems. One was STELLA, a simulation software package that allows the user to diagram the relationships between variables using equations. The variables were represented by boxes. FRAMES took this concept of representing simple variables and expanded it to allow the representation of complex models that produce data which are consumed by other modules. Instead of simple equations defining the relationships, complex solutions to a complex set of equations can be represented.

The other software package that inspired the Simulation Editor was WebCoros. FRAMES took the WebCoros concept of displaying the status of each module to communicate to the user that computations were progressing.

The FRAMES 2.0 Simulation Editor was also designed to generalize the underlying functionality so that the conceptual site model could be used on any software that had a non-cyclic flow to the calculations. For example, FRAMES 2.0 modules can be built to facilitate any process that can be drawn as a diagram of information flow through a system with no cycles. A cycle in this diagram would be a set of objects and the connections between them such that starting at one object allows you to trace back to that object following the flow arrows. Mathematicians call these diagrams Directed Acyclic Graphs (DAGs). FRAMES was designed around DAGs because a DAG has a clear starting point and a clear ending point and can be collapsed into a single serial execution of the connected objects. In a broad sense, what is stored in a FRAMES 2.0 simulation file is the DAG of the modules, models, and schemes the user has chosen. For each object (represented by an icon and a name) the following information is stored:

- Chosen Module: which module to use for the simulation as chosen by the user.
- Module Instance Label: a more intuitive name given by the user.
- Instance Identifier: a unique name given to a module by the Simulation Editor.
- Scheme: which connection configuration was chosen by the user.
- Consumers: list of modules that consume the results of the chosen module.
- Producers: list of modules that produce results consumed by the chosen module.

Additional detail on information storage can be found in the following subsections.

Another important addition to the FRAMES 2.0 Simulation Editor was the concept of schemes. A scheme is a way a model can connect to other models. For example, if an aquifer chemical transport model can consume a chemical concentration contour in some cases and a chemical flux into the aquifer in others, then two different schemes for the same model need to be defined. In the first case, the module developer would specify that the model consumes the "ChemicalConcentrationContour" dataset and in the other it consumes the "ChemicalFlux" dataset. This scheme listing mechanism allows the Simulation Editor to help the user choose modules that can fit the defined producer/consumer relationships. The table below provides an example scheme for three models.

Scheme Example

Model	Scheme	Produces	Consumes
X	Static Boundary	A	
X	Typical	A	B
Y	Static Boundary	B	
Y	Typical	B	B
Z	Typical	C	A

The set of schemes and models shown in the table allow for a large number of diagrams. A simple notation provides a text representation of diagrams:

{Model}{Instance Number}-{Dataset Transferred}{Dataset Instance Number}>{Model}{Instanced Number}

For example, X1-A1>Z1 states that the first instance of X produced the first instance of A, which is consumed by the first instance of Z. A set of such connections represents a diagram:

Y1-B1>Y2
Y2-B2>X1
X1-A1>Z1

Connection 1 uses the "Static Boundary" scheme of model Y, whereas connection 2 uses the "Typical" scheme of model Y. This module set also implies that a connection like Z1-B1>Y1 should not be allowed because there is no scheme where Z produces dataset B.

Another important feature of the Simulation Editor is the visualization of results. Take the simple three connection graph as an example. It is very beneficial to the user if datasets B1, B2, A1, and C1 (produced by Z1) can be visually inspected. Viewers for datasets are just like those for modules, with a couple of exceptions. First, the viewers do not need to be a permanent part of the diagram as long as they are convenient to use. Second, the viewers do not typically produce a dataset that is recognized by other software. For example, a viewer may invoke Microsoft's Excel and produce a chart in a .xls file. The .xls file is probably not going to be consumed by another module. If a viewer does produce results that can be consumed by another module, that viewer can be labeled and used as a module.

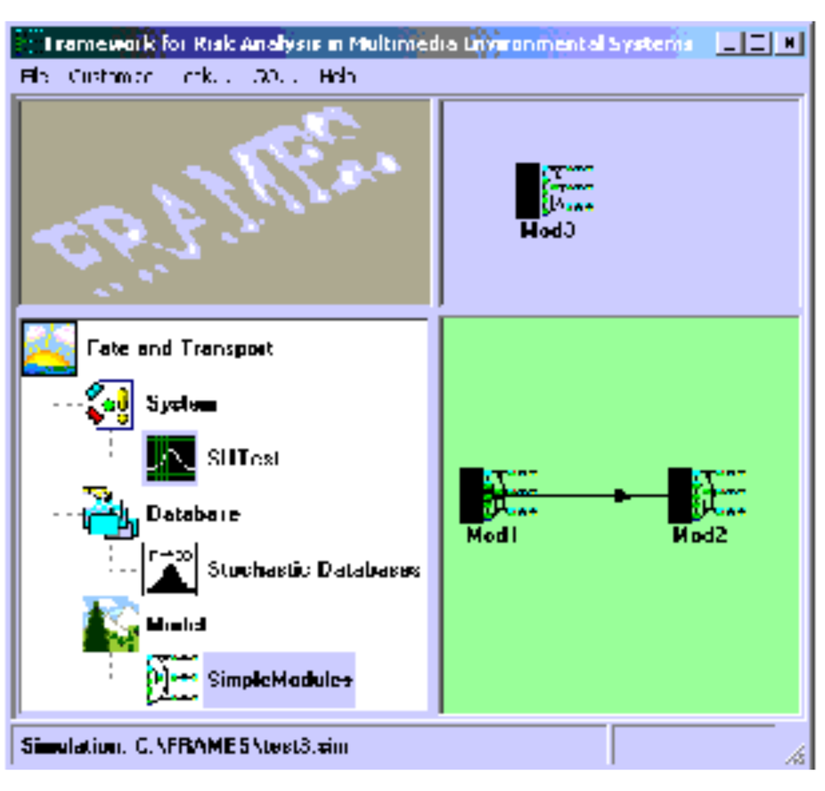
Simulation Editor Input Design

The Simulation Editor is meant to be an intuitive interface for the interaction with the CSM diagram. The user can add modules, connect modules, populate input for modules, and run simulations. What modules are available are defined by the Framework Development Editor (FDE). The FDE is the tool that configures the domain of simulations that the Simulation Editor will allow the user to create. Another way to think about it is the CSM workshop could be used for simulations of chemical movement in the environment or for gravitational forces between bodies in space. The user would not be helped if all the modules for the gravity simulator and chemical transport where lumped together. The two problems can both be modeled in simulations but in different domains. The FDE facilitates the creation and maintenance of those domains. The Simulation Editor uses the currently selected domain to aid the user in diagram development and simulation.

In FRAMES 1.X it was often required that the user connect some module to nearly all other modules. A good example was the Chemical Property Database. This database provided a dataset that contained all chemical properties needed for the models. The problem was that every model in the diagram needed to consume that dataset, and therefore the user was required to connect that single module to all other modules. In FRAMES 2.0, the Simulation Editor includes an additional panel to the top right of the screen. This panel is a global module area. Any module dropped in this area has its output made available to all other modules in the diagram. Thus, the global module panel provides a shortcut to connecting a single module to all other modules. This feature can be overridden by simply adding another module to the main diagram and connecting to other models specifically.

To connect modules, the user simply holds down the "shift" key while using the mouse to drag from one module to another. To disconnect, the user performs the same operation again. Modules are added by highlighting a panel to select it (highlighted in green) and double clicking the selected icon in the module pallette. Modules can be moved to a specific location in the selected area by dragging them with the mouse. Note that there is no meaning to the location of the modules except that it may convey flow better to start at the top left of the area and continue toward the right and down. However, the Simulation Editor does not require or enforce this convention.

The figure below shows the FRAMES 2.0 user interface. This is the starting point for drawing a CSM. The four areas (Logo, Module Pallette, Global Module Area, and the Scenario Area) can be adjusted by dragging the bar that separates them. The domain in this example is simple: It contains SimpleModules, Stochastic Databases, and the SUTest module. In the Scenario Area, "Mod1" produces results for "Mod2" and in the "Global Module Area" Mod3 produces results that can be used by any module in the "Scenario Area".



FRAMES 2.0 User Interface

A right-click on any module in the Scenario Area or the Global Area will pop up a menu of actions that can be taken for that module. The figure below shows an example of this menu. The options are identical to the FRAMES 1.3 User Interface except that the new General Info screen has scheme choice added. A description of the General Info, User Input, Run Model, Delete, View/Print User Input, and View/Print Model Output can be found in the FRAMES 1.3 User Interface document.

Popup Menu

The Note option is new. This option allows the user to associate a note with a particular instance of a module. When this option is chosen, a dialog with an area for entering text is provided for user input.

Specifications for Simulation Files

The *.sim file is designed to follow the FRAMES 2.0 dataset dictionary specification. The entries for the .sim file are shown in the following figure.

Variable Count	Dictionary Description	Dictionary	Privilege	Version	Updated						
26	Simulation Script Definition	Simulation	1	3	0						
Name	Description	Dimension	DataType	PrimaryKey	Scalar	Minimum	Maximum	Stochastic	Index 1	Index 2	Index 3
ConDicName	Consumer dictionary name	3	STRING	FALSE	FALSE	1	32	FALSE	Simulation.ModID	Simulation.ConModID	
ConModID	Module's consumer module ID list	2	STRING	FALSE	FALSE	1	32	FALSE	Simulation.ModID		
ConSetName	Consumer data set name	3	STRING	FALSE	TRUE	1	32	FALSE	Simulation.ModID	Simulation.ConModID	Simulation.ConDicName
LayoutProperties	Layout properties for sizing panels	1	INTEGER	FALSE	FALSE	0	0	FALSE			
LockConnections	Lock connections	0	LOGICAL	FALSE	TRUE	0	0	FALSE			
LockModules	Lock modules	0	LOGICAL	FALSE	TRUE	0	0	FALSE			
LockPassword	Lock password	0	STRING	FALSE	TRUE	0	32	FALSE			
ModClass	Module class name	1	STRING	FALSE	TRUE	1	32	FALSE	Simulation.ModID		
ModDomain	Module domain name	1	STRING	FALSE	TRUE	1	32	FALSE	Simulation.ModID		
ModGlobal	Module is global or local	1	LOGICAL	FALSE	TRUE	0	0	FALSE	Simulation.ModID		
ModGroup	Module group name	1	STRING	FALSE	TRUE	1	32	FALSE	Simulation.ModID		
ModIcon	Module icon	1	STRING	FALSE	TRUE	0	512	FALSE	Simulation.ModID		
ModID	Module ID	1	STRING	FALSE	FALSE	1	32	FALSE			
ModLabel	Module user label	1	STRING	FALSE	TRUE	0	32	FALSE	Simulation.ModID		
ModName	Module name	1	STRING	FALSE	TRUE	0	32	FALSE	Simulation.ModID		
ModNote	Module user note	1	STRING	FALSE	TRUE	0	1024	FALSE	Simulation.ModID		
ModPosX	X screen coordinate for module	1	INTEGER	FALSE	TRUE	0	0	FALSE	Simulation.ModID		
ModPosY	Y screen coordinate for module	1	INTEGER	FALSE	TRUE	0	0	FALSE	Simulation.ModID		
ModScheme	Module scheme	1	STRING	FALSE	TRUE	0	32	FALSE	Simulation.ModID		
ModScope	Module scope	1	INTEGER	FALSE	TRUE	0	1	FALSE	Simulation.ModID		
ModState	Module state	1	INTEGER	FALSE	TRUE	0	3	FALSE	Simulation.ModID		
ModSubGroup	Module subgroup name	1	STRING	FALSE	TRUE	0	32	FALSE	Simulation.ModID		
ProDicName	Producer dictionary name	3	STRING	FALSE	FALSE	1	32	FALSE	Simulation.ModID	Simulation.ProModID	
ProModID	Module's producer module ID list	2	STRING	FALSE	FALSE	1	32	FALSE	Simulation.ModID		
ProSetName	Producer data set name	3	STRING	FALSE	TRUE	1	32	FALSE	Simulation.ModID	Simulation.ProModID	Simulation.ProDicName
SDENote	Simulation comment	0	STRING	FALSE	TRUE	0	1024	FALSE			

Dictionary Entries for the Simulation File

Several items in this file may require description:

- Lock parameters are used to lock different pieces of a simulation. Currently this functionality is not implemented but the variables are there for future use.
- Layout properties controls sizing of panels.
- SDENote is a string that stores a note about the simulation.

The groups of most interest to the Simulation Editor are the parameters starting with Con, Pro, and Mod. The Con parameters define the consumer relationships for a particular module. ConModId is the list of identifiers for modules from which a particular module can consume results. ConDicName is the name of the dictionary associated with that consumption. ConSetName is the name of the dataset associated with that consumption. The Pro parameters are analogous values for items related to producing results.

The Mod parameters can be divided into user interface parameters and module classification parameters. ModPosX, ModPosY, ModLabel, ModIcon, ModNote, ModGlobal, and ModState represent information the user interface needs to render the module on the screen. Most of the other parameters (ModDomain, ModClass, ModGroup, ModSubGroup, ModName, ModScheme, and ModScope) define placement of a particular module in the hierarchy of modules in the current domain. The most important parameter in the .sim file is ModId. This parameter, together with the Con parameters and the Pro parameters, is used to connect modules. ModId is also used to bind the instance of a module in the hierarchy of the modeling domain.

Specifications for the Startup File

The Startup file (.ini file) also follows the FRAMES 2.0 dataset dictionary specification. The following figure shows the entries for the .ini file. Most of the parameters in this dictionary give the FRAMES 2.0 User Interface flexibility in font, presentation of a logo, and color. These parameters are important but mainly influence the way the Simulation Editor draws the CSM or configures the screen.

Variable Count	Dictionary Description	Dictionary	Privilege	Updated						
43	Start Up Settings	StartUp	0	0						
Name	Description	Dimension	DataType	Scalar	Minimum	Maximum	Index 1	Index 2	Index 3	Index 4
AppPath	Path to frames application	0	STRING	TRUE	0	512				
ClassIcons	Class icon path\filename	2	STRING	TRUE	0	512	StartUp.DomainNames	StartUp.ClassNames		
ClassNames	Class names	2	STRING	FALSE	0	32	StartUp.DomainNames			
DataBackColor	Background color of database connection	0	INTEGER	TRUE	0	0				
DataForeColor	Foreground color of database connection	0	INTEGER	TRUE	0	0				
DataVisible	Database connection visible flag	0	LOGICAL	TRUE	0	0				
Dictionaries	Path\filename of included dictionaries	1	STRING	FALSE	0	512				
DictionaryDir	Destination directory for dictionary download	0	STRING	TRUE	0	80				
DomainIcons	Domain icon path\filename	1	STRING	TRUE	0	512	StartUp.DomainNames			
DomainNames	Domain names	1	STRING	FALSE	0	32				
FontBold	Font bold	0	LOGICAL	TRUE	0	0				
FontColor	Font color	0	INTEGER	TRUE	0	0				
FontItalic	Font italic	0	LOGICAL	TRUE	0	0				
FontName	Font name	0	STRING	TRUE	0	512				
FontSize	Font size	0	FLOAT	TRUE	8	24				
GroupIcons	Group icon path\filename	3	STRING	TRUE	0	512	StartUp.DomainNames	StartUp.ClassNames	StartUp.GroupNames	
GroupModules	Group module names	4	STRING	FALSE	0	32	StartUp.DomainNames	StartUp.ClassNames	StartUp.GroupNames	
GroupNames	Group names	3	STRING	FALSE	0	32	StartUp.DomainNames	StartUp.ClassNames		
LinkageServerURL	location of linkage server	0	STRING	TRUE	0	0				
LogoFile	Path\filename of logo image	0	STRING	TRUE	0	512				
LogoVisible	Logo image visible flag	0	LOGICAL	TRUE	0	0				
ModBackColor	Background color of model connection	0	INTEGER	TRUE	0	0				
ModForeColor	Foreground color of model connection	0	INTEGER	TRUE	0	0				
ModIdVisible	Module Id visible flag	0	LOGICAL	TRUE	0	0				
ModuleDir	Destination directory for module description downloads	0	STRING	TRUE	0	80				
Modules	Path\filename of included modules	1	STRING	FALSE	0	512				
ModVisible	Model connection visible flag	0	LOGICAL	TRUE	0	0				
NoticeVisible	Notice visible flag	0	LOGICAL	TRUE	0	0				
RecentFiles	Recent simulation files	1	STRING	FALSE	0	512				
SubGrpIcons	Sub-group icon path\filename	4	STRING	TRUE	0	512	StartUp.DomainNames	StartUp.ClassNames	StartUp.GroupNames	StartUp.SubGrpNames
SubGrpModules	Sub-group module names	5	STRING	FALSE	0	512	StartUp.DomainNames	StartUp.ClassNames	StartUp.GroupNames	StartUp.SubGrpNames
SubGrpNames	Sub-group names	4	STRING	FALSE	0	32	StartUp.DomainNames	StartUp.ClassNames	StartUp.GroupNames	
SysBackColor	Background color of system connection	0	INTEGER	TRUE	0	0				
SysForeColor	Foreground color of system connection	0	INTEGER	TRUE	0	0				
SystemUpdate	Internal flag tracking if a module has been updated	0	LOGICAL	TRUE	0	1				
SystemVersion		0	FLOAT	TRUE	0	200000				
SysVisible	System connection visible flag	0	LOGICAL	TRUE	0	0				
WindowHeight	Height of the window	0	INTEGER	TRUE	0	0				
WindowPosX	X Screen position of window	0	INTEGER	TRUE	0	0				
WindowPosY	Y Screen position of window	0	INTEGER	TRUE	0	0				
WindowWidth	Width of the window	0	INTEGER	TRUE	0	0				
WorkBackColor	Background color of workspace	0	INTEGER	TRUE	0	0				
WorkForeColor	Foreground color of workspace	0	INTEGER	TRUE	0	0				

Dictionary Entries for the StartUp.ini File

The parameters that actually change the capabilities of the Simulation Editor are the parameters that begin with Startup. Other important parameters include DomainNames as index 1, and DomainNames because this set of parameters forms a hierarchy of the modeling domain that the Simulation Editor will present to the user. For example, a domain to model chemical release, transport, and impacts would probably be separate from a domain to model traffic in a transportation system. The hierarchy starts with DomainNames, a list of strings that represents the different domains available. Each DomainName can have a unique icon. Within domains are classes. ClassName is a list of classes within a domain. Each class has an icon and a set of groups associated with it and subgroups within that. GroupNames and SubGroupNames each can have a unique icon.

Only at the level of GroupNames and SubGroupNames can modules be assigned. This feature has the effect of not letting the user drop a module icon for the domain or class. They can, however, drop a module icon for the group or subgroup if a module has been defined.

This organization of modules allows a framework developer to create and organize a set of tools for use with FRAMES 2.0 that will facilitate proper use of the set of modules. Different domains can be recognized and work separately or together with less confusion about available tools.

Documentation for the Simulation Editor of the FRAMEwork System (FRAMES)

Quality Assurance and Testing of the Simulation Editor

FRAMES was developed under a quality assurance (QA) program that looked at the software life cycle: requirements analysis, design, programming, modification, testing, and implementation. A general description of this program is contained in the quality assurance and testing section of the FRAMES system documentation.

Part of the QA program involves testing each component to ensure that it satisfies its requirements. The requirements section of this documentation provides a list of requirements for the Simulation Editor. A test plan was developed with test cases that addressed these requirements. The table below shows how these requirements were addressed in testing.

Testing Matrix for the Simulation Editor Testing

Requirement	Test Case							
---	01	02	03	04	05	06	07	08
1	X	X						
2	X							
3	X							
4	X							
5	X							
6	X							
7								X
8								X
9	X							
10	X							
11		X						
12			X					
13	X							
14		X						
15				X				
16	X							
17					X			
18						X		
19					X			
20							X	
21								X
22	X							
23	X							

An Approach to Ensuring Quality in Environmental Software (Gelston et al. 1998. PNNL-11880)

- *Test Plan for the Simulation Editor*
- Status Report for Simulation Editor Testing
- Test bed for Simulation Editor testing.