



UNF-ST&DARDS Documentation

Release 3.0

**Kaushik Banerjee,
Paul Miller, and Rob Lefebvre**

March 18, 2016

CONTENTS

1	Overview	1
1.1	Introduction	1
1.1.1	Development team	2
1.2	Installation and Verification	2
1.2.1	Bundle Contents	2
1.2.2	System Requirements	3
1.2.3	Installing UNF-ST&DARDS	3
1.2.4	Verifying INSTALLATION	4
1.2.5	Activating Geographic Information System (GIS)	13
1.2.6	Question	17
2	Architecture and Capabilities	18
2.1	System Architecture	18
2.2	Unified Database	20
2.2.1	Unified Database Content	20
2.2.2	Unified Database Design	22
2.2.3	Unified Database Software Development Kit (SDK)	23
2.3	Templates	23
2.3.1	Template Repository	23
2.4	Analysis Sequence	25
2.4.1	Fuel Depletion and Decay	26
2.4.2	Criticality Safety	26
2.4.3	Thermal Hydraulics	28
2.4.4	Shielding	28
2.4.5	Containment Analysis	28
3	User Guide	29
3.1	Graphical User Interface (GUI)	29
3.1.1	Analysis Tab	29
3.1.2	Pad Maps	37
3.1.3	Analysis Model Geometry Viewer	54
3.1.4	Plot Functionality	57
3.1.5	SQL Viewer	58

3.1.6	Reports	59
4	Developer Guide	61
4.1	Development Environment	61
4.1.1	Coding Standards	62
4.1.2	Test Infrastructure	62
4.1.3	Setting up UNF-ST&DARDS Development Environment	62
4.2	TemplateEngine	65
4.2.1	TemplateEngine Requirements	66
4.2.2	JSON Parameter Set	66
4.2.3	TemplateEngine Features	67
4.2.4	Attribute and Expression Format	75
4.2.5	Functions	75
Index		79

OVERVIEW

1.1 Introduction

This technical product is a user guide of the Used Nuclear Fuel-Storage, Transportation & Disposal Analysis Resource and Data System (UNF-ST&DARDS). The objective of this manual is to help users (1) understand the various analysis processes within UNF-ST&DARDS and (2) execute various nuclear safety analyses using the graphical user interface (GUI). This manual will be updated periodically as UNF-ST&DARDS is updated. Questions or comments regarding UNF-ST&DARDS should be submitted to <UNFHelp@ornl.gov>.

UNF-ST&DARDS is being developed as a foundational resource for the DOE-NE to streamline computational analysis capabilities for characterizing the input for the overall waste management system. UNF-ST&DARDS provides a controlled, unified domestic spent nuclear fuel (SNF) system database—the Unified Database—which is integrated with nuclear analysis capabilities to support DOE waste management and fuel cycle-related objectives. The Unified Database is integrated with the analysis tools. This powerful feature of UNF-ST&DARDS provides a unique platform for performing assembly-specific and cask-specific assessments of nuclear safety based on actual characteristics of the as-loaded SNF (e.g., fuel assembly burnup, enrichment, and decay time). The cask-specific safety evaluations allow quantification of realistic safety margins associated with actual fuel loading compared with the regulatory licensing limits, and they enable a better understanding of the implications of and planning for addressing uncertainties related to aging that might arise from extended storage and subsequent transportation.

In this manual, the standard installation instructions of UNF-ST&DARDS are provided in Sect. 1 (Overview). Section 2 (Architecture and Capabilities) presents the software architecture including analysis sequences of UNF-ST&DARDS. Section 3 (User Guide) presents the GUI of UNF-ST&DARDS. Finally, Sect. 4 (Developer Guide) provides guidance to the developers including various features of the TemplateEngine.

1.1.1 Development team

The core UNF-ST&DARDS development team consists of the following ORNL scientists (listed alphabetically):

- Kaushik Banerjee <banerjeek@ornl.gov>
- Justin Clarity <clarityjb@ornl.gov>
- Rob Lefebvre <lefebvrera@ornl.gov>
- Henrik Liljenfeldt <liljenfeldth@ornl.gov>
- Paul Miller <millerpp@ornl.gov>
- Josh Peterson <peterstonjl@ornl.gov>
- Georgeta Radulescu <radulescug@ornl.gov>
- Kevin Robb <robbkr@ornl.gov>
- John Scaglione <scaglionejm@ornl.gov>
- Bret van den Akker <bpvda@ornl.gov>

General questions about UNF-ST&DARDS software and methods should be directed to the [email list](#).

1.2 Installation and Verification

The UNF-ST&DARDS installation bundled is a beta version with Unified Database access, and generated assembly type depletion libraries, and canister thermal axial heat and 3d dose maps.

Upon completion of this section you will have a functioning UNF-ST&DARDS installed.

1.2.1 Bundle Contents

This UNF-ST&DARDS installation bundle should include the following files:

UNF-STANDARDS-INSTALL-XXXXXXXXX.zip, and Manual.pdf (this document)

Where the XXXXXX file suffix indicates the packing date.

E.g., UNF-STANDARDS-INSTALL-2015-09-29_144036.jar indicates the bundle was created on September 29, 2015, at 144036 or 2:40:36PM.

1.2.2 System Requirements

UNF-ST&DARDS has the following minimum requirements:

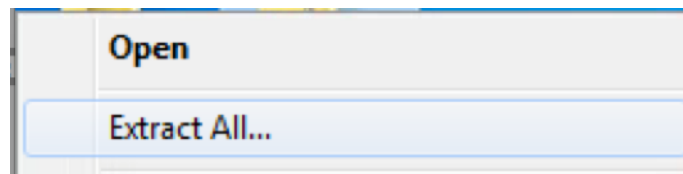
- 64bit Windows 7
- Java 8 (available free at <https://java.com/download>)
- 4 GB Hard Disk Space
- 2 GB RAM

1.2.3 Installing UNF-ST&DARDS

If the UNF-ST&DARDS installation bundle USB is not already inserted into your USB drive, do so now.

Copy the UNF-STANDARDS-INSTALL-XXXXXXXXX.zip to the desired location of install. For example purposes, this section will use installation path of **C:\Users\user\Desktop\UNF-STANDARDS** where **user** is your specific user name.

Right click the UNF-STANDARDS-INSTALL-XXXXXXXXX.zip file and select the ‘Extract all...’ menu item.

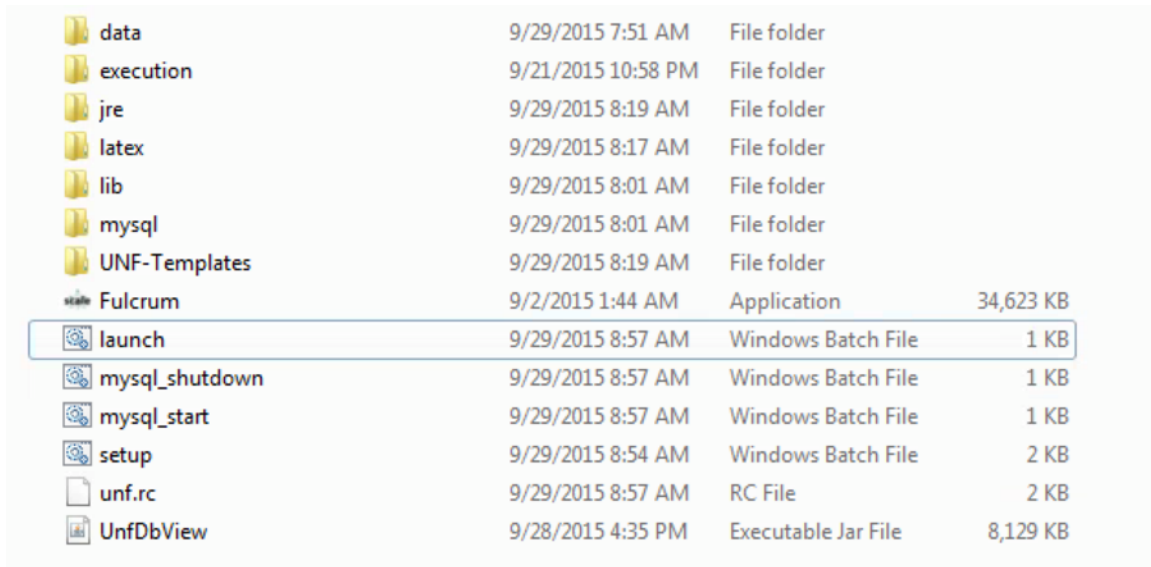


After several minutes the installation will complete as indicated after which double left clicking the contained setup.bat script will produce installation specific configuration files needed for launching UNF-ST&DARDS.

Name	Date modified	Type	Size
data	9/29/2015 7:51 AM	File folder	
execution	9/21/2015 10:58 PM	File folder	
jre	9/29/2015 8:19 AM	File folder	
latex	9/29/2015 8:17 AM	File folder	
lib	9/29/2015 8:01 AM	File folder	
mysql	9/29/2015 8:01 AM	File folder	
UNF-Templates	9/29/2015 8:19 AM	File folder	
Fulcrum	9/2/2015 1:44 AM	Application	34,623 KB
setup	9/28/2015 4:42 PM	Windows Batch File	2 KB
unf	9/28/2015 4:42 PM	RC File	2 KB
Unt	9/28/2015 4:35 PM	Executable Jar File	8,129 KB

Tooltip for 'unf':
 Type: Windows Batch File
 Size: 1.33 KB
 Date modified: 9/28/2015 4:42 PM

Once `setup.bat` has been executed, the `launch.bat` script will be available for execution. Double click the `launch.bat` to start UNF-ST&DARDS.

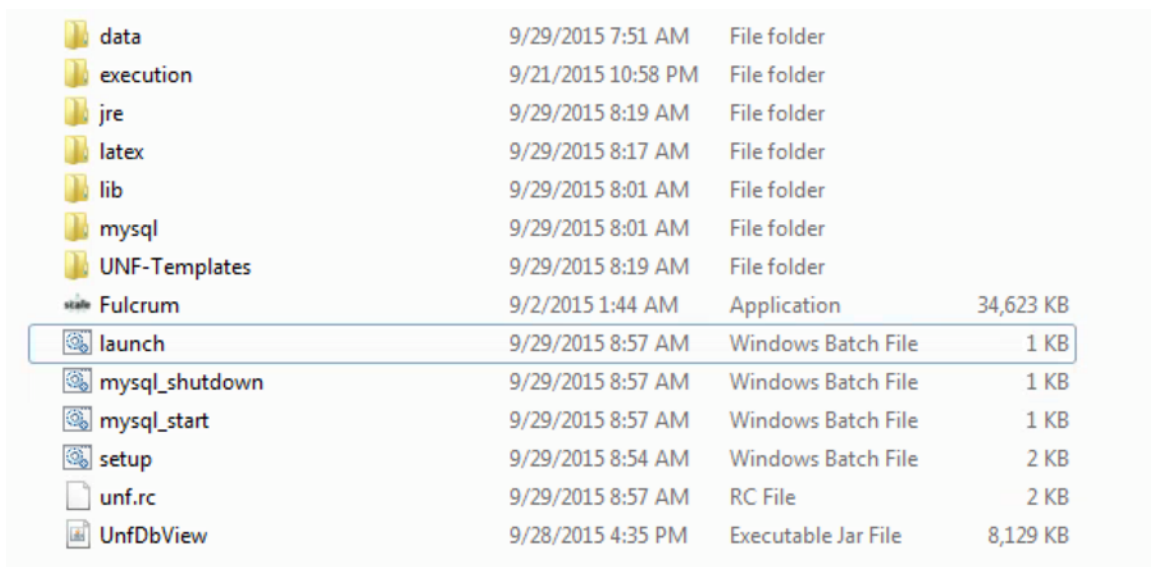


data	9/29/2015 7:51 AM	File folder	
execution	9/21/2015 10:58 PM	File folder	
jre	9/29/2015 8:19 AM	File folder	
latex	9/29/2015 8:17 AM	File folder	
lib	9/29/2015 8:01 AM	File folder	
mysql	9/29/2015 8:01 AM	File folder	
UNF-Templates	9/29/2015 8:19 AM	File folder	
Fulcrum	9/2/2015 1:44 AM	Application	34,623 KB
launch	9/29/2015 8:57 AM	Windows Batch File	1 KB
mysql_shutdown	9/29/2015 8:57 AM	Windows Batch File	1 KB
mysql_start	9/29/2015 8:57 AM	Windows Batch File	1 KB
setup	9/29/2015 8:54 AM	Windows Batch File	2 KB
unf.rc	9/29/2015 8:57 AM	RC File	2 KB
UnfDbView	9/28/2015 4:35 PM	Executable Jar File	8,129 KB

1.2.4 Verifying INSTALLATION

Upon completing INSTALLING UNF-ST&DARDS you can verify your installation.

Navigate to the location to which you installed UNF-ST&DARDS.



data	9/29/2015 7:51 AM	File folder	
execution	9/21/2015 10:58 PM	File folder	
jre	9/29/2015 8:19 AM	File folder	
latex	9/29/2015 8:17 AM	File folder	
lib	9/29/2015 8:01 AM	File folder	
mysql	9/29/2015 8:01 AM	File folder	
UNF-Templates	9/29/2015 8:19 AM	File folder	
Fulcrum	9/2/2015 1:44 AM	Application	34,623 KB
launch	9/29/2015 8:57 AM	Windows Batch File	1 KB
mysql_shutdown	9/29/2015 8:57 AM	Windows Batch File	1 KB
mysql_start	9/29/2015 8:57 AM	Windows Batch File	1 KB
setup	9/29/2015 8:54 AM	Windows Batch File	2 KB
unf.rc	9/29/2015 8:57 AM	RC File	2 KB
UnfDbView	9/28/2015 4:35 PM	Executable Jar File	8,129 KB

Double left-click **launch.bat** windows batch script file to launch a **cmd** console and UNF-ST&DARDS.

A **cmd** console will display the activity of UNF-ST&DARDS. Specifically, as UNF-ST&DARDS starts, it attempts to determine if the Unified Database and associated MySQL database

server are running and starts them as needed.

```

C:\Windows\system32\cmd.exe - java -jar -Xmx2048m -D.unf.rc=C:\Users\raq\Desktop\UNF-STANDARDS\unf.rc C:\Users\raq\Desktop\UNF-STANDAR...
C:\Users\raq\Desktop\UNF-STANDARDS>cd execution
C:\Users\raq\Desktop\UNF-STANDARDS\execution>cmd /k java -jar -Xmx2048m -D.unf.rc=C:\Users\raq\Desktop\UNF-STANDARDS\unf.rc C:\Users\raq\Desktop\UNF-STANDARDS\UNF-Db-Viewer\dist\UnfDbUview.jar
Java version: 1.7.0_65
Connecting to database...
[EL Info]: 2014-09-29 18:30:06.157--ServerSession(144134682)--EclipseLink, version: Eclipse Persistence Services - 2.3.2.v20111125-r10461
[EL Severe]: 2014-09-29 18:30:08.318--ServerSession(144134682)--Exception [EclipseLink-4002] (Eclipse Persistence Services - 2.3.2.v20111125-r10461): org.eclipse.persistence.exceptions.DatabaseException
Internal Exception: com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link failure

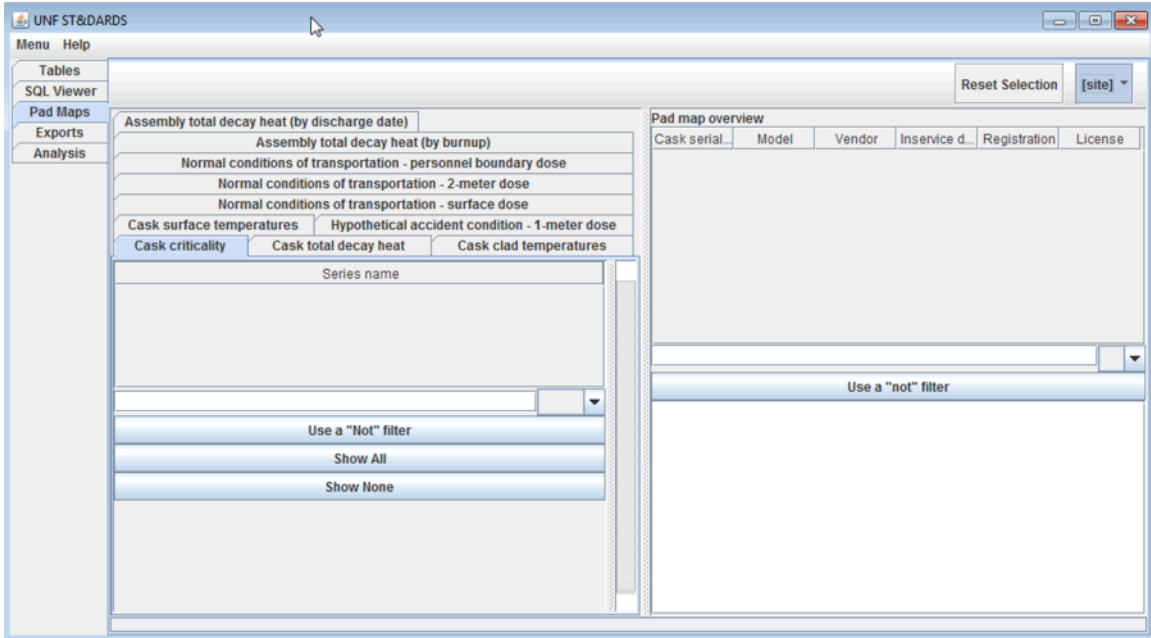
The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.
Error Code: 0
Attempting to start the database at C:\Users\raq\Desktop\UNF-STANDARDS\mysql\bin\mysqld.exe...
[EL Info]: 2014-09-29 18:30:10.332--ServerSession(144134682)--EclipseLink, version: Eclipse Persistence Services - 2.3.2.v20111125-r10461
[EL Info]: 2014-09-29 18:30:13.015--ServerSession(144134682)--file:C:/Users/raq/Desktop/UNF-STANDARDS/UNF-Db-Viewer/dist/11b/UNF_Db.jar_unf_db_modelIPU_url_jdbc:mysql://localhost:3306/unf_db_user=unf_model login successful
Info: The database version is 2014.09.25.
2014/09/29 18:30:15 setSelectedCanister null
Sep 29, 2014 6:30:19 PM view.AnalysisPage initVariables
INFO: duration: 2072ms
Sep 29, 2014 6:30:20 PM view.AnalysisPage initVariables
INFO: duration: 770ms
Starting the JobServer as a local ShepherdServer.
ShepherdServer: listening on port 6700 with 1 workers standing by...
Command port 6745.
Thread Thread[Thread-5,6,main] starting...

```

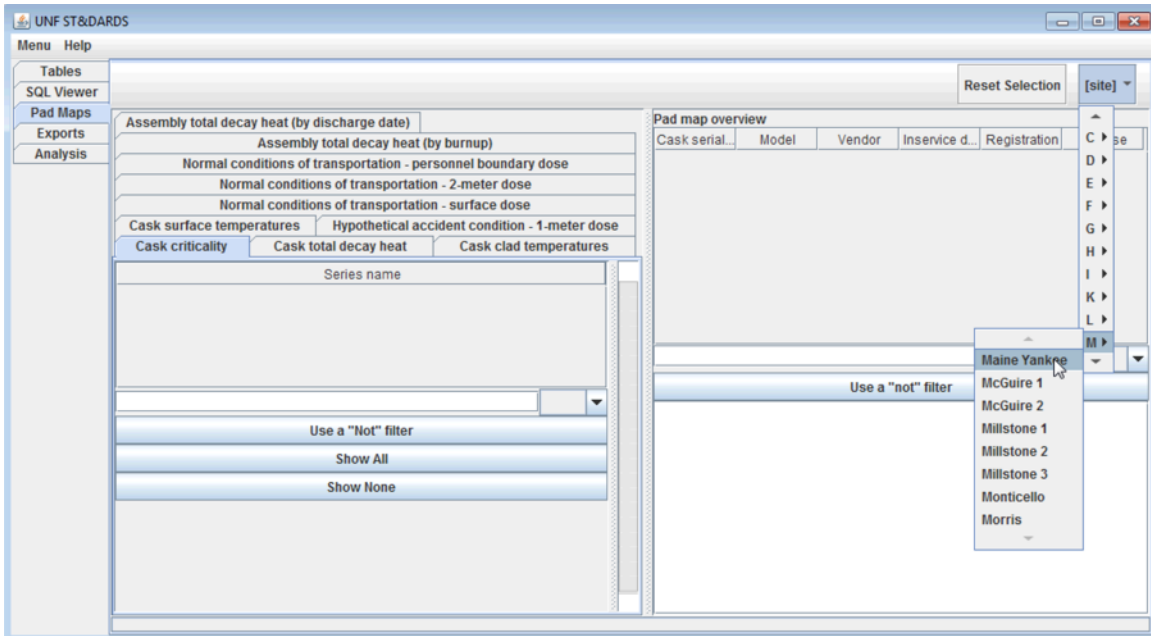
UNF-ST&DARDS will start on the Analysis Page.

Row	Reactor	Assembly	Initial Uranium (kg)	Initial Enrichment (%)	Max Burnup (Mwd/)	Db Id	Run
1	Arkansas Nuclear...	NJ00GW	464.537	2.652	29,714	168505	✓
2	Arkansas Nuclear...	NJ00GX	464.323	2.652	27,218	168506	✓
3	Arkansas Nuclear...	NJ00GY	464.165	2.652	25,489	168507	✓
4	Arkansas Nuclear...	NJ00GZ	464.127	2.652	24,361	168508	✓

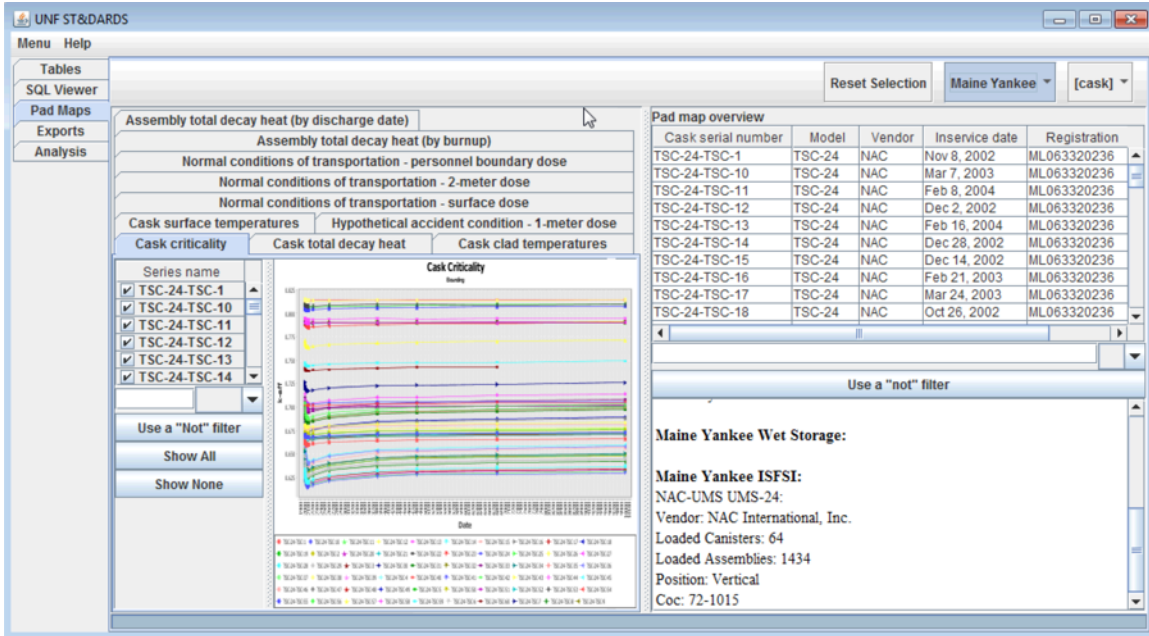
To verify integration click the Pad Maps tab.



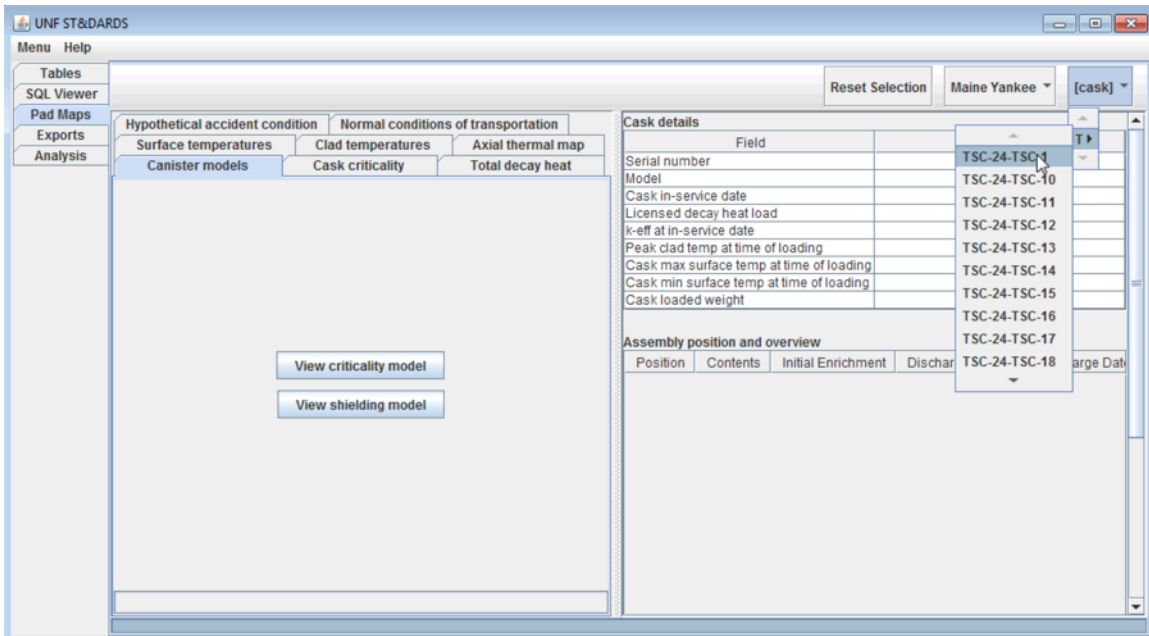
Select the Maine Yankee site from the [site] drop.



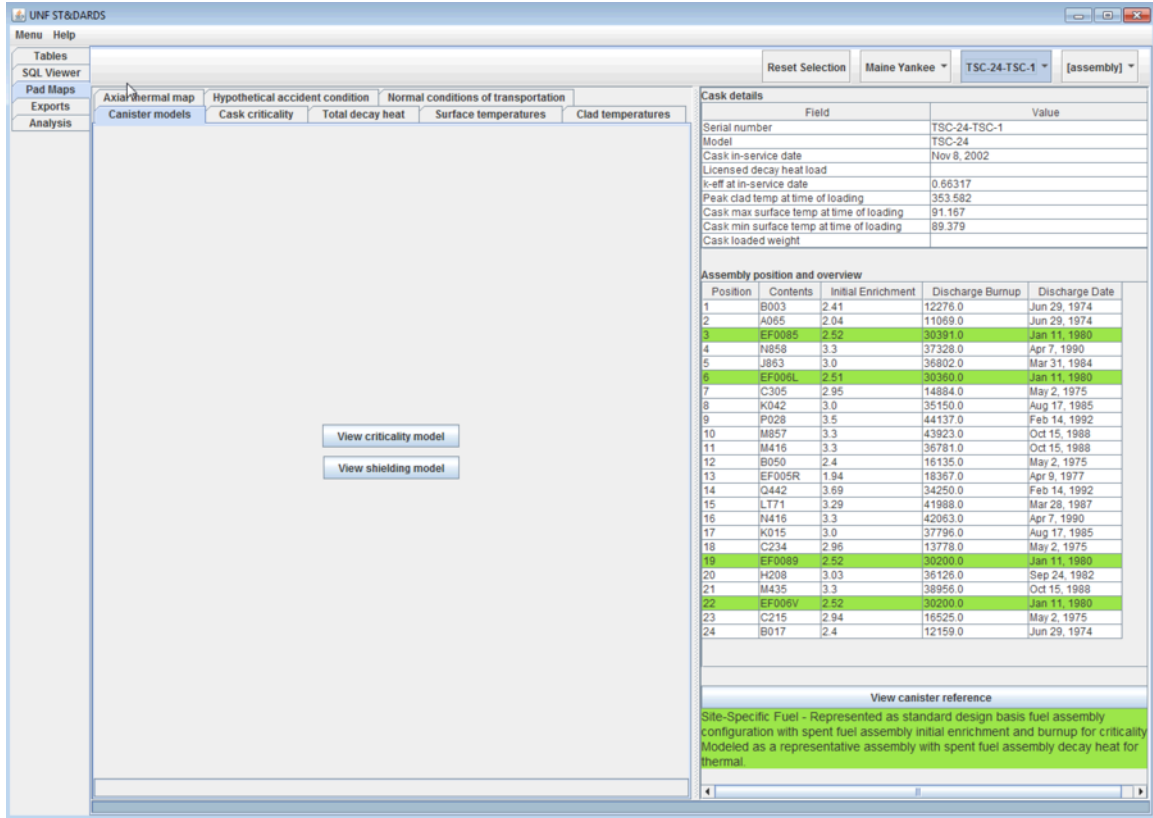
Upon selection, the data tabs will be populated for viewing.



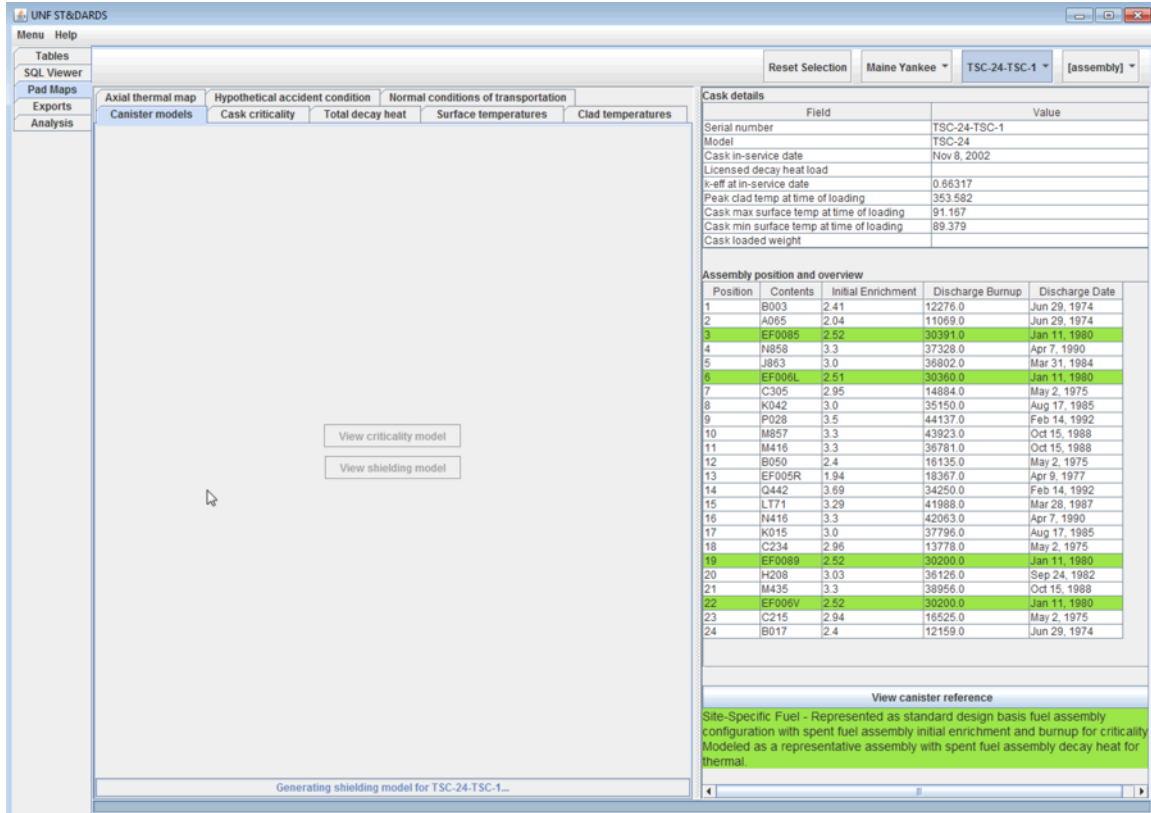
Select the cask labeled **TSC-24-TSC-1** from the [cask] drop down.



Cask-specific data will be loaded and available for viewing.



Click the **View shielding model** button to generate the cask-specific shielding model and render it in the **Fulcrum** graphical user interface. This can take a few seconds as the Unified Database is queried for all parameters needed for construction of the shielding model.



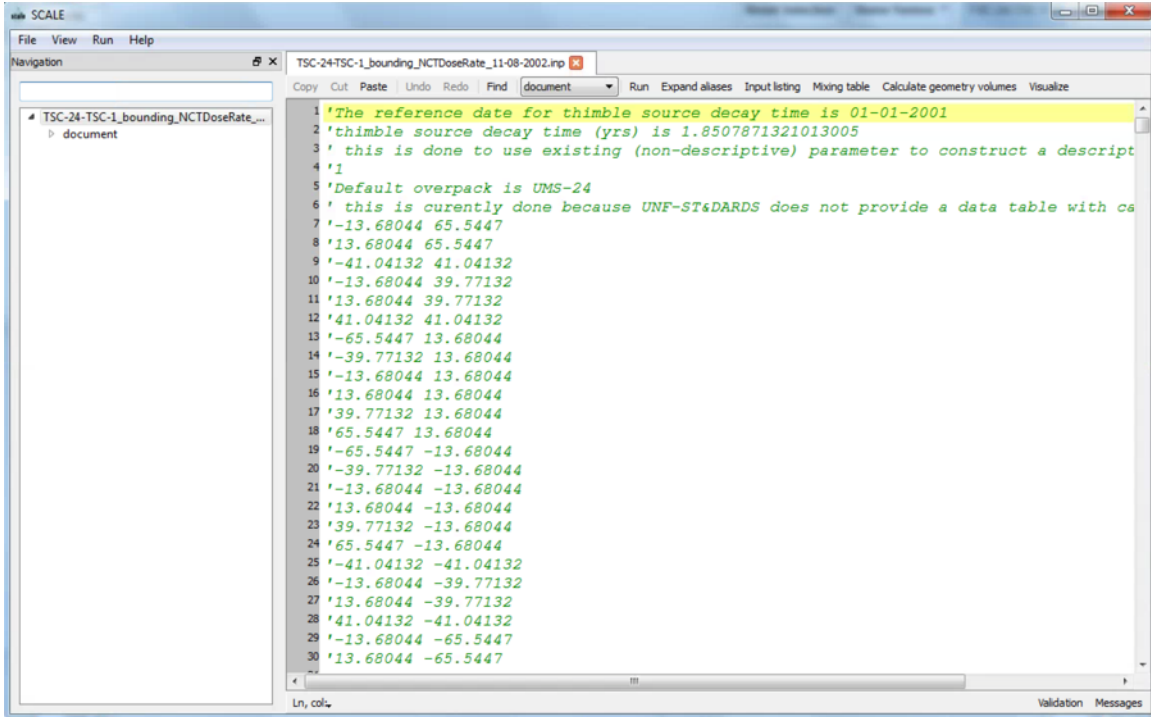
The progress bar will indicate **Generating shielding model for TSC-24-TSC-1...** with the **cmd** console windows providing detailed activity listing the shielding model construction. Additionally, because **Fulcrum** is an external application that communicates with UNF-ST&DARDS the potential exists that Fulcrum is not running. A **ConnectException**, as seen in the **cmd** console, will be produced when UNF-ST&DARDS starts **Fulcrum**.

```

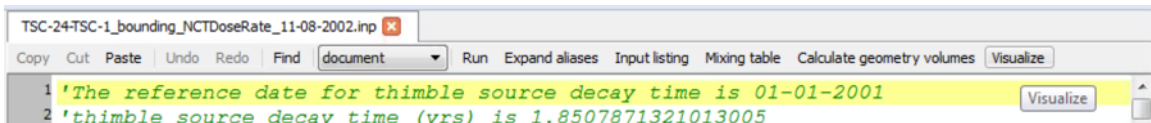
C:\Windows\system32\cmd.exe - java -jar -Xmx2048m -D.unf.rc=C:\Users\raq\Desktop\UNF-STANDARDS\unf.rc C:\Users\raq\Desktop\UNF-STANDAR...
n_11-08-2002.inp.json.gen
Creating input file C:\Users\raq\Desktop\UNF-STANDARDS\execution\2801_C305_bounding_activation_11-08-2002.inp
Aggregating parameter set for assembly K042 in Maine Yankee of type C1414A
Creating input parameter file C:\Users\raq\Desktop\UNF-STANDARDS\execution\2801_K042_bounding_activation_11-08-2002.inp.
json
Parameter set and template Gen file located at C:\Users\raq\Desktop\UNF-STANDARDS\execution\2801_K042_bounding_activation_11-08-2002.inp.json.gen
Creating input file C:\Users\raq\Desktop\UNF-STANDARDS\execution\2801_K042_bounding_activation_11-08-2002.inp
Creating input parameter file C:\Users\raq\Desktop\UNF-STANDARDS\execution\TSC-24-TSC-1_bounding_NCTDoseRate_11-08-2002.inp.json
Parameter set and template Gen file located at C:\Users\raq\Desktop\UNF-STANDARDS\execution\TSC-24-TSC-1_bounding_NCTDoseRate_11-08-2002.inp.json.gen
Creating input file C:\Users\raq\Desktop\UNF-STANDARDS\execution\TSC-24-TSC-1_bounding_NCTDoseRate_11-08-2002.inp
Finished generating input parameter files for bounding canister shielding for canister TSC-24-TSC-1
Finished bounding canister shielding calculations for canister TSC-24-TSC-1
Sep 29, 2014 6:42:04 PM util.FulcrumManager connect
SEVERE: null
java.net.ConnectException: Connection refused: no further information
    at sun.nio.ch.SocketChannelImpl.checkConnect(Native Method)
    at sun.nio.ch.SocketChannelImpl.finishConnect(Unknown Source)
    at sun.nio.ch.SocketAdaptor.connect(Unknown Source)
    at util.FulcrumManager.connect(FulcrumManager.java:59)
    at view.padmap.CanisterView$8.doInBackground(CanisterView.java:1150)
    at view.padmap.CanisterView$8.doInBackground(CanisterView.java:1068)
    at javax.swing.SwingWorker$1.call(Unknown Source)
    at java.util.concurrent.FutureTask.run(Unknown Source)
    at javax.swing.SwingWorker.run(Unknown Source)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(Unknown Source)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source)
    at java.lang.Thread.run(Unknown Source)

```

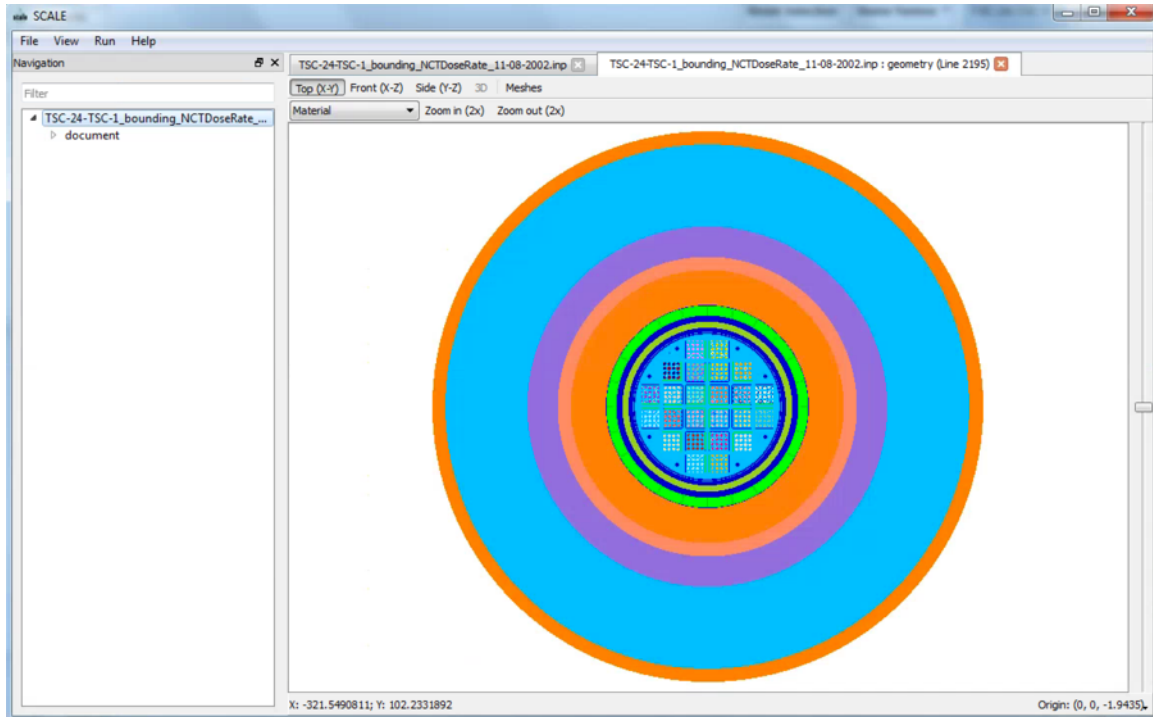
Upon successfully constructing the shielding model, it is displayed in **Fulcrum**.



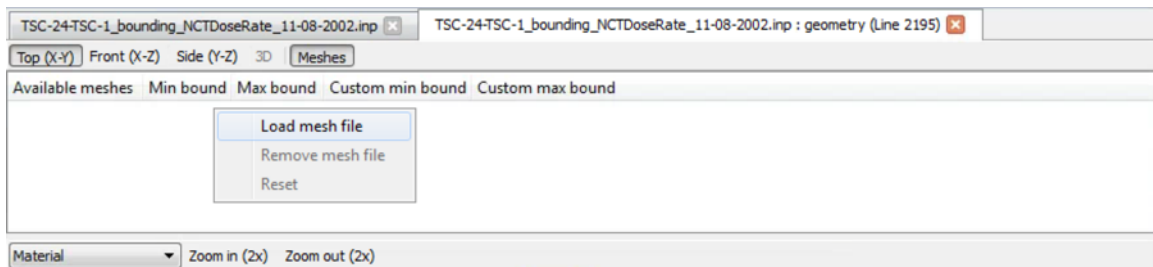
We can now render the shielding model using Fulcrum’s built-in geometry viewer by left clicking the **Visualize** button.



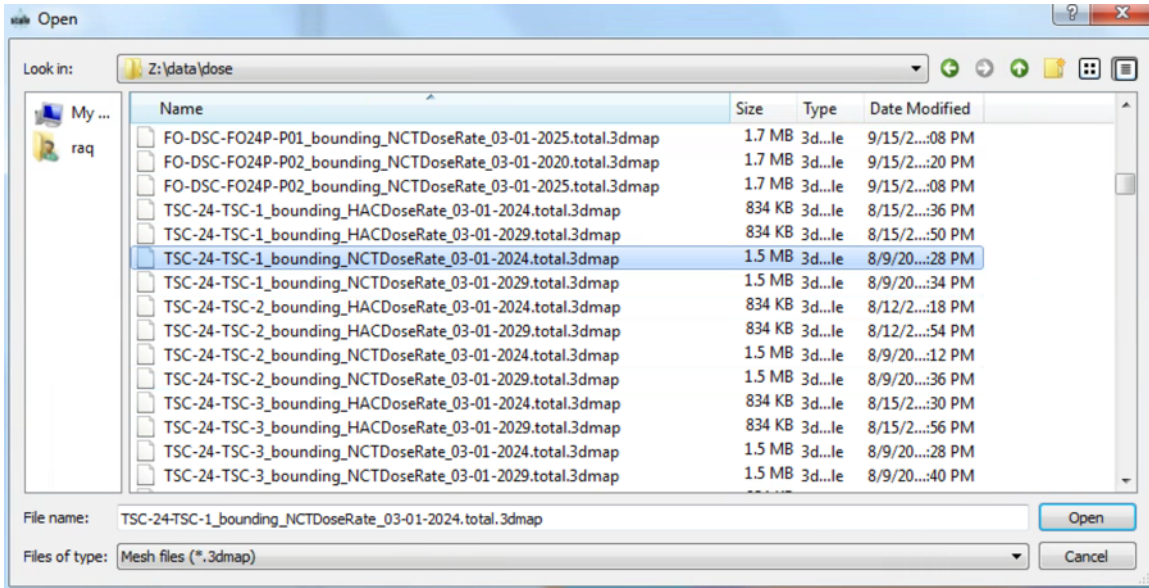
The default Top (X-Y) 2D cross-section through the middle of the cask will be rendered.



We want to verify the Dose Rate Map Pack was properly installed, so we will load a Dose Rate Map for the selected Cask by clicking the **Mesh** menu button and right clicking and selecting **Load mesh file**.

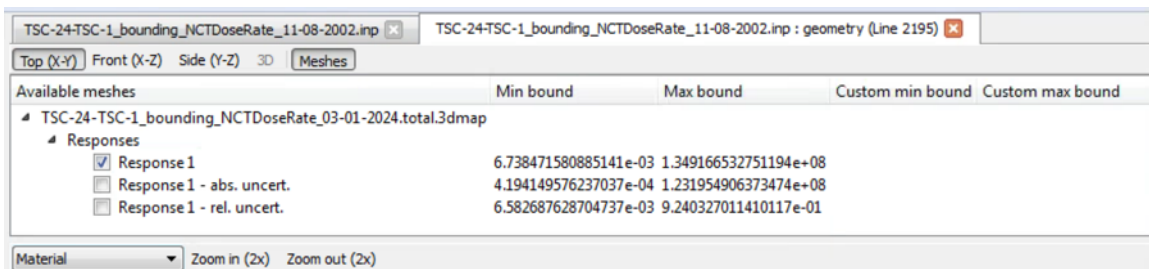


You will be prompted with a file browser where you need to navigate to the **C:\Users\user\Desktop\UNF-STANDARDS\data\dose** directory, where user has been replaced with your user name.

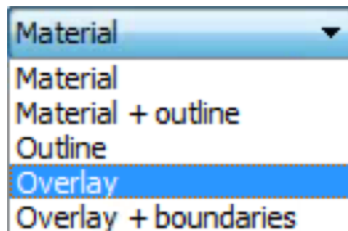


Select the **TSC-24-TSC-1_bounding_NCTDoseRate_03-01-2024.total.3dmap** file and left-click the **Open** button. HAC and NCT stand for Hypothetical Accident Conditions and Normal Conditions of Transportation respectively.

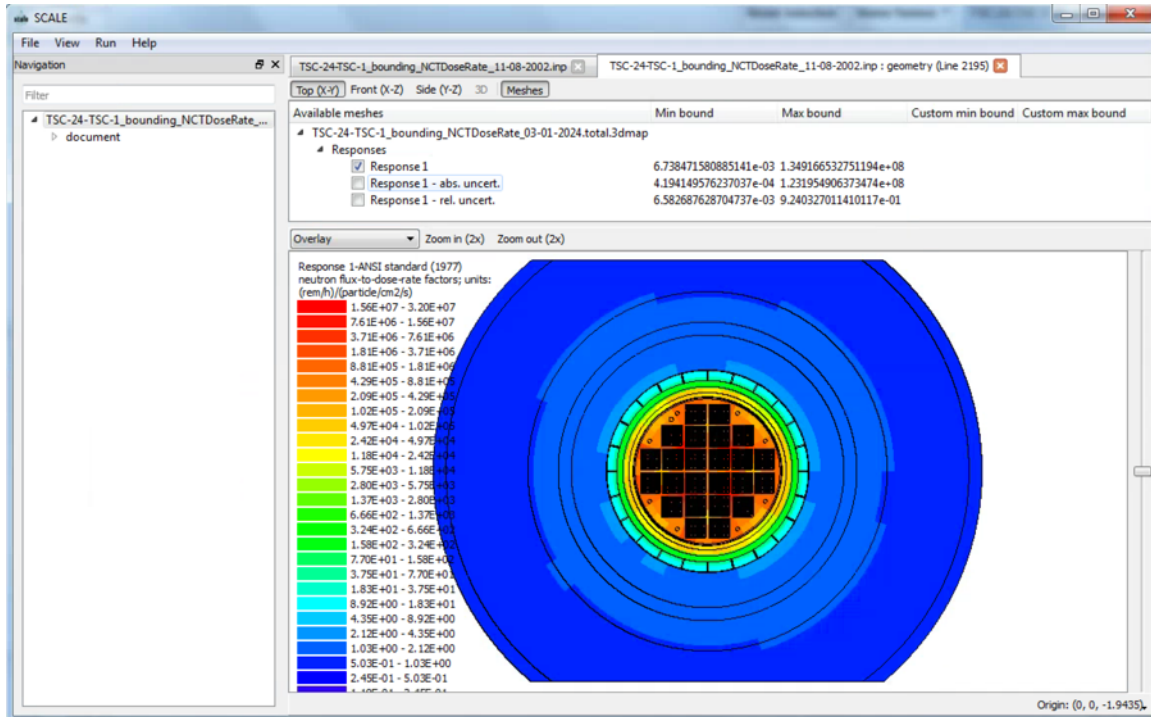
The Fulcrum mesh manager will now indicate the associated 3dmap Dose Rate file with the shielding model document loaded. Expand the Responses and select **Response 1**.



The dose rate will be loaded into Fulcrum and can now be visualized as an overlay on the geometry by updating the **Material** selection to be **Overlay**.



The overlay view presents the 3d dose data overlaid on the geometry.



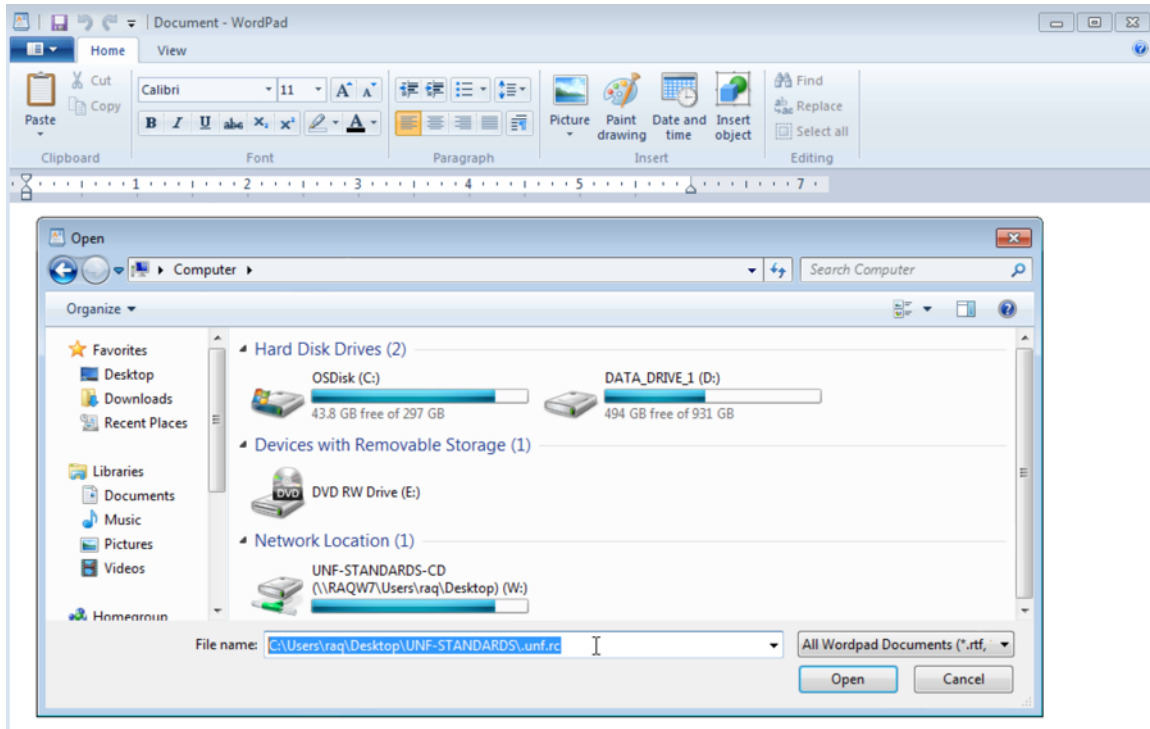
These steps can be reproduced for casks for which Dose Rate results are available.

You have now verified UNF-ST&DARDS has successfully been installed by launching UNF-ST&DARDS, requesting visualization of a specific cask requiring UNF-ST&DARDS to query the Unified Database, construct a cask-specific shielding input by expanding appropriate templates. And lastly, viewing installed Dose Rate results.

1.2.5 Activating Geographic Information System (GIS)

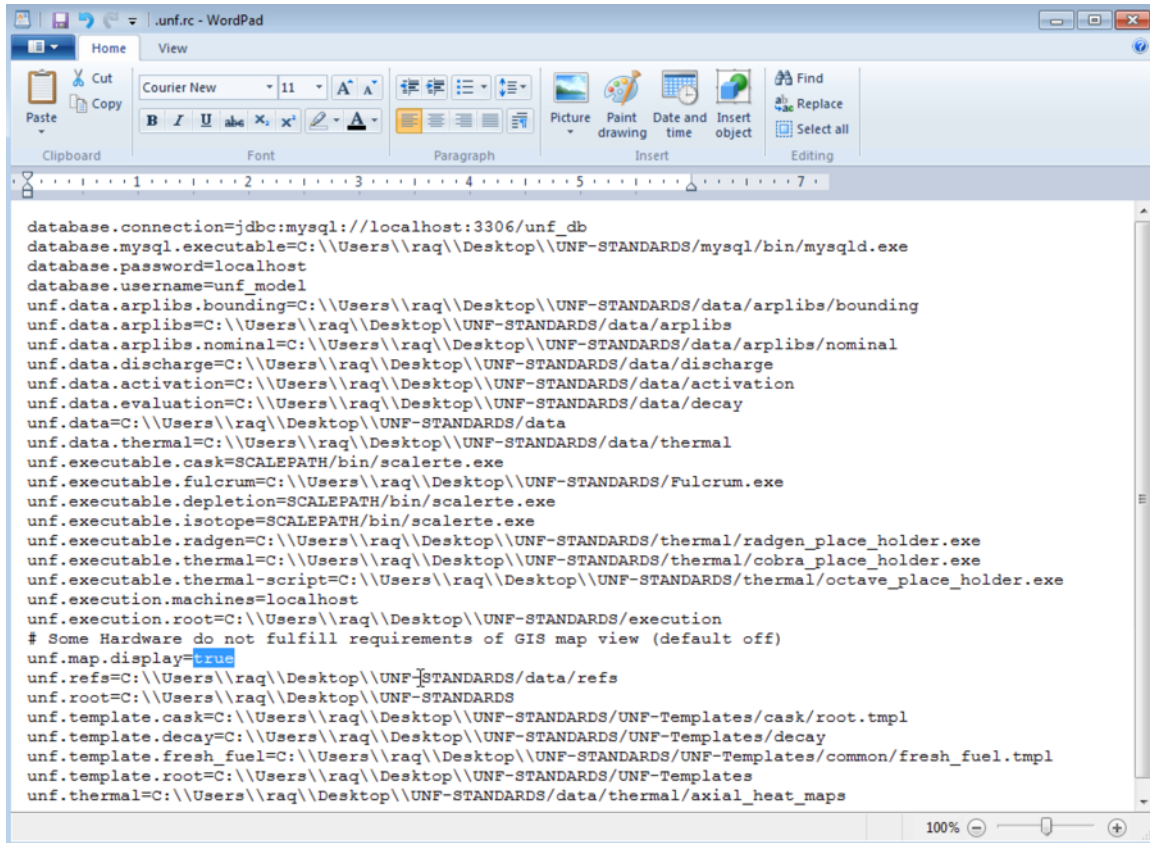
UNF-ST&DARDS out of the box has the GIS pad map view inactive due to potential graphics card hardware support issues. Activating the GIS map view is easy and can be done by updating the UNF-ST&DARDS resource file. If you followed the instructions INSTALLING UNF-ST&DARDS the resource file will be located at **C:\Users\user\Desktop\UNF-STANDARDS\.unf.rc**, where **user** is your user name.

With UNF-ST&DARDS not running, edit the file with your Windows **WordPad** application.



In order to activate the GIS pad map view, change the **unf.map.display** property from **false** to **true**.

Your **unf.map.display** should now be **true**.

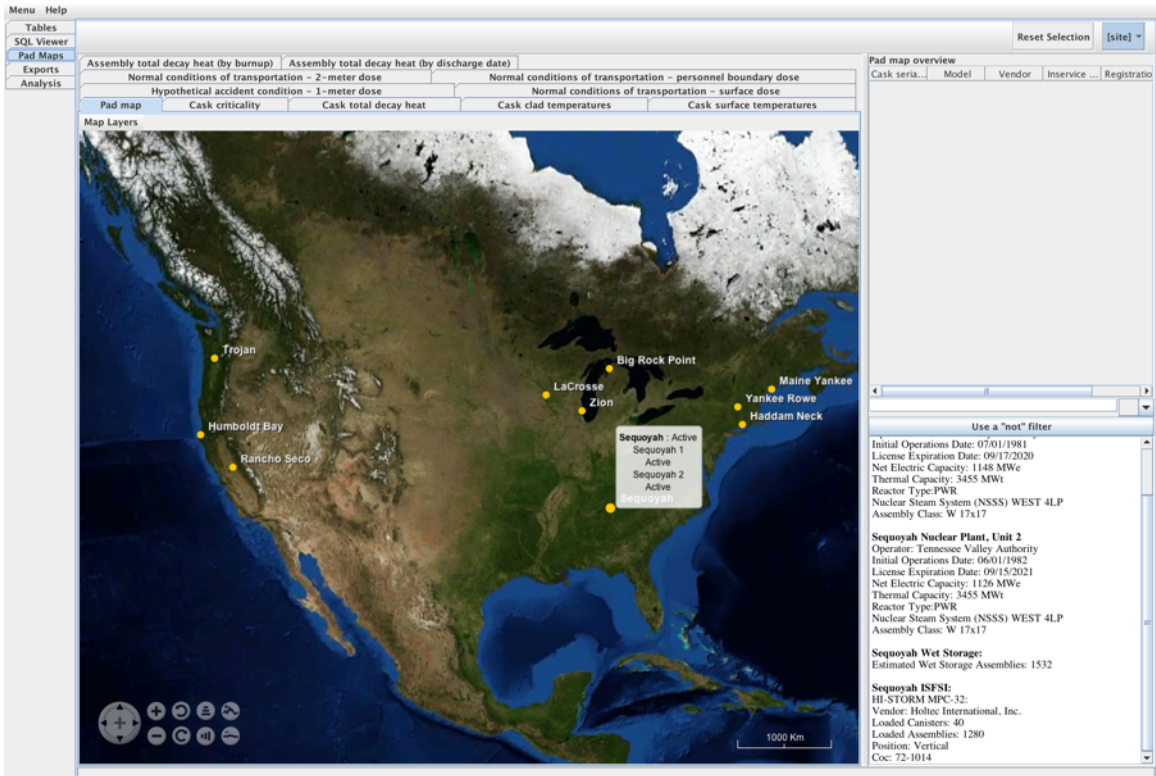


```

.unf.rc - WordPad
Home View
Clipboard Font Paragraph Insert Editing
Courier New 11
Cut Copy Paste
B I U abc Xx
Picture Paint drawing Date and time Insert object Find Replace Select all
1 2 3 4 5 6 7
database.connection=jdbc:mysql://localhost:3306/unf_db
database.mysql.executable=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\mysql\\bin\\mysqld.exe
database.password=localhost
database.username=unf_model
unf.data.arplibs.bounding=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\data\\arplibs\\bounding
unf.data.arplibs=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\data\\arplibs
unf.data.arplibs.nominal=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\data\\arplibs\\nominal
unf.data.discharge=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\data\\discharge
unf.data.activation=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\data\\activation
unf.data.evaluation=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\data\\decay
unf.data=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\data
unf.data.thermal=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\data\\thermal
unf.executable.cask=SCALEPATH/bin/scalerte.exe
unf.executable.fulcrum=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\Fulcrum.exe
unf.executable.depletion=SCALEPATH/bin/scalerte.exe
unf.executable.isotope=SCALEPATH/bin/scalerte.exe
unf.executable.radgen=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\thermal\\radgen_place_holder.exe
unf.executable.thermal=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\thermal\\cobra_place_holder.exe
unf.executable.thermal-script=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\thermal\\octave_place_holder.exe
unf.execution.machines=localhost
unf.execution.root=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\execution
# Some Hardware do not fulfill requirements of GIS map view (default off)
unf.map.display=true
unf.refs=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\data\\refs
unf.root=C:\\Users\\raq\\Desktop\\UNF-STANDARDS
unf.template.cask=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\UNF-Templates\\cask\\root.tpl
unf.template.decay=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\UNF-Templates\\decay
unf.template.fresh_fuel=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\UNF-Templates\\common\\fresh_fuel.tpl
unf.template.root=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\UNF-Templates
unf.thermal=C:\\Users\\raq\\Desktop\\UNF-STANDARDS\\data\\thermal\\axial_heat_maps
100%

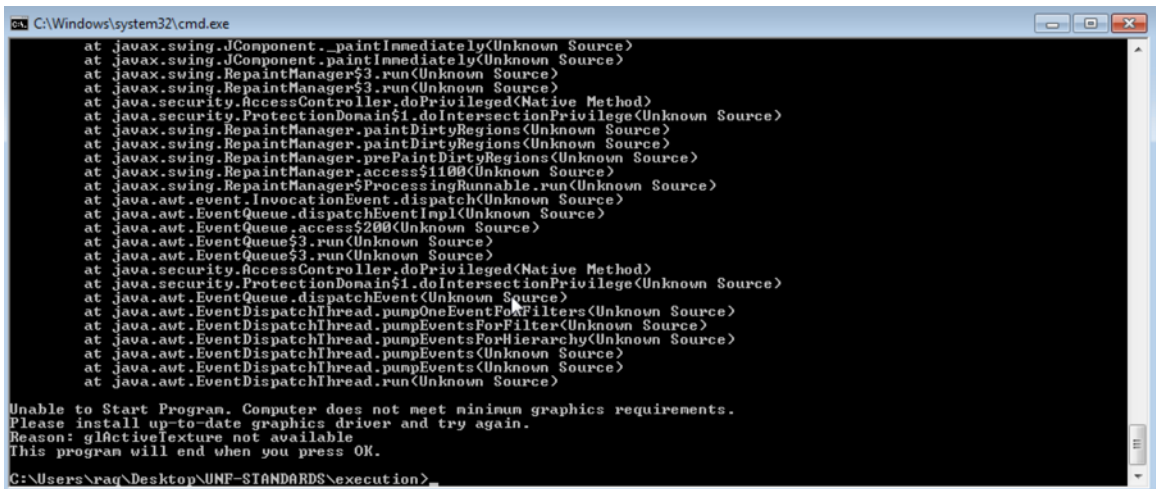
```

Save the file, and restart UNF-ST&DARDS. If your graphics card hardware supports the necessary components in UNF-ST&DARDS, the GIS map view will appear.



Consult the UNF-ST&DARDS Help documentation (press F1 key or click Help>View Help) for assistance with interacting with the GIS map view.

If your graphics card hardware does not support the necessary components used by UNF-ST&DARDS, UNF-ST&DARDS will quit upon selection of the Pad Maps tab and you will see the following message in your **cmd** console.



Reverting your **C:\Users\user\Desktop\UNF-STANDARDS\.unf.rc** file to have **unf.map.dispaly=false** will restore default pad map behavior.

1.2.6 Question

UNF-ST&DARDS is deployed with help documentation to assist users. The help documentation is available via the **Help>View Help** menu item. Additionally, context specific help is available by pressing the **F1** key.

Further assistance can be acquired by emailing UNFHelp@ornl.gov

ARCHITECTURE AND CAPABILITIES

2.1 System Architecture

UNF-ST&DARDS is illustrated in Fig. 2.1 and consists of six primary elements: (1) a user interface (graphical as well command line), (2) the Unified Database, (3) a Unified Database software development kit, (4) a template repository that contains constructs for specific nuclear safety analysis tools that have been integrated, (5) an application-agnostic template engine, and (6) a process manager that handles interactions between the different elements. The nuclear safety analysis tools and their user interfaces are externally developed to function independently, but they have been integrated into UNF-ST&DARDS for automating different analysis streams. The current nuclear safety analysis tools consist of SCALE¹, a comprehensive modeling and simulation suite for nuclear safety analysis and design, and Coolant Boiling in Rod Arrays–Spent Fuel Storage (COBRA-SFS)², a thermal-hydraulic analysis code. SCALE’s Fulcrum user interface provides input editing and validation, data visualization, and output viewing, and it has been integrated into UNF-ST&DARDS to provide SCALE input preview, as well as model and results visualization. The software infrastructure has been designed with the capability to integrate other tools, tool interfaces, and analysis packages in a similar fashion, but this would require development of appropriate templates for inclusion in the template repository, as well as identification of specific input attributes to ensure they are present in the Unified Database or added as necessary. The UNF-ST&DARDS process manager was developed to manage (1) the workflow across the Unified Database, (2) integrated nuclear safety analysis tool templates and applications, and (3) results retrieval and interrogation. This software uses the Java Persistence application programming interface (API) (JPA) entity, controller, and annotation classes to abstract the communication with the Unified Database. The SNF templates are implemented, in conjunction with structured parameter sets and TemplateEngine, to abstract the management of nuclear safety and systems processes. UNF-ST&DARDS uses the Java programming language to provide cross-platform capabilities

¹ SCALE: A Comprehensive Modeling and Simulation Suite for Nuclear Safety Analysis and Design, ORNL/TM-2005/39, Version 6.1, Oak Ridge National Laboratory, Oak Ridge, Tennessee, June 2011.

² T.E. Michener, D. R. Rector, J. M. Cuta et al., COBRA-SFS: A Thermal-Hydraulic Code for Spent Fuel Storage and Transportation Casks, PNL-10782, Pacific Northwest National Laboratory, Richland, Washington, 1995.

and increase developer throughput.

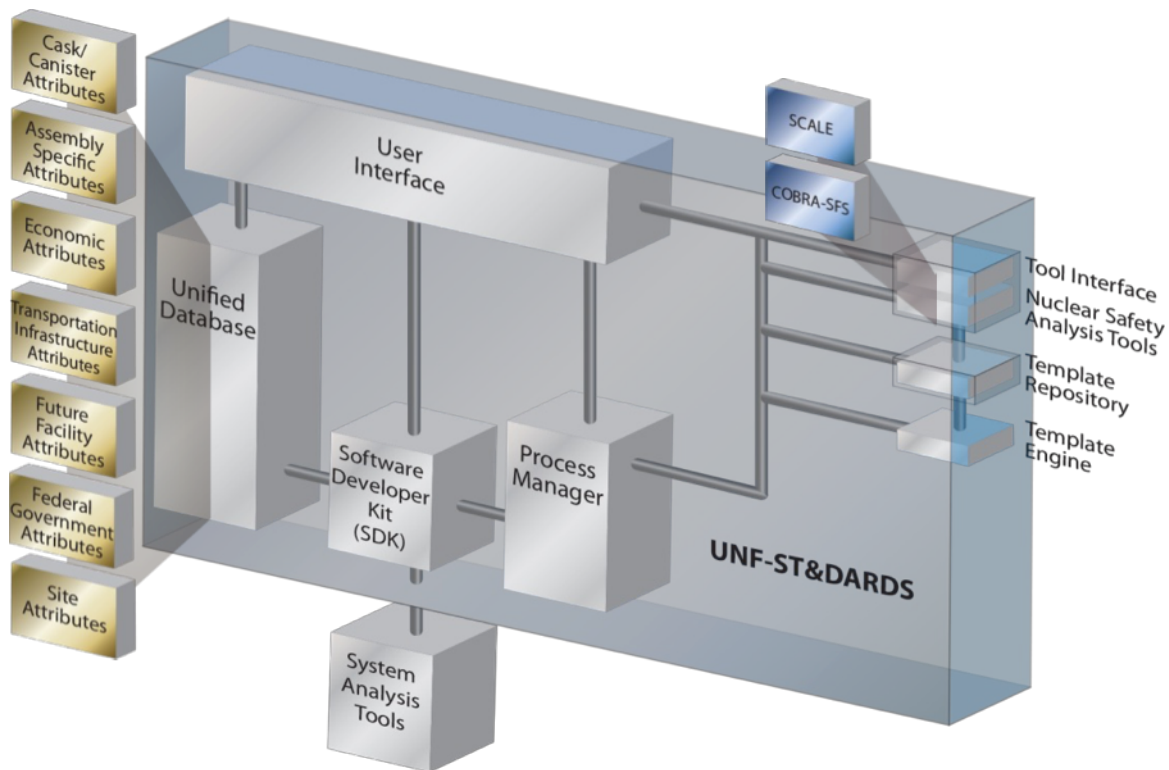


Fig. 2.1: UNF-ST&DARDS architecture.

UNF-ST&DARDS is a three-tier application consisting of a presentation, process, and data tier. In a three-tier application, the presentation tier is the top tier, providing display of available services and results, typically in a GUI. The process or business tier is the middle tier, providing access to process functionality and controls. The data tier is the bottom tier, providing data access and persistence, and it often includes an API or a software development kit (SDK). The UNF-ST&DARDS data tier is composed of the Unified Database and its SDK. The process tier coordinates the nuclear safety applications and the Unified Database. The presentation tier is a GUI that allows point-and-click access to analysis capabilities via the process tier, as well as data extraction and visualization.

The process tier includes a command line interface for access to execute nuclear safety applications with a concise set of input parameters. The input parameters are described in JavaScript Object Notation (JSON)³. JSON is a standard, human readable, lightweight (i.e., very few format constructs) data interchange format. The command line interface provides access to launch all UNF-ST&DARDS analysis capabilities. In the GUI, an analysis page allows the user to select an analysis type (e.g., criticality) and either (1) reactor type batch (e.g., all pressurized water reactors [PWRs]), or (2) specific analysis. The GUI is described Sect. 5.

³ ECMA-International, The JSON Data Interchange Format, ECMA-404, CH-1204 Geneva, October 2013.

2.2 Unified Database

The Unified Database provides a comprehensive, controlled source of technical data for various waste management system analysis/evaluation tools, as well as fuel cycle system analyses and safeguards and security studies. The Unified Database has been designed to be flexible and expandable to provide controlled inputs to a variety of tools and applications. Interface controls, relevant data, and data formats will continue to be identified and added as additional requirements for other tools and capabilities are identified.

Development of the Unified Database involves a staged database approach that includes the following database stages:

- Development – contains databases used for development purposes, such as structural change and new data set import verification and validation
- Production – the database from which input parameters for all processes are acquired and their results imported
- Test – the database used for continuous integration (CI), containing the minimum set of data required to test UNF-ST&DARDS
- Deployed – databases that have been released (i.e., versioned snapshots of the production database)

With these stages, the Unified Database development team can work in clearly delimited areas, eliminating the potential of affecting production capabilities while conducting development tasks.

2.2.1 Unified Database Content

The following datasets are stored in the Unified Database:

Facility attributes include utility, site, reactor, pool, and independent spent fuel storage installation (ISFSI) data. **Federal government radioactive waste attributes** include government-owned high-level radioactive waste (HLW) and government-owned SNF data. **Future facility attributes** include interim storage facility (ISF), repackaging facility, and repository data. **Transportation infrastructure attributes** include rail, heavy-haul truck, legal-weight truck, and barge assets data, in addition to transfer times between these transportation modes. **Assembly-specific attributes** include actual and projected assembly discharge information and characteristics (e.g., isotopic concentrations, source terms). **Cask/canister attributes** include overpack, capacity, physical properties, unit processing times, unit costs, and characteristics (e.g., dose, component temperatures, keff). **Economic attributes** include the unit cost of reactor sites, transportation infrastructures, ISF, repackaging facility, and repository surface costs.

These data have been collected from the DOE Office of Civilian Radioactive Waste Man-

agement, the General Council (GC) nuclear data forms (RW-859/GC-859)¹ database, and other sources. The data not obtained from the RW-859 database were obtained by having subject matter experts data mine various sources to extract the required information. These sources include DOE studies, open literature, vendor data, and utility data. One essential requirement of the Unified Database is that all of the data within the database must be traceable. All data references are uploaded to the Centralized Used Fuel Resource for Information Exchange (CURIE) website at curie.ornl.gov, allowing permanent storage and simple retrieval for all imported references. Correlating the data from the subject matter experts to the appropriate reference source is initially handled with the use of Microsoft Excel spreadsheets that include the data and web links to the references stored on CURIE prior to import into the database. The subject matter experts are responsible for verifying that the data and the reference sources concur.

A data importer was developed to upload the Excel spreadsheets to the Unified Database. The data importer uses the JPA entity, annotation, and controller classes to generate the appropriate modifications to the Unified Database and to successfully conduct data imports. The importer uses an extension convention to identify the type of data being imported. For each data record created, a reference record is also created that contains the source of the import, such as the Excel spreadsheets populated by the subject matter experts.

Data that have been imported to the Unified Database continue to be editable. Any changes to the data require review and approval before they are implemented into the production version of the database. Once changes are made to the production version, the data that have been changed and the rationale for the changes are logged and distributed with the next deployment of the database.

The Unified Database change control process records the subject matter expert's name and archives the previous data within the database. Storing the data, notes, references, and archived data is accomplished by grouping the tables into three main areas: "data," including data tables with data necessary for analysis tools; "reference," including tables of reference information corresponding to specific data tables, and "history," including tables that serve as change logs which record users and modification times of specific data and reference tables, along with the data and references that were changed.

The reference tables reflect the data tables, with the exception that each data column is a text field containing the reference as a uniform resource identifier (URI). In addition to reflecting each table's data column with a reference URI, each reference column has a corresponding modification date column indicating the time at which the data and their associated references were imported into the database. A history table exists for each reference and data table with the log of table changes. Specifically, history tables reflect the associated tables column for column, along with the time of the change, the type of change (e.g., "update," "delete"), and the identification of the user making the change.

¹ "RW-859 Nuclear Fuel Data," US Department of Energy, Energy Information Administration, Washington, D.C., October 2004.

2.2.2 Unified Database Design

The Unified Database design was created using the MySQL Community edition database. MySQL Community edition is a freely available, well-documented, open source, relational database server that uses SQL. Many utilities exist to assist in development and maintenance of MySQL databases, making it a productive environment to streamline the data needs of nuclear safety and systems analysis tools.

SQL databases are composed of tables, each of which contains a fixed number of named columns. Each table row constitutes a “record,” many of which can exist in a given table. “Attributes” exist that describe each column in a table. These attributes are often the type of data in the column (integer, real, text, date, etc.), as well as the constraints (unique, auto-incrementing, optional, table relations, etc.). The SQL database, with its structured and relational constructs, lends itself to data normalization. The process of data normalization involves separating tables into their constituents until the table represents one and only one component. Normalization removes redundant data and minimizes data dependencies.

The Unified Database employs normalization as a general best practice in data management. Due to normalization, each record in a table is uniquely identifiable, creating what is called a primary key. Relationships between tables are described with a foreign key field that stores the primary key of the related table.

With hundreds of tables and thousands of table columns, nomenclature consistency is of paramount importance. The Unified Database uses a consistent convention for naming tables and columns to ensure consistent use and interpretation of the data and their relationships. Columns use a verbose name with a suffix indicating the unit of measure. For example, the assembly table contains assembly-specific attributes, one of which is the initial uranium mass. These attributes use a verbose name and the column data type (e.g., kilograms), as in *initial_uranium_kg*. Relationship columns include the related table’s name and the primary key column name separated by an underscore. The assembly decay results table has a relationship with the assembly listed in the assembly table (assembly) from which the results were derived; the column that stores the foreign key is named *assembly_id*, from which the table (assembly) and primary key (*id*) of the related table are easily discernable.

In addition to the consistent nomenclature and the suffix for the unit of measure, the Unified Database includes a data dictionary to provide a more detailed explanation of what each table and column contains. The data dictionary stores a description for each column and a full name for the unit of measure. As an example, the entry for column *reactor_id* on the cycle table explains, “Foreign key to the reactor table indicating the reactor to which this cycle belongs.” For a user viewing an unfamiliar table, this information can help clarify any questions about what the data mean.

2.2.3 Unified Database Software Development Kit (SDK)

One of the key features of the Unified Database is the capability for integration with multiple computational tools and models. Integration with the Unified Database is streamlined with a Java programming language, SDK. The majority of the Unified Database's SDK consists of automatically generated JPA classes. JPA interfaces the concrete data storage medium (MySQL) from the abstract data relationships and includes the Java Persistence Query Language (JPQL). The JPA classes consist of entities, annotations, and controllers. The entity classes represent the data tables. The annotation classes are used by JPQL to provide programmatic (i.e., via software), storage agnostic, conditional entity retrieval. The controller classes provide a means to create, edit, and delete entities. An advantage of the JPA is that significant portions of user-experienced SQL runtime errors are migrated to developer-mitigated compile-time errors. Compile-time errors are easier to manage by developers and help to ensure that the system is more stable for users.

2.3 Templates

UNF-ST&DARDS uses templates that serve as building blocks from which application-specific input is defined. The UNF-Templates repository provides UNF-ST&DARDS application-agnostic input generation. Templates allow UNF-ST&DARDS to use the same set of parameters to succinctly communicate to each application while decoupling UNF-ST&DARDS from any specific application. The TemplateEngine, which processes and expands templates, is written in the Java programming language and includes an SDK and standalone Java executable. The SDK facilitates the TemplateEngine integration into UNF-ST&DARDS, while the standalone Java executable provides a means by which template creators can test and verify their templates. All the functionalities of the TemplateEngine are described in Appendix A. The parameter set is defined as a structured collection of attributes, objects, and arrays of attributes. Objects and arrays can be embedded within one another and themselves. JSON is the syntax that represents a parameter set.

2.3.1 Template Repository

The UNF-Templates repository uses the [Mercurial Distributed Version Control System \(DVCS\)](#) to track changes, manage collaboration between template developers, and mitigate conflicts between developer's changes. Mercurial is integrated with [FogCreek's Fogbugz](#) management system, which features issue tracking, discussion forums, customer relationship management, and evidence-based scheduling to further enhance project management and collaboration. Current tools with integrated templates are the SCALE code system and COBRA-SFS. The template repository structure is as follows:

- input templates for SCALE's CSAS6 criticality safety analysis sequence,

- common templates used by different analysis types (e.g., header templates, file copy, etc.),
- input templates for containment analysis,
- input templates for SCALE's Transport Rigor Implemented with Time-dependent Operation for Neutronic (TRITON) fuel assembly depletion tool,
- input templates for SCALE's ORNL Isotope Generation and Depletion Code (ORIGEN) Assembly Isotopics (ORIGAMI) analysis sequence for computing fuel assembly discharged and decayed isotopic inventory and decay heat,
- the regression test directory containing JSON files for testing,
- input templates for SCALE's MAVRIC shielding and dose analysis sequence,
- input templates for COBRA-SFS thermal analysis, and
- COBRA Creator Of Most Assembly's Necessary Data, Even Radgen (COMANDER) utility template for generating fuel assembly geometry subtemplates for COBRA-SFS thermal analysis.

These templates use [Kitware's CMake](#), the cross-platform, open-source build and test system, in conjunction with the SCALE regression harness, to construct a UNF-Templates test harness that helps prevent regression in automated template execution capabilities. UNF-Templates testing needs the TemplateEngine and SCALE regression harness to be available. The TemplateEngine is needed for the expansion of the templates and subsequent verification of template intent. SCALE is needed for its regression harness, which drives the TemplateEngine, dissects the result, and compares it against the expected result.

A template expansion test case is created and placed into the regression test directory (UNF-Templates/regression/input). CMake is executed to create a platform-specific configuration of the test in the testing directory. Once configured, the test is verified, generating a .table file containing extracted points of interest which are subsequently compared against the expected results via a specialized difference tool provided by the SCALE regression harness. Once the result from the test case is as expected, the resulting .table file is moved into the expected results directory (UNF-Templates/regression/output). Upon future regression testing, the testing harness has an approved test result to compare against to demonstrate that the acceptance criteria have been satisfied, resulting in a pass or signaling a regression. The regression tests examine the functional requirements of the system. All new templates and their tests are added to the UNF-Templates Mercurial repository and pushed back to the central development repository for integration into UNF-ST&DARDS and other templates.

The [Wiki page](#) (inside ORNL network) presents (1) the template repository cloning process from the central repository to a local machine for template development and (2) processes for pushing and pulling changes from the central repository. Access to the template repository requires permission from the system administrators and access to the Oak Ridge National Laboratory (ORNL) internal network. The permission process can be initiated by sending an email to UNFHelp@ornl.gov.

The criticality analysis–specific template development process is described in Ref. ¹. Similarly, the thermal ², shielding ³, and containment ⁴ template development processes are documented.

2.4 Analysis Sequence

UNF-ST&DARDS currently provides the following nuclear safety analyses capabilities:

- Depletion and decay: provides SNF isotopic compositions (subsequently used in criticality analysis), heat load (subsequently used in thermal analysis), radiation source (subsequently used in dose/shielding calculations), and isotopic activities in curies (subsequently used in containment calculations),
- Criticality: calculates the keff for as-loaded casks,
- Thermal: provides cladding and surface temperatures of as-loaded casks,
- Shielding: provides radiation dose rate maps outside the as-loaded transportation casks, and
- Containment: calculates allowable leakage rate for as-loaded transportation casks.

The current integrated nuclear safety analysis tools that perform all of the above functions (excluding containment) are SCALE and COBRA–SFS. SCALE provides the ORIGEN ORIGAMI sequence, the Criticality Safety Analysis Sequence with KENO-VI (CSAS6), and the Monaco with Automated Variance Reduction using Importance Calculations (MAVRIC) shielding sequence, which are used for performing the depletion and decay, criticality, and shielding analyses, respectively. The COBRA-SFS thermal hydraulics application is used with the COBRA-COMANDER utility ¹ to provide system component temperatures, including peak and minimum clad temperatures, as well as cask surface temperatures. The containment analysis equations are implemented in templates, and the TemplateEngine expands these templates with cask-specific parameters to determine cask-specific allowable leakage rate.

UNF-ST&DARDS process manager contains the business logic (i.e., rules on how data can be created, displayed, stored, and changed) to manage the preparation, execution, and

¹ J.B. Clarity et al., Criticality Process, Modeling and Status for UNF-ST&DARDS, FCRD-NFST-2015-000440, Oak Ridge, TN, July 2015.

² K.R. Robb et al., Report Documenting Model Development for Thermal Analyses, ORNL/LTR-2013/434, Oak Ridge, TN, September 2013.

³ G.Radulescu et al., Models for Dose Rate Analyses, FCRD-NFST-2014-000358, Oak Ridge, TN, August 2014.

⁴ G.Radulescu et al., Containment Analysis Capability of UNF-ST&DARDS, FCRD-NFST-2015-000657, Oak Ridge, TN, March 2015.

¹ COBRA-COMANDER cycle 1.0, Input Generator for COBRA-SFS, ORNL/TM-2014/9 Oak Ridge National Laboratory, Oak Ridge, Tennessee, March 2014.

results retrieval of these applications and their subsequent processes. Process management and results are formally described in the unified modeling language (UML) sequence diagram depicted in [Fig. 2.2](#) In general, an application's process flow is as follows:

1. Retrieve the entities required for process execution from the Unified Database.
2. Retrieve the process input template.
3. Create a process input by expanding the process template with a JSON parameter set describing the entities.
4. Execute the process with the expanded process input.
5. Upon completion of processing, extract and store the results.

2.4.1 Fuel Depletion and Decay

The fuel assembly depletion and decay process requires assembly characteristics to execute. The process manager aggregates these characteristics (e.g., assembly-specific initial enrichment and uranium mass, irradiation history information, and discharge burnup) from the Unified Database, creating a parameter set. The fuel assembly depletion and decay template is acquired from the UNF-Templates repository and provided to the TemplateEngine with the parameter set, creating a problem- and application-specific input file. The application-specific input file to initiate the fuel assembly depletion and decay process is then executed by the process manager. This process results in generation of assembly-specific isotopic concentrations, decay heat, and radiation source term results that are extracted and stored for future use. The application's entire result set is archived, and specific parameters or subsets are imported into the Unified Database.

2.4.2 Criticality Safety

The cask criticality safety process requires cask type, cask inventory position, cask content characteristics, and an evaluation date to execute. The process manager aggregates these characteristics from the Unified Database and creates a parameter set. The fuel assembly inventory depletion and decay prerequisite processes are executed for each assembly in a given cask if the results are not available at the evaluation date. The criticality template is acquired from the UNF-Templates repository and provided to the TemplateEngine with the parameter set, creating a problem- and application-specific input file. The application-specific input file required to initiate the cask criticality analysis process is then executed. This process generates cask-specific neutron multiplication factor (k -effective) and uncertainty results that are extracted and stored for future use.

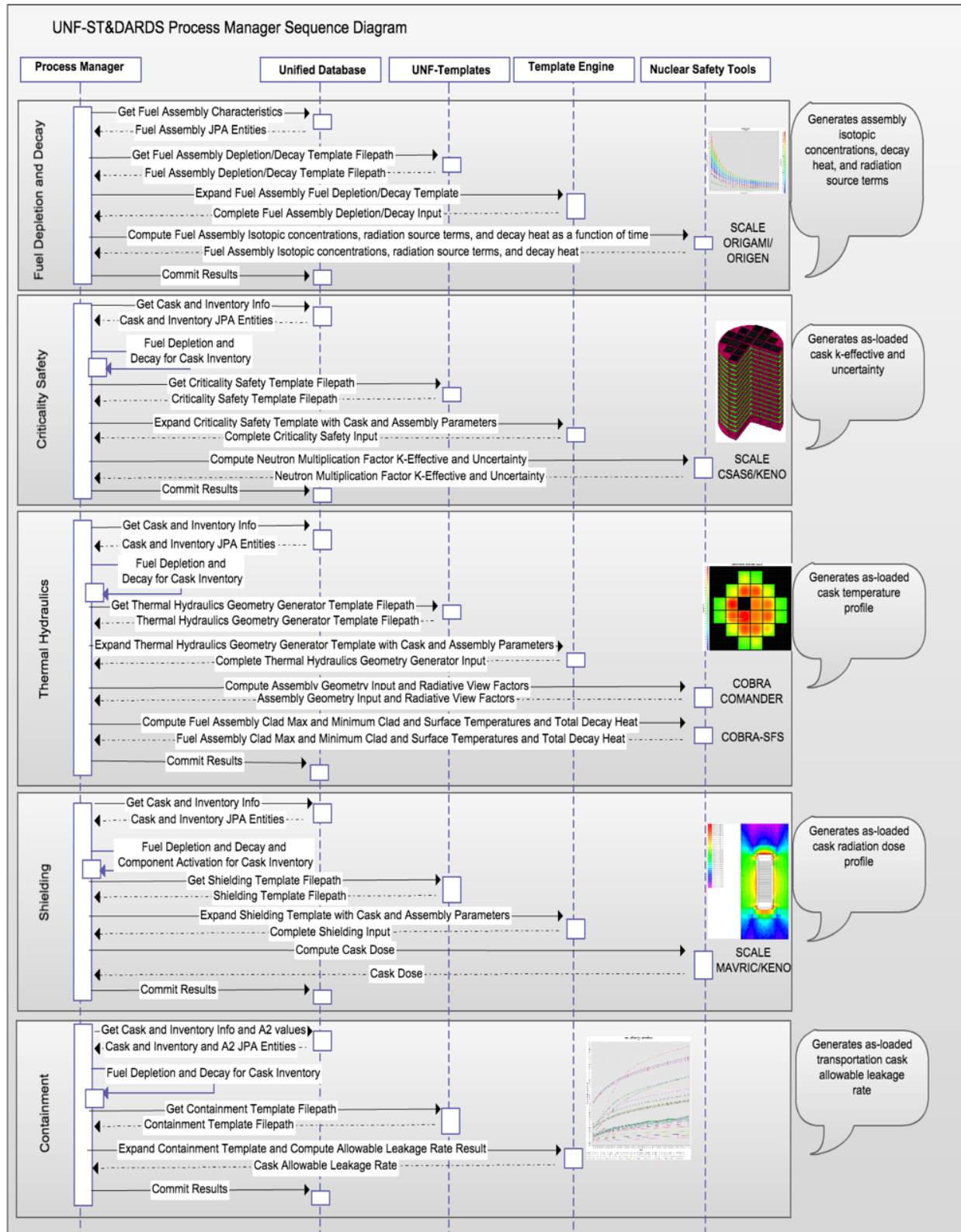


Fig. 2.2: UNF-ST&DARDS nuclear safety analysis sequence.

2.4.3 Thermal Hydraulics

The thermal hydraulics analysis process requires cask type, cask inventory position, cask content characteristics, and an evaluation date to execute. The process manager aggregates these characteristics from the Unified Database, creating a parameter set. Cask-specific fuel assembly inventory depletion and decay prerequisite processes are executed if results are not already available for the given evaluation date. The thermal hydraulics template is acquired from the UNF-Templates repository and provided to the TemplateEngine with the parameter set, creating a problem- and application-specific input file. The application-specific input file to initiate the cask thermal hydraulics analysis process is executed by the process manager. This process results in generation of assembly-specific peak and minimum clad temperatures, cask-specific surface temperatures, and total decay heat results that are extracted and stored for future use.

2.4.4 Shielding

The cask shielding process requires cask type, cask position, cask inventory characteristics, and an evaluation date to execute. The process manager aggregates these characteristics from the Unified Database, creating a parameter set. Cask-specific fuel assembly and component inventory depletion, decay, and activation prerequisite processes are executed if results are not already available for the given evaluation date. The shielding template is acquired from the UNF-Templates repository and provided to the TemplateEngine with the parameter set, creating a problem- and application-specific input file. The cask shielding process is executed with the input by the process manager. This process results in generation of cask-specific dose rates and uncertainty results that are extracted and stored for future use.

2.4.5 Containment Analysis

The cask containment analysis process requires cask type, cask inventory characteristics, and an evaluation date to execute. The process manager aggregates these characteristics from the Unified Database and creates a parameter set. The fuel assembly inventory depletion and decay prerequisite processes are executed for each assembly in a given cask if the results are not available at the evaluation date. The containment template is acquired from the UNF-Templates repository and provided to the TemplateEngine with the parameter set. The TemplateEngine replaces all the variables in the containment template by their respective values and in terms solve the containment analysis equations to calculate the allowable leakage rate. This process generates cask-specific allowable leakage rate results that are extracted from the expanded template and stored in the Unified Database for future use.

3.1 Graphical User Interface (GUI)

The graphical user interface (GUI) contains the following components (Fig. 3.1, left panel):

- Analysis Tab - UNF-ST&DARDS analysis capabilities accessible from this GUI form.
- Exports Tab - Commonly requested datasets available in tabular format from this GUI form.
- Pad Maps Tab - Geographic Information System (GIS)-Unified Database integrated results navigator.
- SQL Viewer Tab - Structured query language input and results form for native read-only Unified Database access.
- Tables Tab - Some consolidated Unified Database information in tabular format.

Menu and help buttons are on the top left corner of the GUI. Except for the “change user password” option the other items in the menu drop-down list are under development. “Change user password” can be used to change the password for the Unified Database. The help button provides UNF-ST&DARDS help. In addition to the description of the GUI, the help document provides (1) a description of the configuration file required to run UNF-ST&DARDS, (2) a description of the Unified Database (provided in Section 3.1) with the list and description of all the tables in the Unified Database, and (3) a description of templates (Section 2.2) and the template repository (Appendix A). The analysis tab, pad maps tab, and SQL viewer tab are described below. The exports and tables tabs will be added in a future revision of this manual.

3.1.1 Analysis Tab

The analysis tab is shown in Fig. 3.1. The analysis tab allows for selection of an evaluation type and site or reactor (PWR and boiling water reactor [BWR]) type-specific analysis. Currently the only analysis type activated is the bounding analysis, which refers to analysis

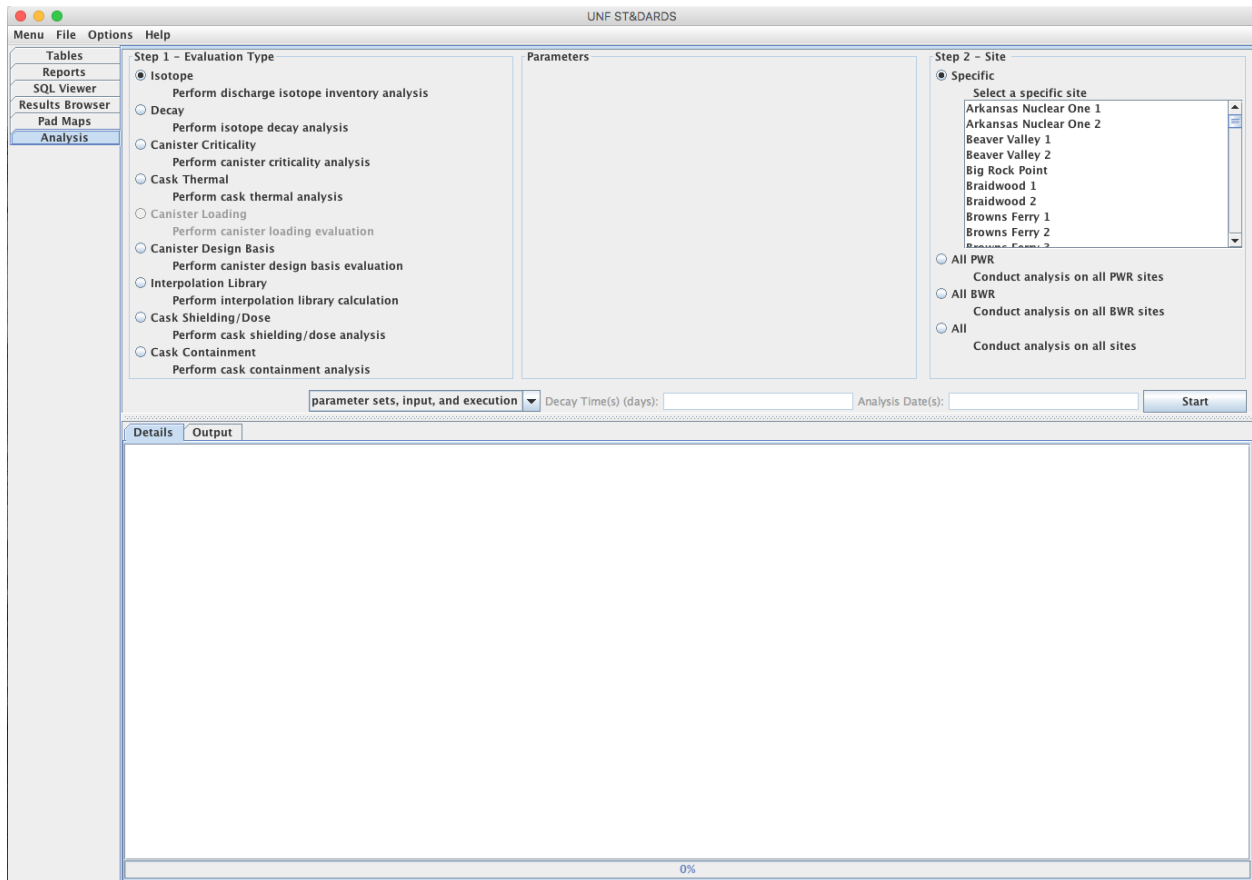


Fig. 3.1: UNF-ST&DARDS GUI.

with conservative reactor irradiation parameters that harden the neutron energy spectrum. Nominal (using nominal reactor irradiation parameters) and detailed (using detailed reactor irradiation parameters) analysis types will be implemented in future.

All analyses can have their execution modes changed to generate the template parameter sets, parameter sets and input, or parameter sets, input, and execution of the analysis application. This can be useful in conducting test runs of a desired analysis and inspecting the analyses parameters or inputs.

When conducting parameter set and input-only execution, the inputs generated may utilize surrogate information, as the actual input would require execution of a prerequisite process. For example, “parameter set” and “input only” execution of canister criticality will generate required isotope discharge and decay parameter sets and inputs. However, it will not execute these inputs, so the canister criticality input being supplemented with surrogate input. This is due to the required isotopic concentration input sections not being generated by the prerequisite decay analyses because no execution of the analysis applications are performed. The start button is used to run the selected analysis type.

Evaluation Type: Isotope

The isotope evaluation type allows for generation of assembly-specific discharged isotopic libraries. These libraries can be used for subsequent assembly isotope decay analysis. These analyses require the precalculated appropriate assembly type interpolation libraries. The analysis scope can be reduced by deselecting the run checkbox for the listed assemblies for which analysis is not desired. [Fig. 3.2](#) shows an isotope analysis option for two selected assemblies (LM01B8 and LM01B9) from the Beaver Valley 1 reactor site. The two assemblies are selected by deselecting all the other assemblies in the run checkbox.

Evaluation Type: Decay

The decay evaluation type is used for the generation of assembly-specific decayed isotopic libraries. These libraries require the corresponding pre-generated discharged isotopic inventory (discussed in Section 4.1.1) libraries. If the discharge libraries do not exist that UNF-ST&DARDS will conduct the pre-requisite discharged isotopic inventory library analyses. The generated assembly-specific decayed isotopic inventory libraries can subsequently be used for criticality, shielding, thermal, and containment analyses. The decay evaluation type requires a specific evaluation date or set of dates (comma delimited) of the format *MM/DD/YYYY*. The analysis scope can be reduced by deselecting the run checkbox for listed assemblies for which analysis is not desired. Results of the decay calculations are stored in the *Decay_Concentration_Summary_bounding* table in the unified database.

[Fig. 3.3](#) shows the decay evaluation type for all the discharged assemblies at Maine Yankee site for two specified decay dates.

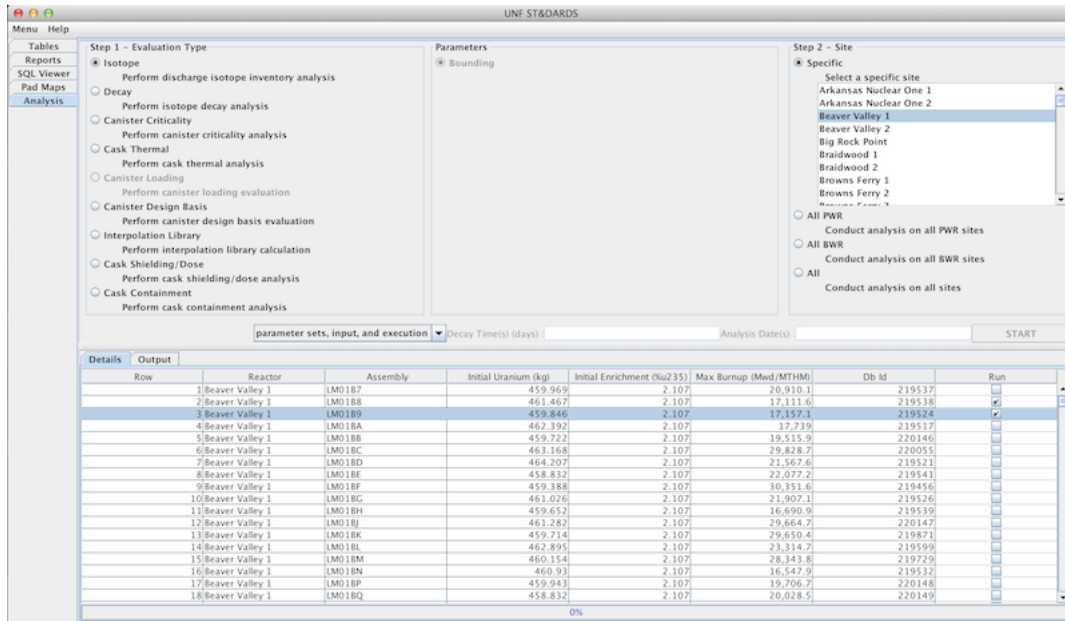


Fig. 3.2: Discharge isotope analysis for two selected assemblies from Beaver Valley 1 reactor site.

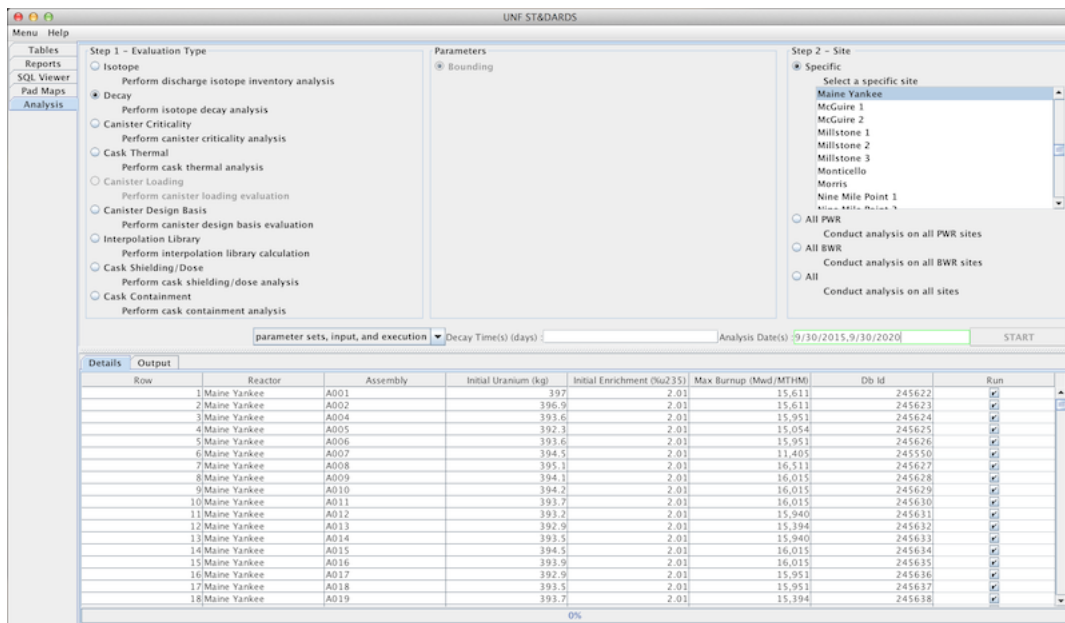


Fig. 3.3: Decay evaluation type for all the discharged assemblies at the Maine Yankee site for two specified dates.

Evaluation Type: Canister Criticality

The canister criticality evaluation type is used for as-loaded, canister-specific criticality safety analysis. These calculations require a significant amount of time to run and potentially large amounts of system resources. This evaluation type requires a specific evaluation date or set of dates (comma delimited) of the format *MM/DD/YYYY*. In addition, the keyword *inservice* can be used to conduct analysis for a canister's inservice date. The analysis scope can be reduced by deselecting the run checkbox for listed canisters for which analysis is not desired.

For criticality analysis, the decayed isotopic inventories for the requested analysis dates are required to be pregenerated for all of the assemblies in the selected canisters. If the decayed isotopic inventories do not exist, UNF-ST&DARDS will conduct the prerequisite assembly-specific decayed isotopic inventory analysis before conducting the criticality analysis. Upon completion of analyses, UNF-ST&DARDS will extract the k-effective and uncertainty values and import them into the Unified Database (*canister_criticality_bounding* table) for future analysis.

Fig. 3.4 presents the criticality analysis type for two specified dates.

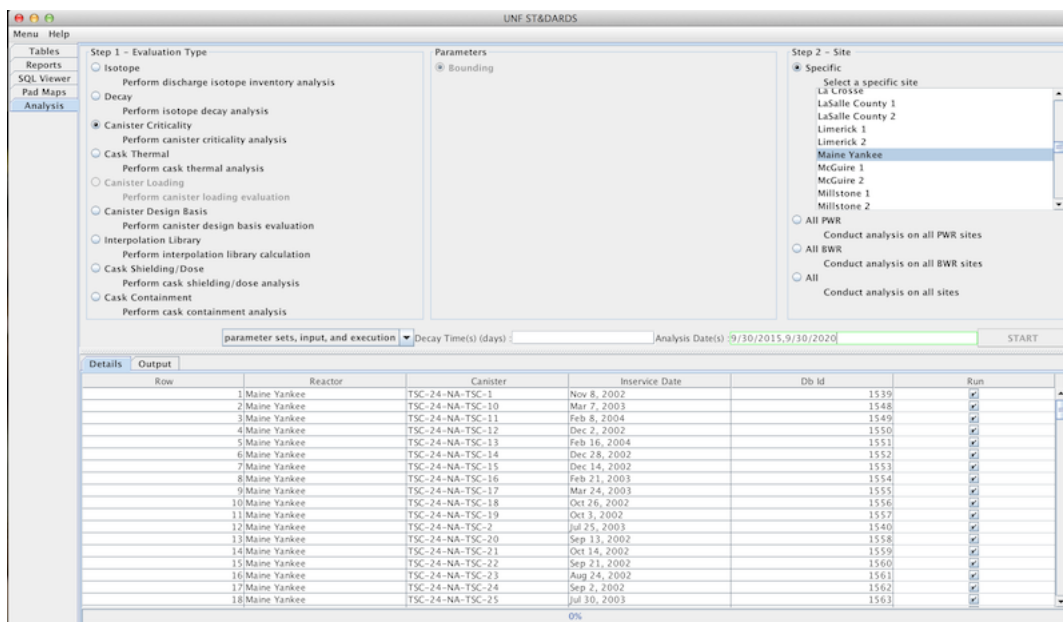


Fig. 3.4: Criticality analysis type for all the canisters at Maine Yankee site for two specified dates.

Evaluation Type: Cask Thermal

The thermal evaluation type is used to generate assembly-specific peak and minimum clad temperatures and cask-specific surface temperatures and total decay heat results. These

calculations require a significant amount of time to run. This evaluation type requires a specific evaluation date or set of dates (comma delimited) of the format *MM/DD/YYYY*. In addition, the keyword *inservice* can be used to conduct analysis for a canister's inservice date. The analysis scope can be reduced by deselecting the run checkbox for listed canisters for which analysis is not desired.

For thermal analysis, the decayed isotopic inventories for the requested analysis dates are required to be pregenerated for all the assemblies in the selected canisters. If the decayed isotopic inventories do not exist, UNF-ST&DARDS will conduct the prerequisite assembly-specific decayed isotopic inventory analysis before conducting the thermal analysis. Upon completion of analyses, UNF-ST&DARDS will extract the peak and minimum clad and surface temperature values and import them into the Unified Database (*cask_thermal_bounding* table) for future analysis. In the GUI, the thermal evaluation type appears similar to the criticality evaluation type presented in [Fig. 3.4](#).

Evaluation Type: Canister Design Basis

The canister design basis evaluation type is used for the design basis criticality analysis using a uniform loading map (same burnup and enrichment in all canister locations). A step-by-step description of canister design basis calculation is provided below.

1. Select a canister model (see [Fig. 7](#)) from the drop-down menu. (Note: Not all canister models have canister input templates, so some may not work).
2. Select the assembly type from the drop-down menu to populate the canister.
3. Enter a desired 235U enrichment for the assemblies.
4. Enter the burnup (megawatt days per metric ton of uranium [MWd/MTU]) to be applied to each assembly.
5. Enter a unique canister name.
6. Enter a specific power (megawatt per metric ton of uranium [MW/MTU]) to be applied to each assembly.
7. Indicate whether it is desired to reuse discharge f71, which will reuse any assembly-discharged isotopic inventory libraries previously generated.
8. Indicate if it is desired to reuse decayed values which will reuse any assembly decayed isotopic inventory data previous generated.
9. Indicate the canister's inventory count. When this inventory count is specified, the canister inventory table is populated for inspection.
10. Specify the decay time(s) (days). If the analysis date(s) is specified, the decay time is computed as the difference (in days) from the day of analysis to the analysis date(s). The analysis date(s) cannot be prior to the day of analysis (i.e., negative decay time).
11. Press the START button to conduct the requested analysis.

Note that the inventory can be removed by specifying an inventory count of 0, changing the select parameters, and entering and re-entering inventory count. Fig. 3.5 presents the canister design basis evaluation type.

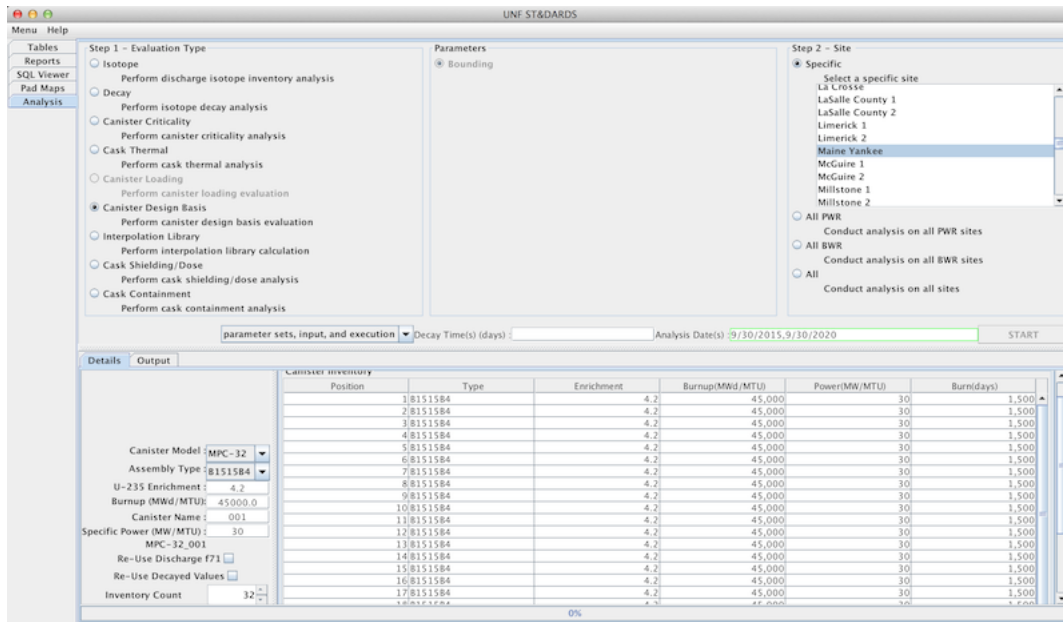


Fig. 3.5: Canister design basis evaluation type.

Evaluation Type: Interpolation Library

The interpolation library evaluation type is used for the generation of cross section libraries specific to the assembly type. This analysis can be conducted with the following steps:

1. Select the library type from the drop-down list (see Fig. 3.6) of assembly types (Note: Not all assembly types have depletion input templates, so analysis may fail at input generation.)
2. Enter the desired enrichment(s).
3. Press the START button to commence analysis.

Upon completion of the application, the generated library is copied back to the appropriate arpLib resource directory. Note that the final step in the interpolation library generation is the update of the arplib.txt located at the location that is pointed to by the appropriate arpLib resource. The interpolation library generation requires significant computation resources. It is advised to run only a single process at a time. Fig. 3.6 presents the interpolation library evaluation type as can be seen in the GUI.

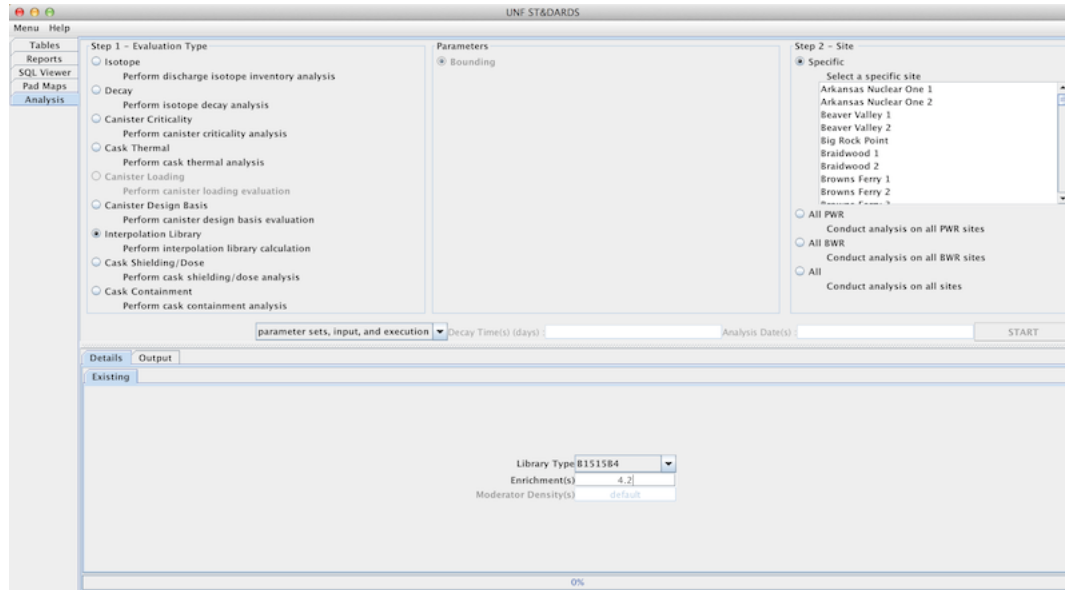


Fig. 3.6: Interpolation library evaluation type.

Evaluation Type: Cask Shielding/Dose

The shielding evaluation type is used for the generation of canister-specific shielding analysis for the transportation cask. These calculations require a significant amount of time to run. This evaluation type requires a specific evaluation date or set of dates (comma delimited) of the format *MM/DD/YYYY*. In addition, the keyword *inservice* can be used to conduct analysis for a canister's inservice date. The analysis scope can be reduced by deselecting the run checkbox for listed canisters for which analysis is not desired. For shielding analysis, the decayed isotopic inventories for the requested analysis dates are required to be pregenerated for all the assemblies in the selected canisters. If the decayed isotopic inventories do not exist, UNF-ST&DARDS will conduct the prerequisite assembly-specific decayed isotopic inventory analysis before conducting the thermal analysis. Upon completion of analyses, UNF-ST&DARDS will extract the dose rate values on and near the surface of the transportation cask and import them into the Unified Database (*dose_hac_bounding* and *dose_nct_bounding* tables) for future analysis. In the GUI, the shielding evaluation type appears similar to the criticality evaluation type presented in Fig. 3.4.

Evaluation Type: Cask Containment

Containment evaluation type is used for canister-specific containment analysis of a transportation package. This evaluation type requires a specific evaluation date or set of dates (comma delimited) of the format *MM/DD/YYYY*. In addition, the keyword *inservice* can be used to conduct analysis for the canister's inservice date. The analysis scope can be reduced by deselecting the run checkbox for listed canisters for which analysis is not desired. For containment analysis, the decayed isotopic inventories for the requested analysis

dates are required to be pregenerated for all the assemblies in the selected canisters. If the decayed isotopic inventories do not exist, UNF-ST&DARDS will conduct the prerequisite assembly-specific decayed isotopic inventory analysis before conducting the containment analysis. Upon completion of analyses, UNF-ST&DARDS will extract the allowable leakage rate values of the transportation cask and import them into the Unified Database (*containment_bounding* tables) for future analysis. In the GUI, the containment evaluation type appears similar to the criticality evaluation type presented in Fig. 3.4.

3.1.2 Pad Maps

The pad maps tab provides information for each site. It provides a GIS-integrated navigation panel, allowing quick site-specific viewing of results. Fig. 3.7 presents the pad map view.

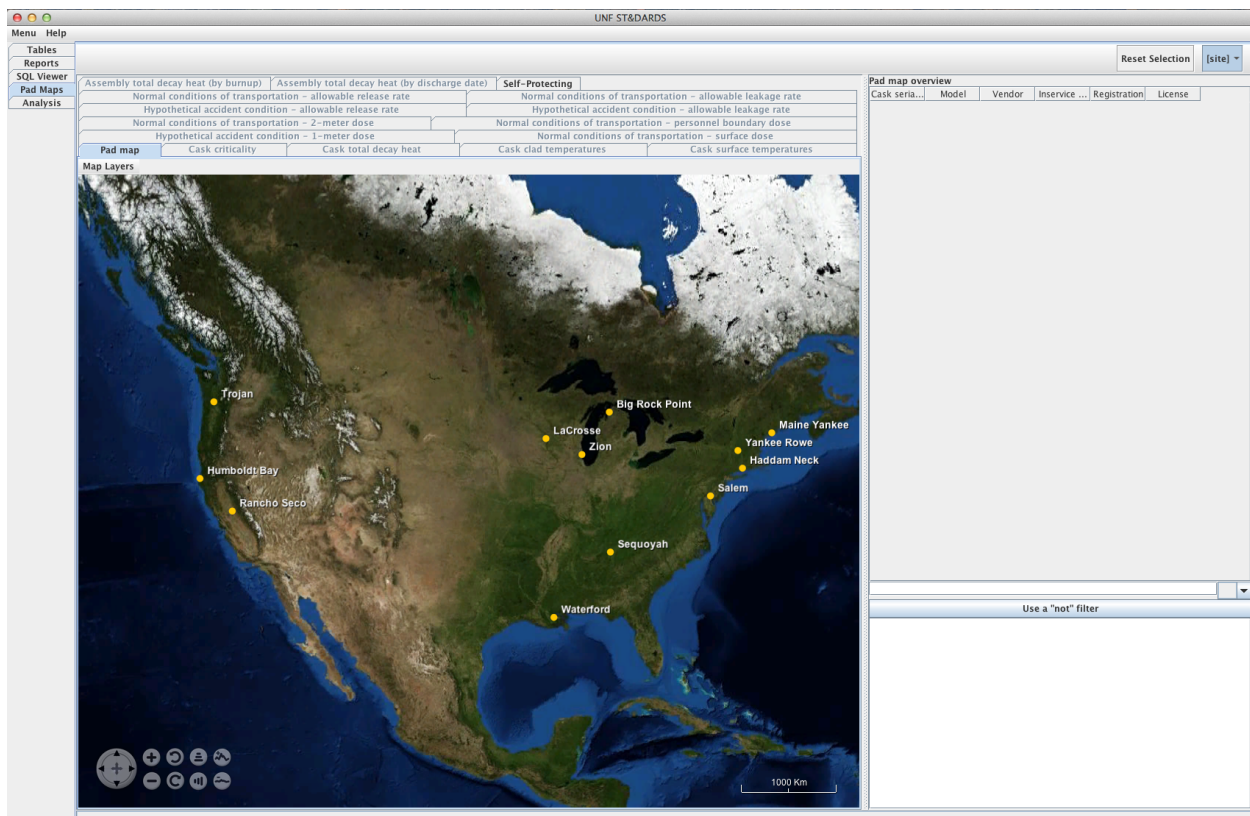


Fig. 3.7: Pad map view.

Typical GIS controls are available in the map view:

- click and drag to pan, or click the pan arrows in the lower left corner,
- double click to center map on clicked point,

- scroll in and out to zoom in and out respectively, or click the + and - buttons in the lower left corner,
- hover over a site to display site information in a popup bubble or site information in the information box in lower right corner,
- use the controls in the lower left corner for perspective rotation (left and right) and angle controls, as well as landscape relief increase and decrease, and
- at any point during navigation, reset the view by clicking the reset selection button in the upper right corner.

Nationwide assembly self-protection status (as shown in Fig. 3.8) can also be viewed from the global pad map view by clicking the **Self-Protecting** tab on the upper right corner. The Self-Protecting tab allows user to change the self-protection threshold (dose above which the assemblies are considered as self-protected) and recalculate the self-protection status. The default threshold is set to 100 Rem. Note that in the global pad map view all the other tabs are disabled and they can be only viewed by selecting a site. The following figure presents the nationwide self-protecting status as can be seen by selecting the Self-Protecting tab from the global pad map view.

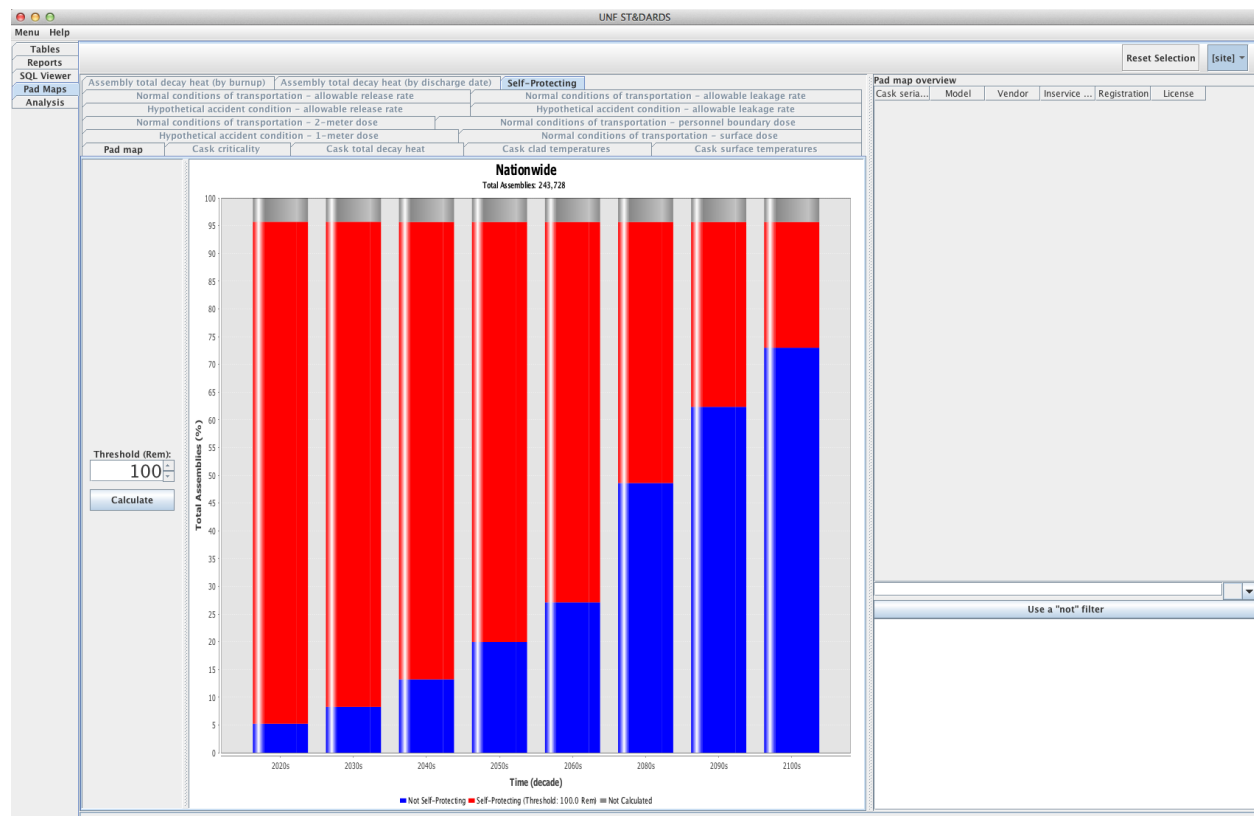


Fig. 3.8: Self-protecting status of assemblies discharged nationwide.

Clicking the pad maps (site) breadcrumb in the upper right corner provides a list of sites

to choose from. Clicking a site auto-zooms the map perspective to the site's dry storage pad, allowing for inspection of the site's dry storage cask inventory, as shown in Fig. 3.9.

The screenshot displays the UNF-ST&DARDS software interface. At the top, there is a menu bar with 'Menu Help' and a toolbar with 'Reset Selection', 'Maine Yankee', and '[cask]'. Below the toolbar are several tabs for analysis: 'Assembly total decay heat (by burnup)', 'Assembly total decay heat (by discharge date)', 'Self-Protecting', 'Normal conditions of transportation - allowable release rate', 'Hypothetical accident condition - allowable leakage rate', 'Normal conditions of transportation - allowable leakage rate', 'Hypothetical accident condition - 1-meter dose', 'Normal conditions of transportation - 2-meter dose', 'Normal conditions of transportation - personnel boundary dose', 'Hypothetical accident condition - 1-meter dose', and 'Normal conditions of transportation - surface dose'. A 'Map Layers' section is also visible. The central part of the interface is an aerial map showing a site with a large rectangular pad containing several rows of casks. A scale bar indicates 50 meters. On the right side, there is a 'Pad map overview' table with columns for 'Cask serial number', 'Model', 'Vendor', 'Inservice date', 'Registration', and 'Lic'. Below the table is a section titled 'Use a "not" filter' with details for the 'Maine Yankee Site', including 'Decommissioning Completed', 'Maine Yankee' information (Initial Operations Date: 06/29/1973, License Expiration Date: 12/06/1996, Net Electric Capacity: N/A, Thermal Capacity: N/A, Reactor Type: PWR, Nuclear Steam System (NSSS) CE, Assembly Class: CE 14x14), and 'Maine Yankee Wet Storage' information (Maine Yankee ISFSE, NAC-UMS UMS-24, Vendor: NAC International, Inc., Loaded Canisters: 64, Loaded Assemblies: 1434, Position: Vertical, Coc: 72-1015).

Fig. 3.9: Pad view of a site.

In the site's pad view, hovering the mouse over a cask will provide the cask's identification in a popup bubble. Available details can be viewed by clicking the cask. Clicking the pad maps (cask) breadcrumb in the upper right corner will provide a list of site-specific casks to choose from. The tabs described below are available for each analyzed site.

Cask criticality

The criticality view allows viewing of all site-specific cask criticality results as shown in Fig. 3.10. On the left is the series name listing of all plotted series, with a check box allowing for toggle of the series display. The cask criticality plot is in the center. The cask criticality plot interactions are described in the plots section. The pad map overview is on the right and lists the casks on the pad. Right clicking a row in the pad map overview list provides a context menu which allows for viewing the details of the selected cask or saving the table as a comma separated value (csv) formatted file to a location of the user's choice.

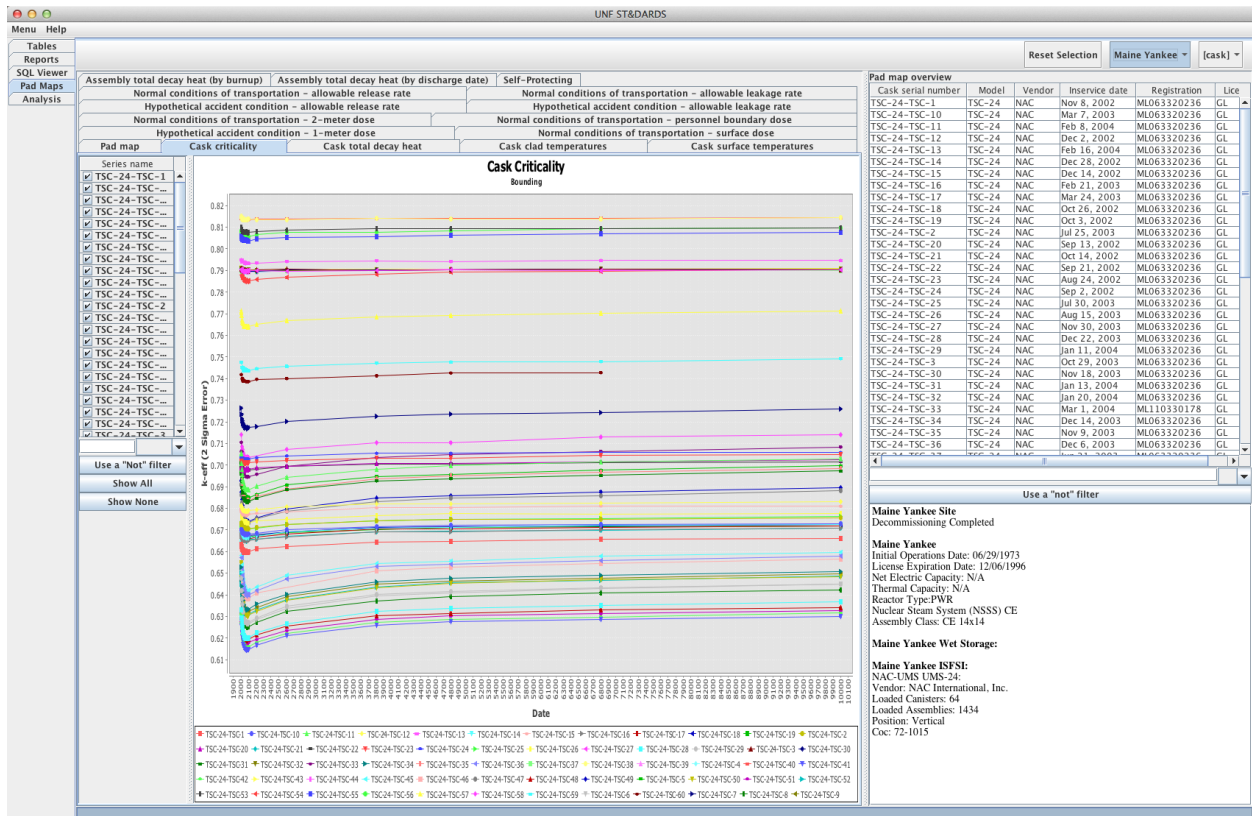


Fig. 3.10: Criticality results for all the canisters of a site.

Cask total decay heat

The total decay heat tab provides a site-wide canister-specific total decay heat view, as shown in Fig. 3.11. On the left is the series name listing of all plotted series, with a check box allowing for toggle of the series display. The total decay heat plot is in the center. The total decay heat plot interactions are described in the plots section. The pad map overview is on the right and lists the casks on the pad. Right clicking a row in the pad map overview list provides a context menu, which allows the user to view the details of the selected cask or to save the table as a comma-separated value (csv) formatted file to a location of the user's choosing.

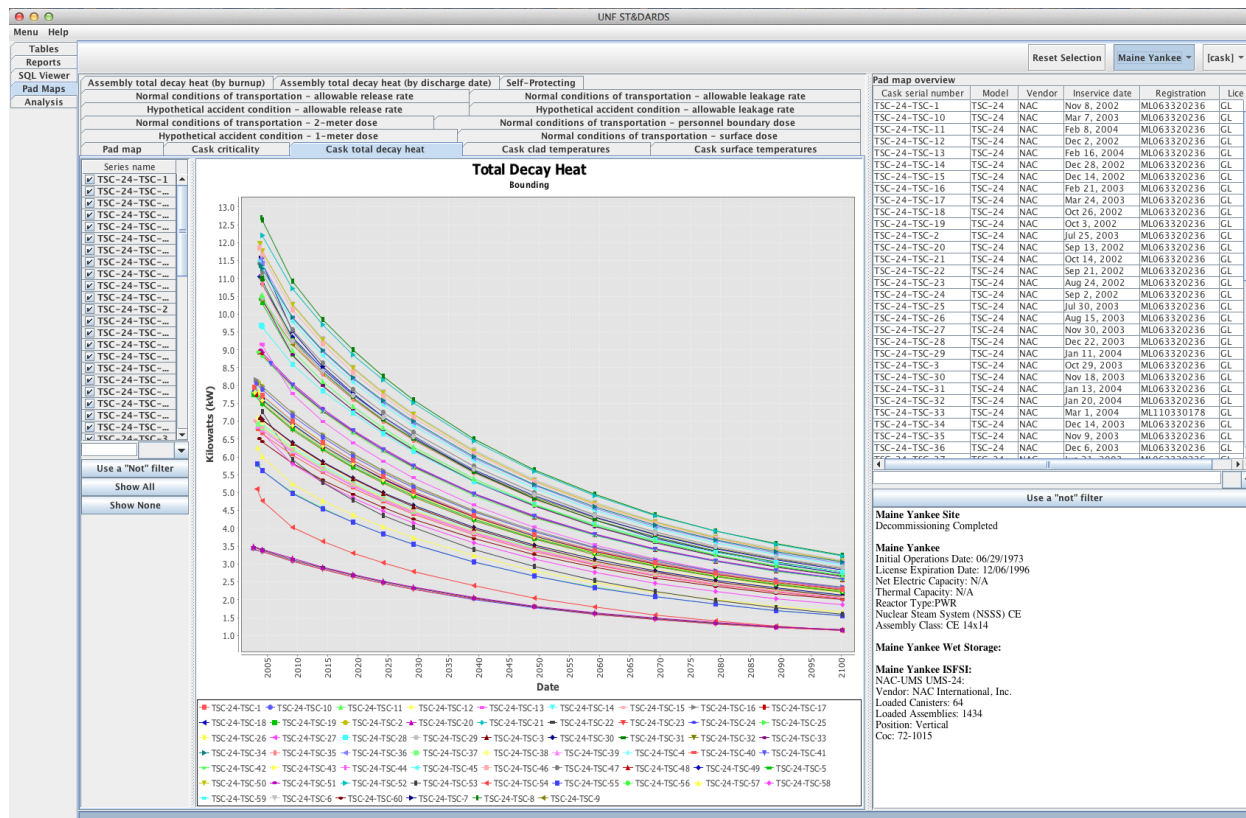


Fig. 3.11: Total decay heat results for all the canisters of a site.

Cask clad temperatures

The canister cladding temperatures provide site-wide canister-specific cladding temperatures, as shown in Fig. 3.12. The Y-axis displays the cladding temperature (degrees Fahrenheit), and the X-axis shows the evaluation date.

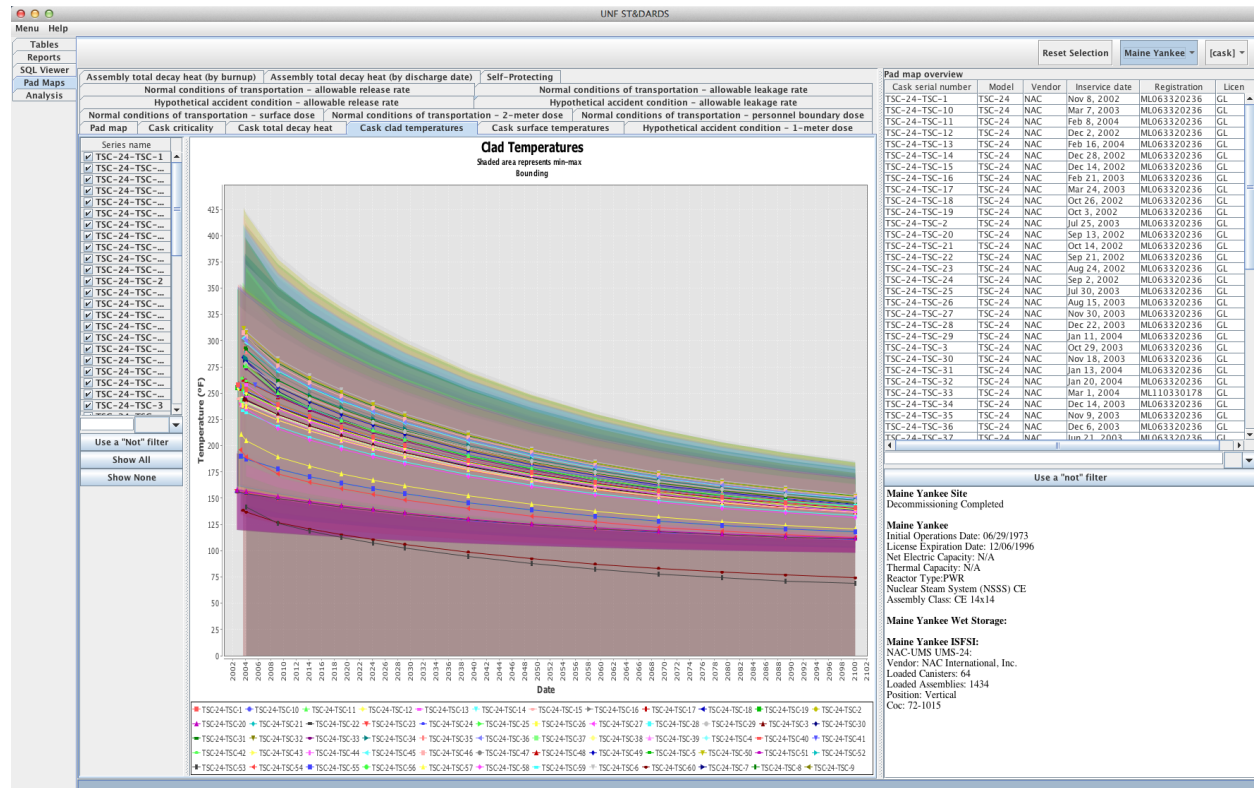


Fig. 3.12: Total Clad temperature results for all the casks of a site.

Cask surface temperatures

The canister surface temperature results provide site-wide canister-specific surface temperatures, as shown in Fig. 3.13. The Y-axis displays the surface temperature (degrees Fahrenheit), and the X-axis shows the evaluation date.

Hypothetical accident condition – 1-meter dose

The canister hypothetical accident condition 1-meter dose provides the site-wide canister-specific (canister in its designated transportation cask) hypothetical accident condition one-meter dose rates as shown in Fig. 3.14. The Y-axis displays the dose rate (mrem/hr), and the X-axis presents the evaluation date.

Normal condition of transportation – surface dose

The canister normal conditions of transportation surface dose rate results provide site-wide canister-specific (each canister in their designated transportation cask) normal conditions of transportation surface dose rates, as presented in Fig. 3.15. The Y-axis displays the dose rate (mrem/hr), and the X-axis shows the evaluation date.

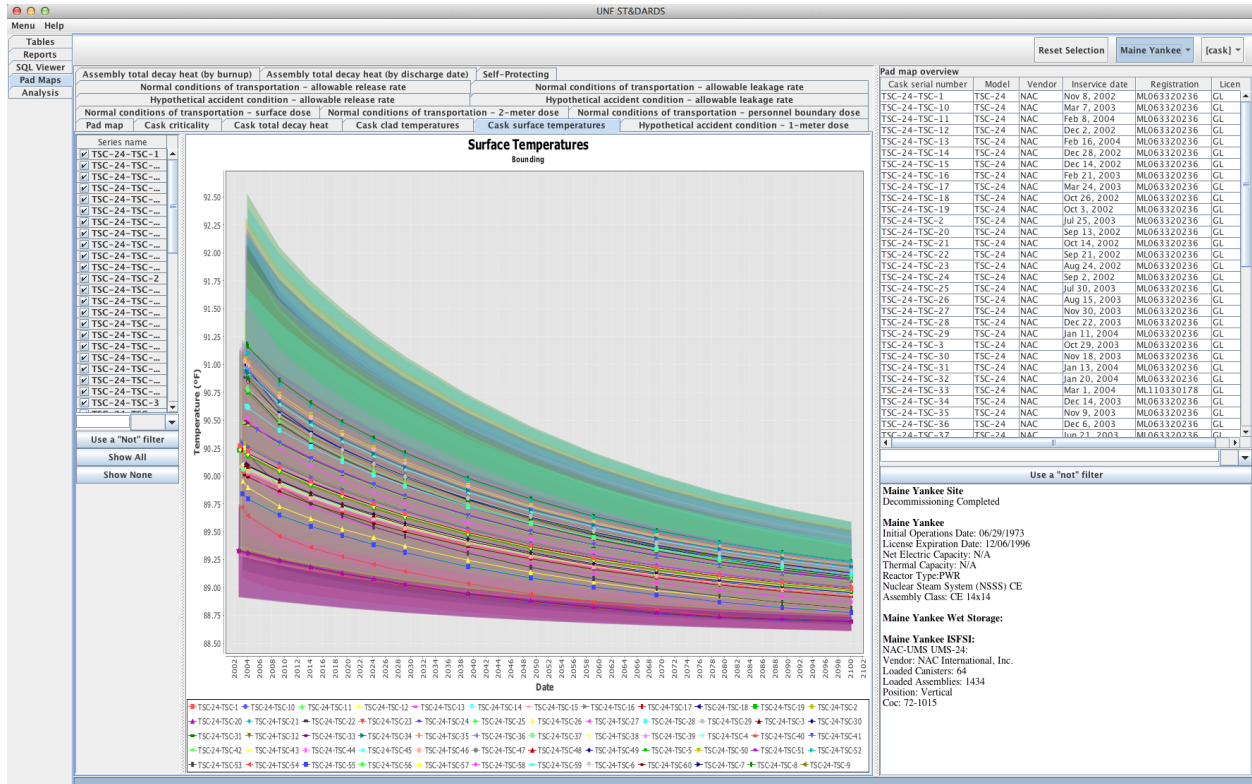


Fig. 3.13: Surface temperature results for all the casks of a site.

Normal condition of transportation – 2-meter dose

The canister normal conditions of transportation 2-meter dose rate results provide site-wide canister-specific (each canister in their designated transportation cask) normal conditions of transportation 2-meter dose results, as shown in Fig. 3.16. The Y-axis displays the dose rate (mrem/hr), and the X-axis shows the evaluation date.

Normal condition of transportation – personnel boundary dose

The canister normal conditions of transportation personnel boundary dose rate results provide site-wide canister-specific (each canister in its designated transportation cask) normal conditions of transportation personnel boundary dose rates, as shown in Fig. 3.17. The Y-axis displays the dose rates (mrem/hr), and the X-axis shows the evaluation date.

Assembly total decay heat (by burnup)

The assembly total decay heat (grouped by burnup) provides site wide assembly-specific total decay heat as a function of time, as shown in Fig. 3.18. The Y-axis on the left displays the total assembly decay heat (watts/assembly). The color bar on the right displays the burnups, and the X-axis shows the evaluation date.

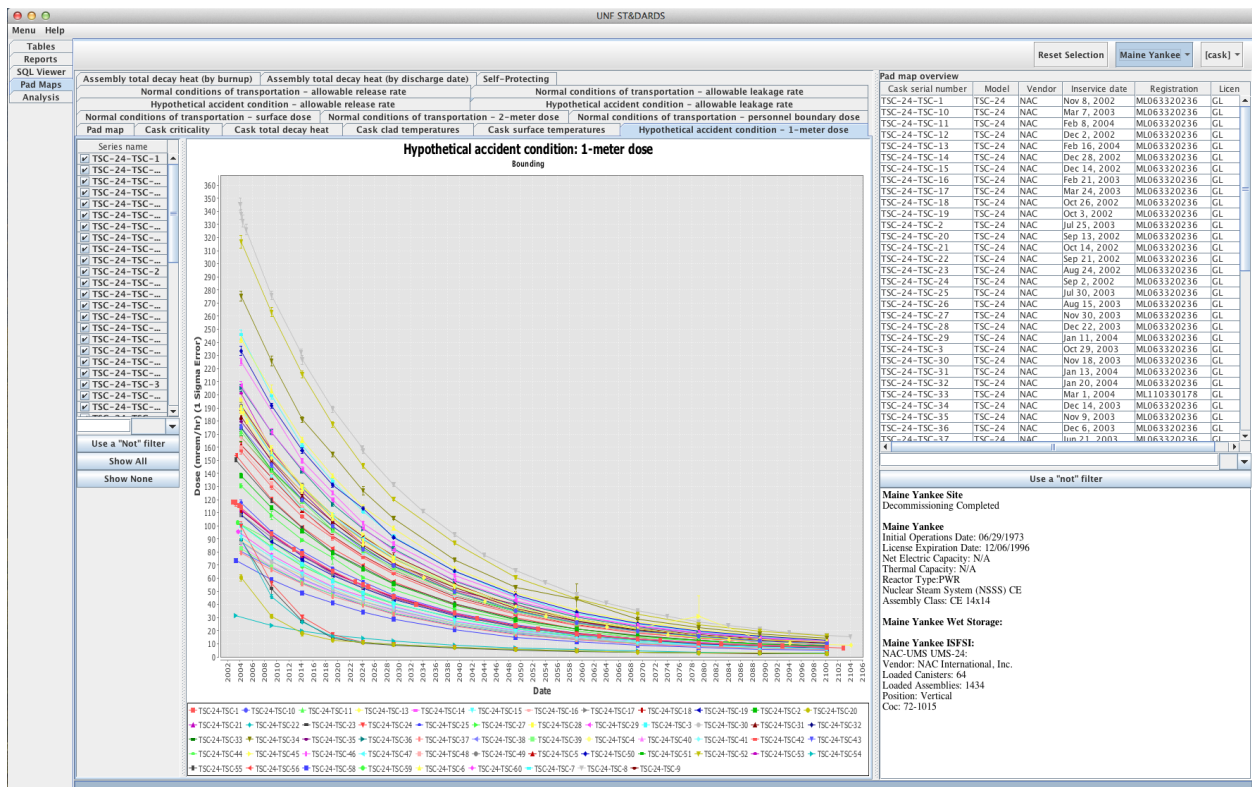


Fig. 3.14: Hypothetical accident condition 1-meter dose rates for all the canisters at a site in their designated transportation cask.

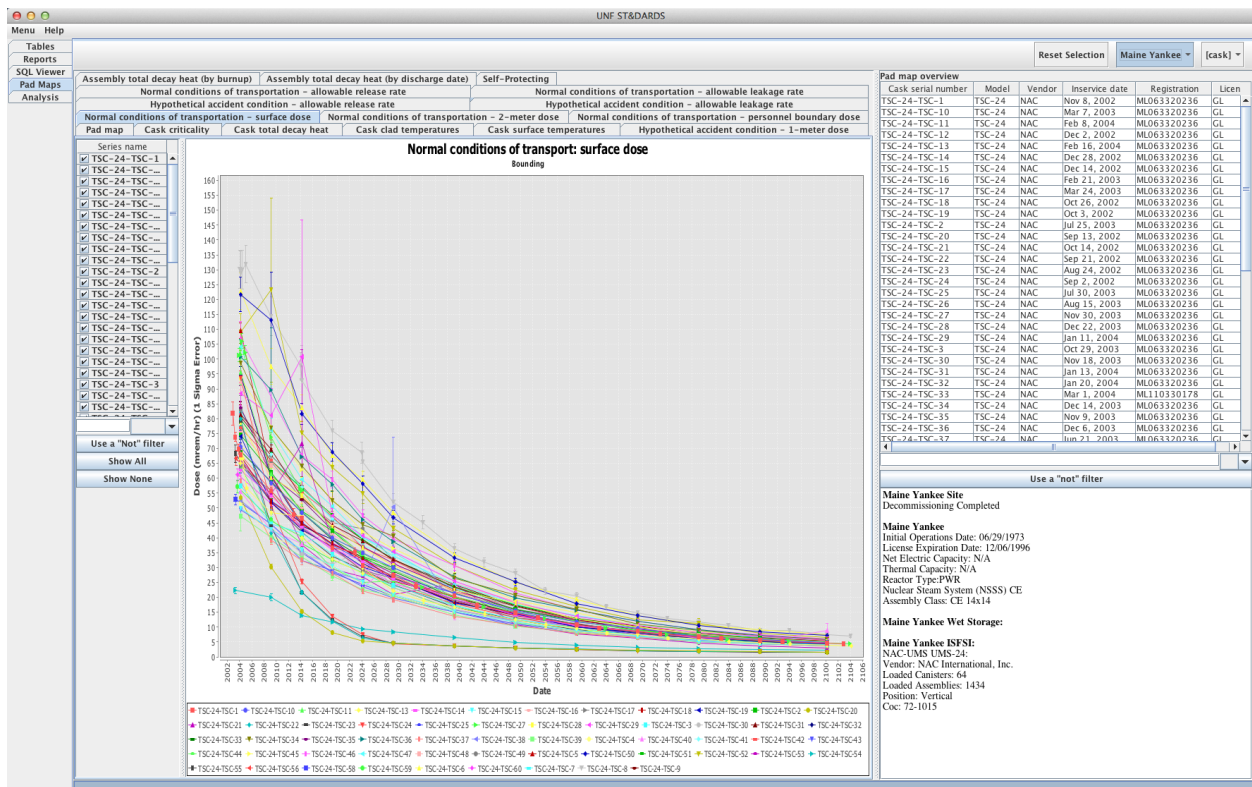


Fig. 3.15: Normal condition of transport surface dose rates for all the canisters at a site in their designated transportation cask.

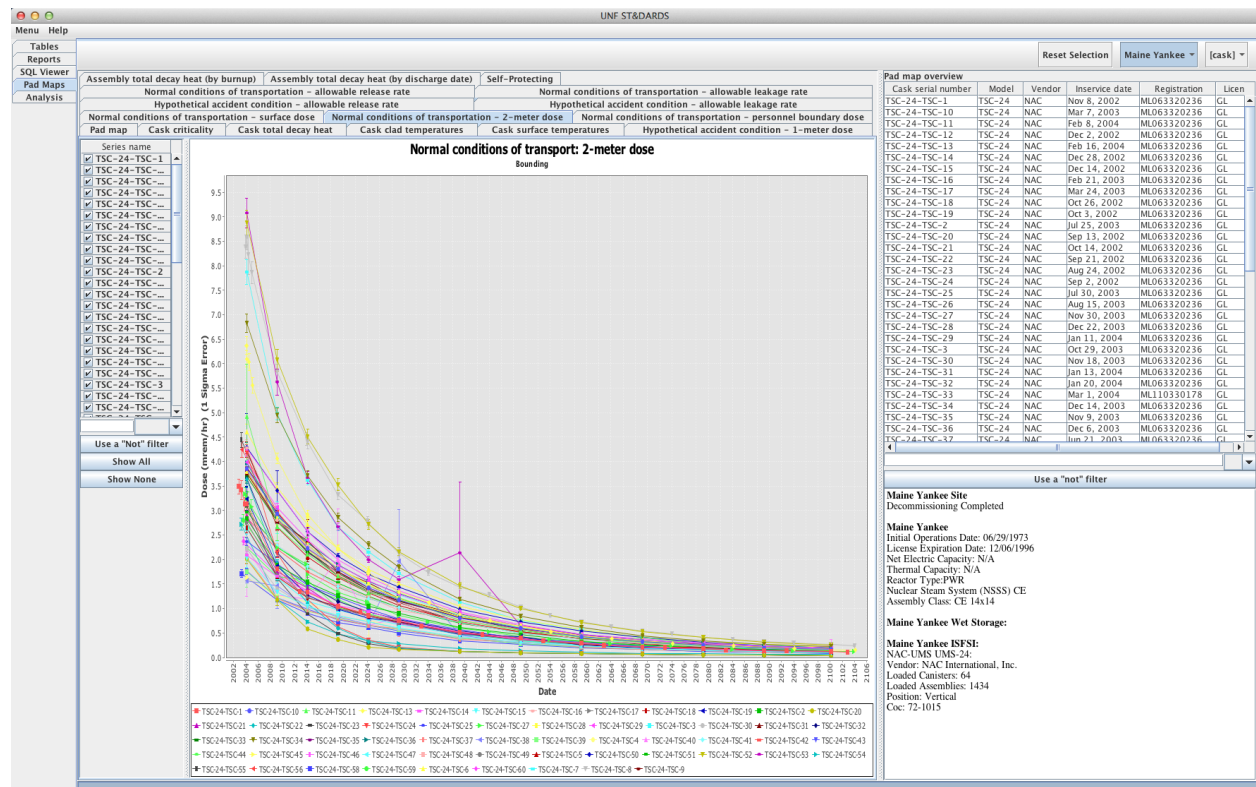


Fig. 3.16: Normal condition of transport 2-meter dose rates for all the canisters at a site in their designated transportation cask.

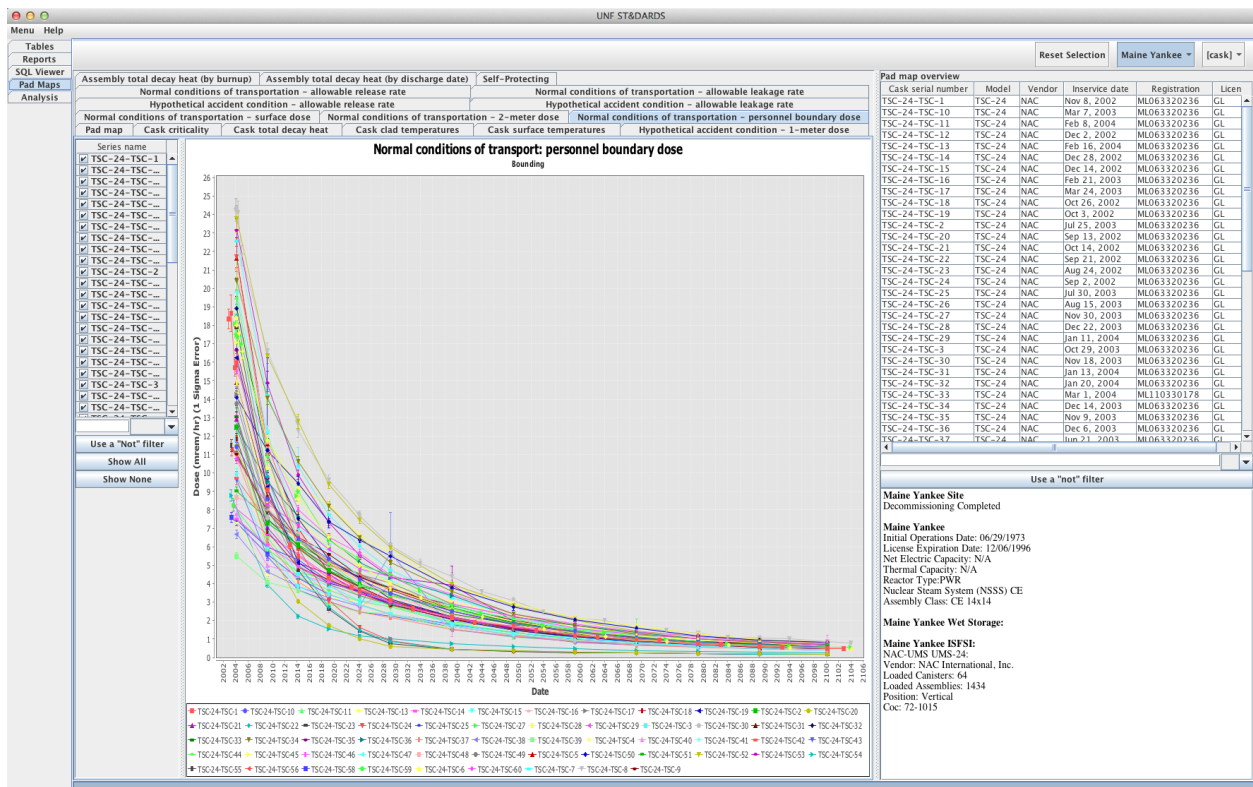


Fig. 3.17: Normal condition of transport personnel boundary dose rates for all the canisters at a site in their designated transportation cask.

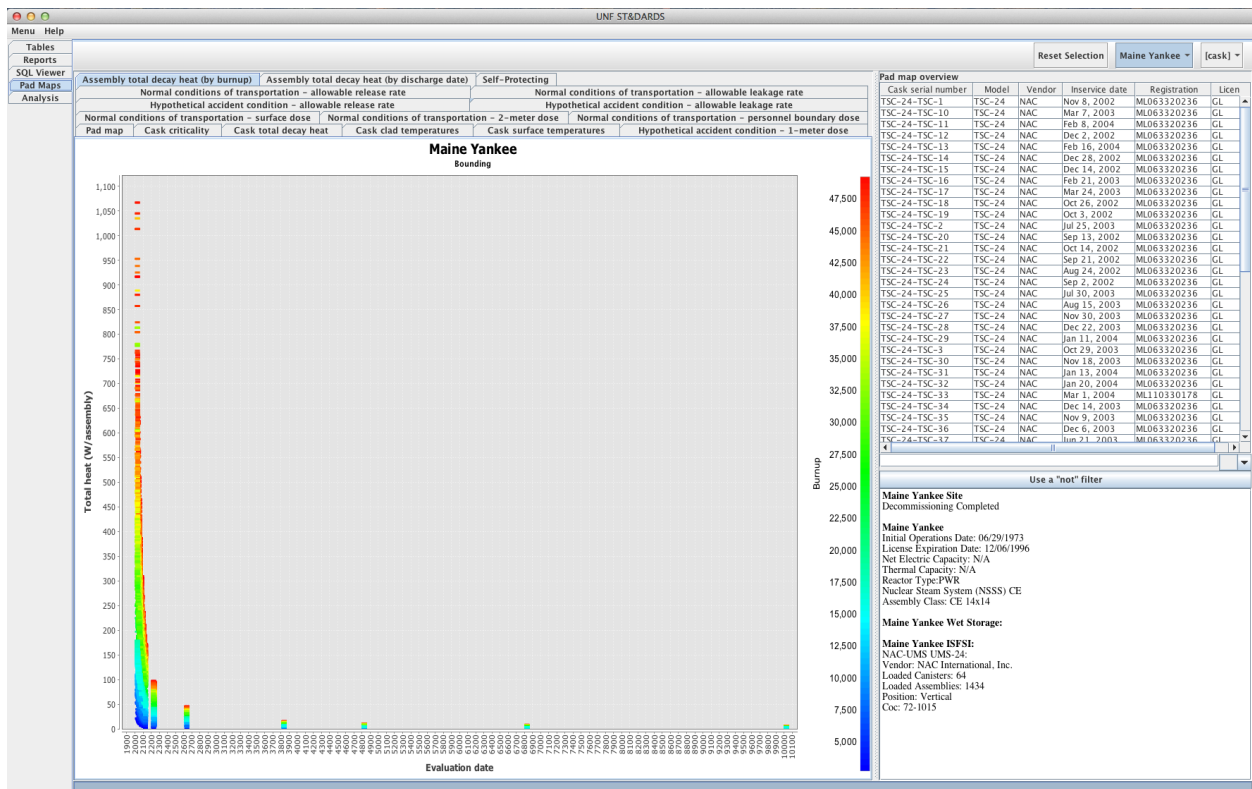


Fig. 3.18: The total decay heat as a function of time of all the assemblies at a site grouped by burnup.

Assembly total decay heat (by discharge date)

The assembly total decay heat (grouped by discharge date) provides site-wide assembly-specific total decay heat as a function of time, as shown in Fig. 3.19. The Y-axis on the left displays the total assembly decay heat (watts/assembly). The color bar on the right displays the discharge dates, and the X-axis shows the evaluation date.

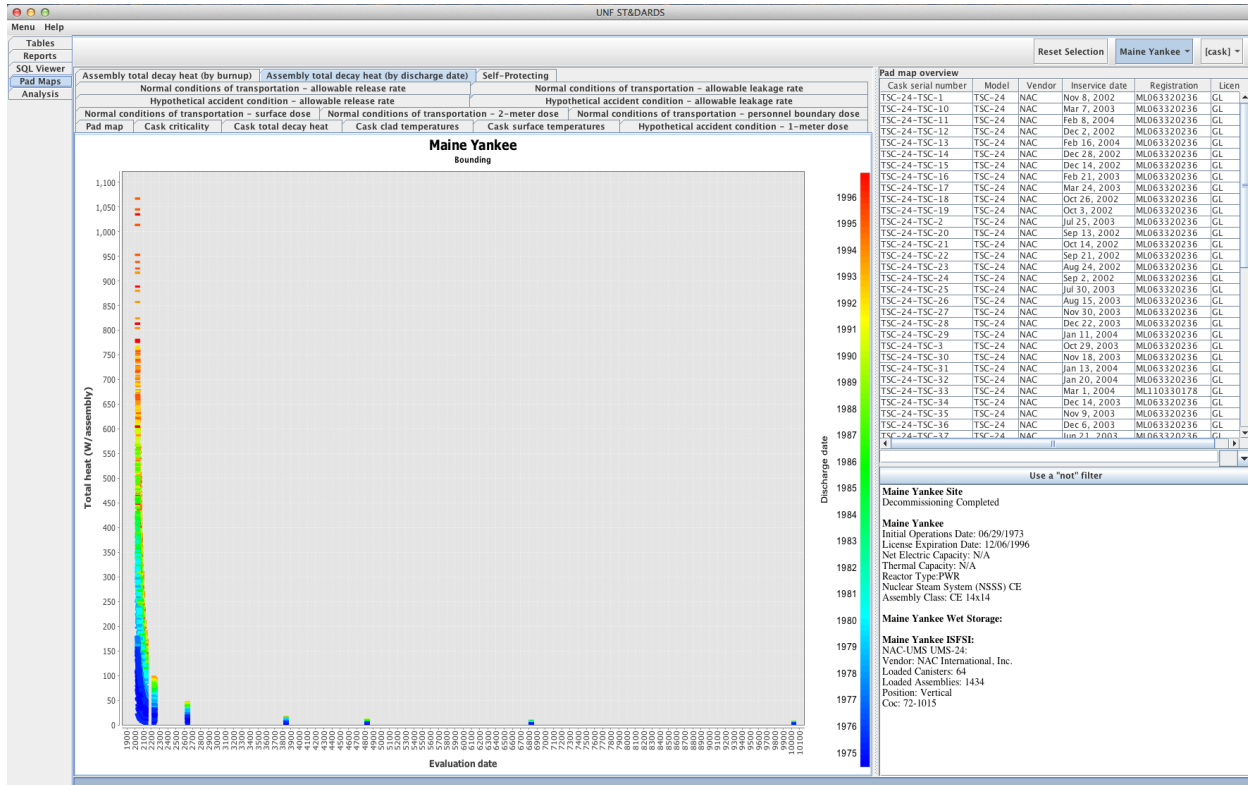


Fig. 3.19: The total decay heat as a function of time of all the assemblies at a site grouped by discharge date.

Normal condition of transport – allowable leakage rate

The canister normal conditions of transportation allowable leakage rate results provide site-wide canister-specific (each canister in their designated transportation cask) normal conditions of transportation allowable leakage rates as shown in Fig. 3.20. The Y-axis displays the allowable leakage rates (cc/sec), and the X-axis shows the evaluation date.

Normal condition of transport – allowable release rate

The canister normal conditions of transportation allowable release rate results provide site-wide canister-specific (each canister in their designated transportation cask) normal

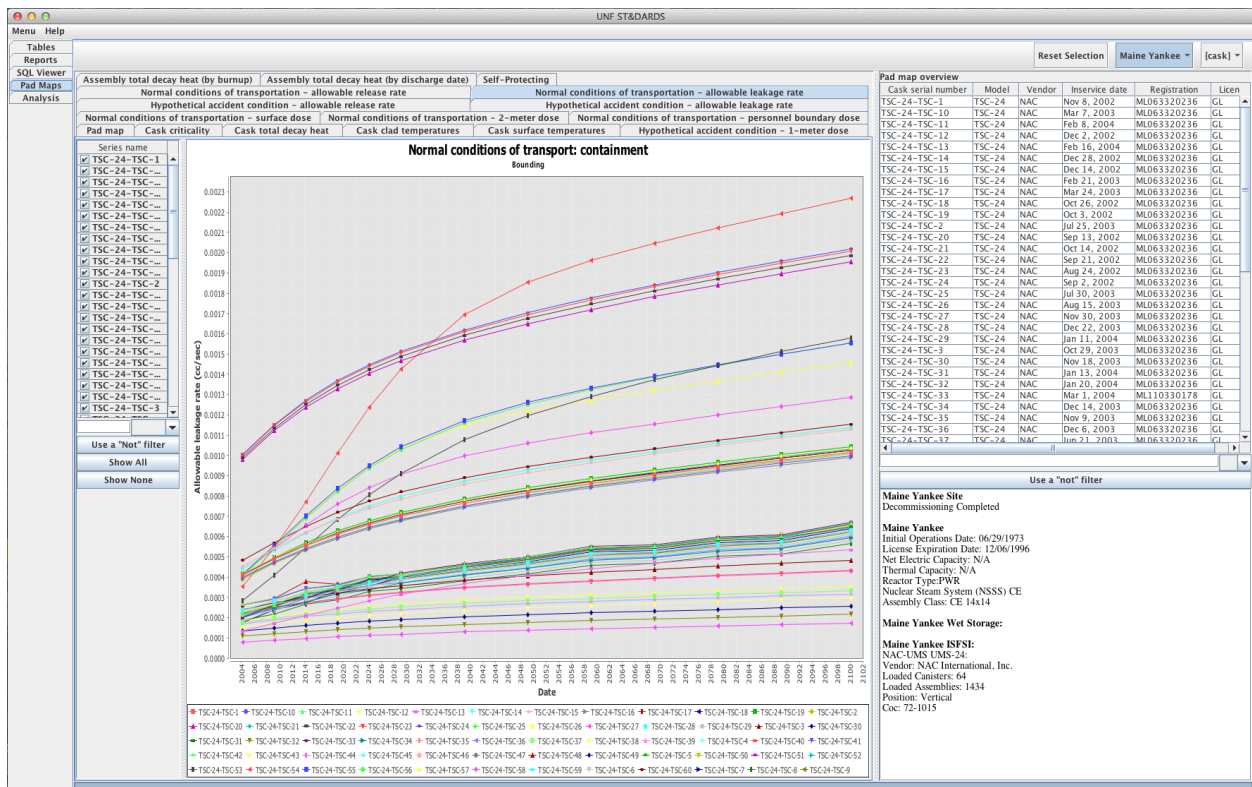


Fig. 3.20: Normal condition of transport containment results (allowable leakage rate) for all the canisters at a site in their designated transportation casks.

conditions of transportation allowable release rates, as shown in Fig. 3.21. The Y-axis displays the allowable leakage rates (curies/sec), and the X-axis shows the evaluation date.

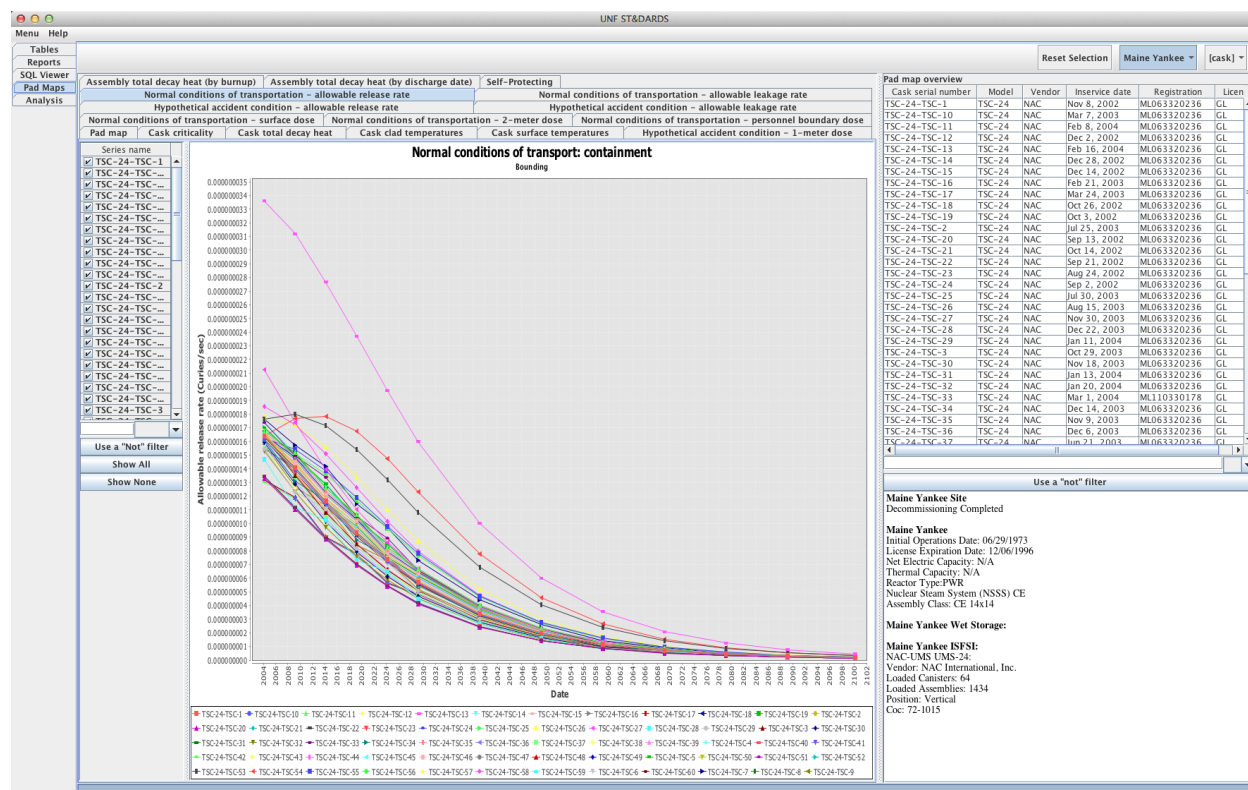


Fig. 3.21: Normal condition of transport containment (allowable release rate) results for all the canisters at a site in their designated transportation casks.

Hypothetical accident condition – allowable leakage rate

The hypothetical accident condition allowable leakage rate results provide site-wide canister-specific (each canister in their designated transportation cask) hypothetical accident condition allowable leakage rates, as shown in Fig. 3.22. The Y-axis displays the allowable leakage rates (cc/sec), and the X-axis shows the evaluation date.

Hypothetical accident condition – allowable release rate

The hypothetical accident condition allowable release rate results provide site-wide canister-specific (each canister in their designated transportation cask) hypothetical accident condition allowable release rates, as shown in Fig. 3.23. The Y-axis displays the allowable leakage rates (curies/sec), and the X-axis shows the evaluation date.

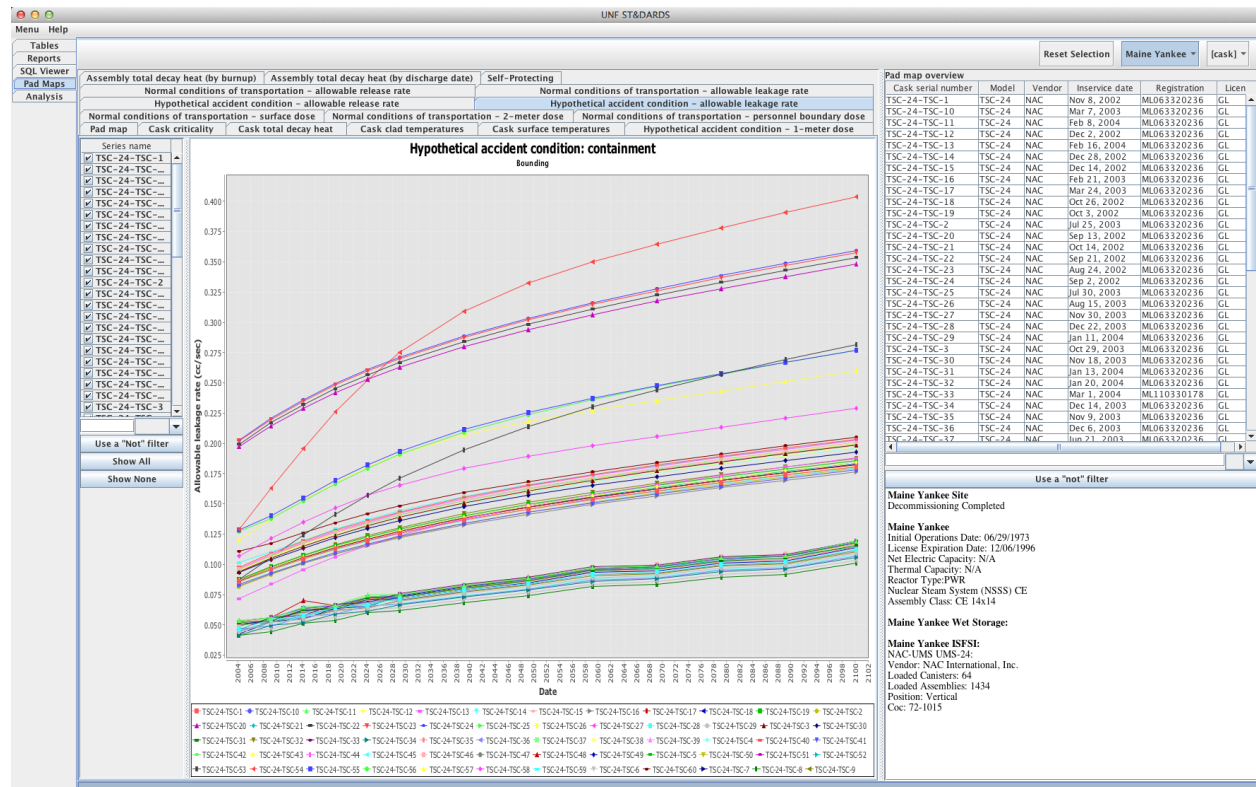


Fig. 3.22: Hypothetical accident condition containment results (allowable leakage rate) for all the canisters at a site in their designated transportation casks.

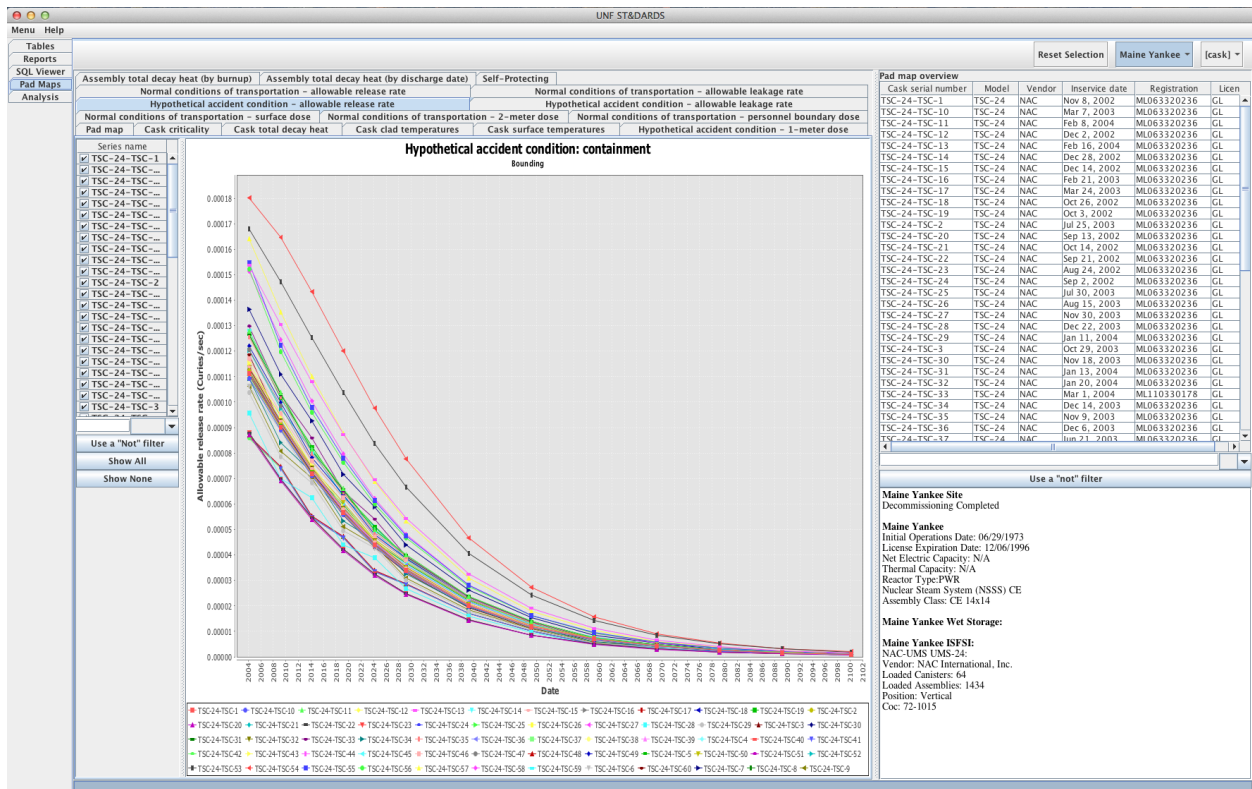


Fig. 3.23: Hypothetical accident condition containment (allowable release rate) results for all the canisters at a site in their designated transportation casks.

Self-Protecting

Site-wise assembly self-protection status can be viewed (as shown in Fig. 3.24) using the Self-Protecting tab. The Self-Protecting tab allows user to change the self-protection threshold (dose above which the assemblies are considered as self-protected) and recalculate the self-protection status. The default threshold is set to 100 Rem. The following figure presents the Maine Yankee self-protecting status.

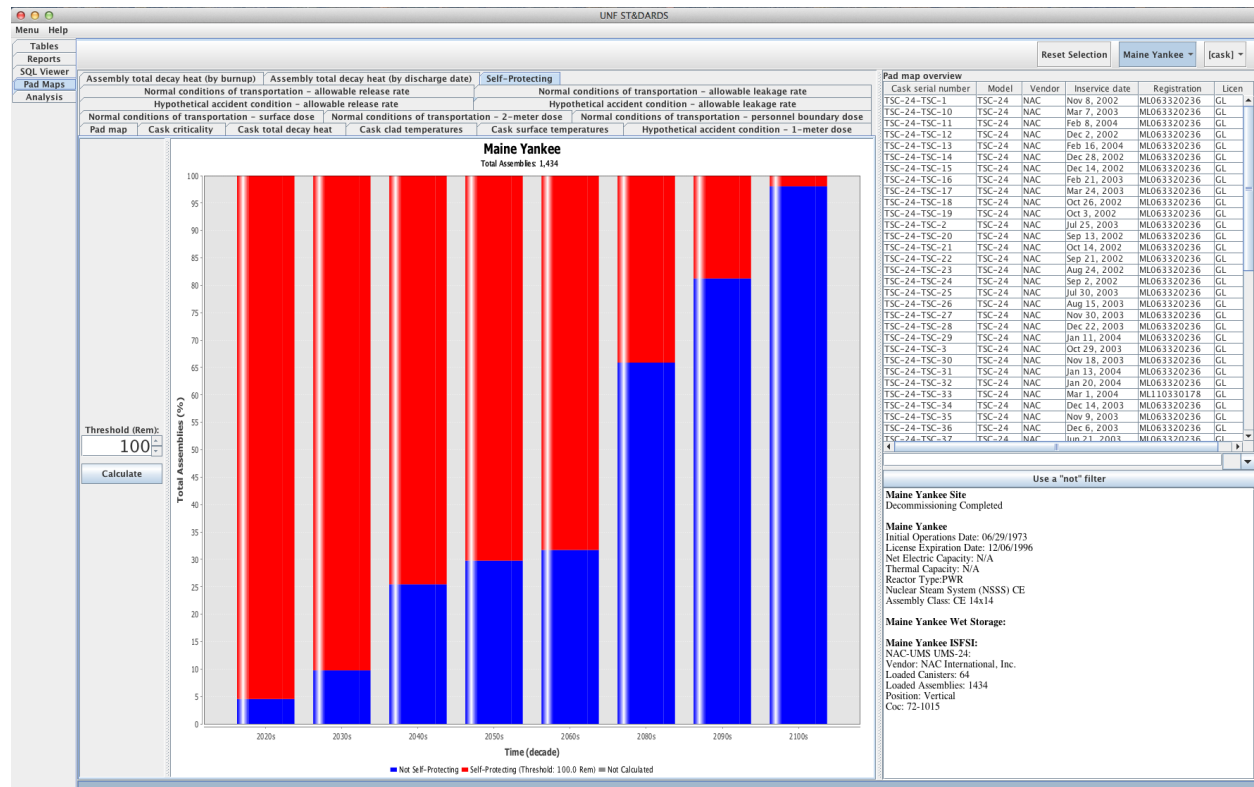


Fig. 3.24: Self-protecting status of discharged assemblies at a selected site.

3.1.3 Analysis Model Geometry Viewer

Criticality, thermal, dose, and containment results for a particular canisters/casks can be found by selecting that canister from the pad maps (cask) breadcrumb in the upper right corner.

Fulcrum Editor

If the latest version of SCALE is available in the local machine, then the criticality and the shielding analysis model for the selected canister/cask can be opened in SCALE Fulcrum editor by clicking “view criticality model” and or “view shielding model” (Fig. 3.25).

The Fulcrum editor can be used to plot the analysis model as shown in Fig. 3.26 (Criticality Model). If the dose map is available in the local machine, then the Fulcrum editor can also be used to superimpose a dose map using the mesh tab on the analysis model, as shown in Fig. 3.27. All the available features in the Fulcrum editor can be found in Fulcrum manual 1.

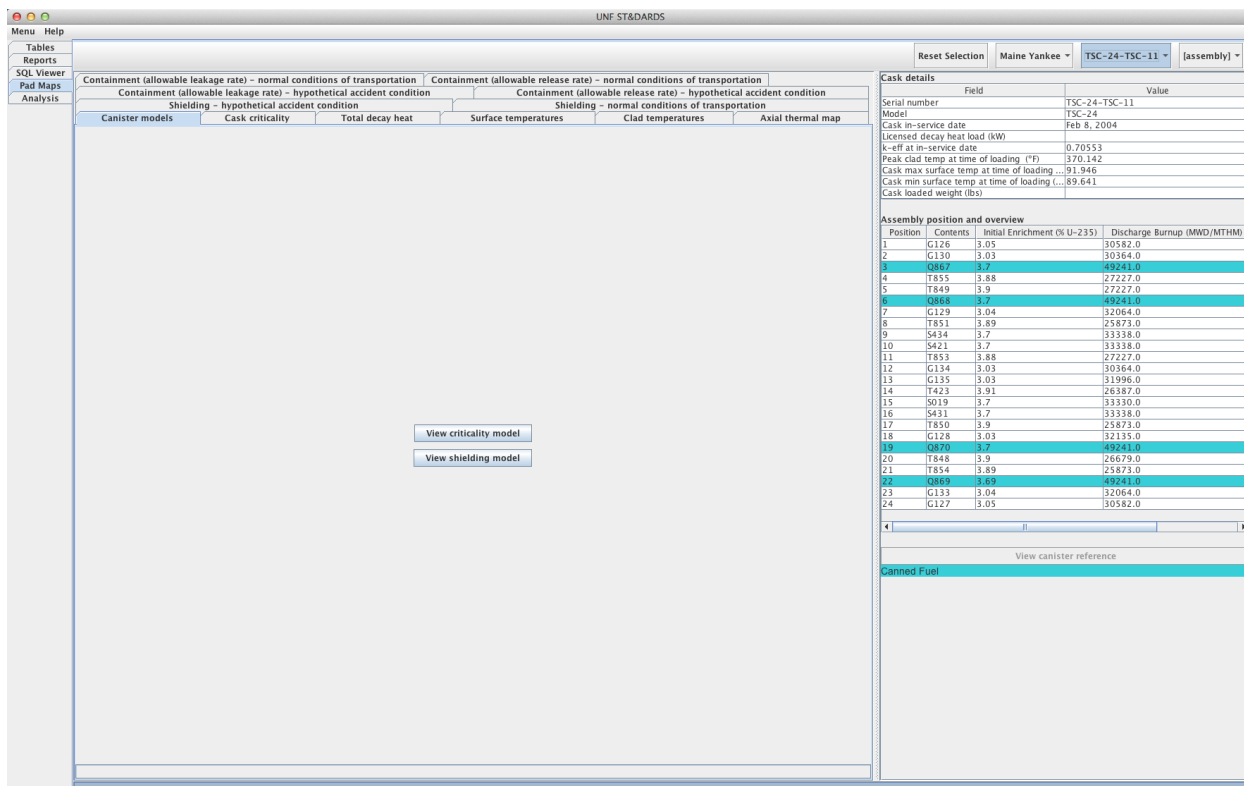


Fig. 3.25: The main display for an individual canister/cask.

The individual cask view (Fig. 3.25) provides results viewing tabs (e.g., cask criticality, surface temperature) similar to those described above for the site view (Fig. 3.9) that provides analysis results of all casks/canisters on the same plot.

Heat Map Viewer

In addition, the individual cask view provides an axial thermal map tab displaying an axial thermal map of the selected cask, as shown in Fig. 3.28. The cask thermal map can be varied axially and as a function of time using the side and bottom scroll bars.

¹ SCALE: A Comprehensive Modeling and Simulation Suite for Nuclear Safety Analysis and Design, ORNL/TM-2005/39, Version 6.1, Oak Ridge National Laboratory, Oak Ridge, Tennessee, June 2011.

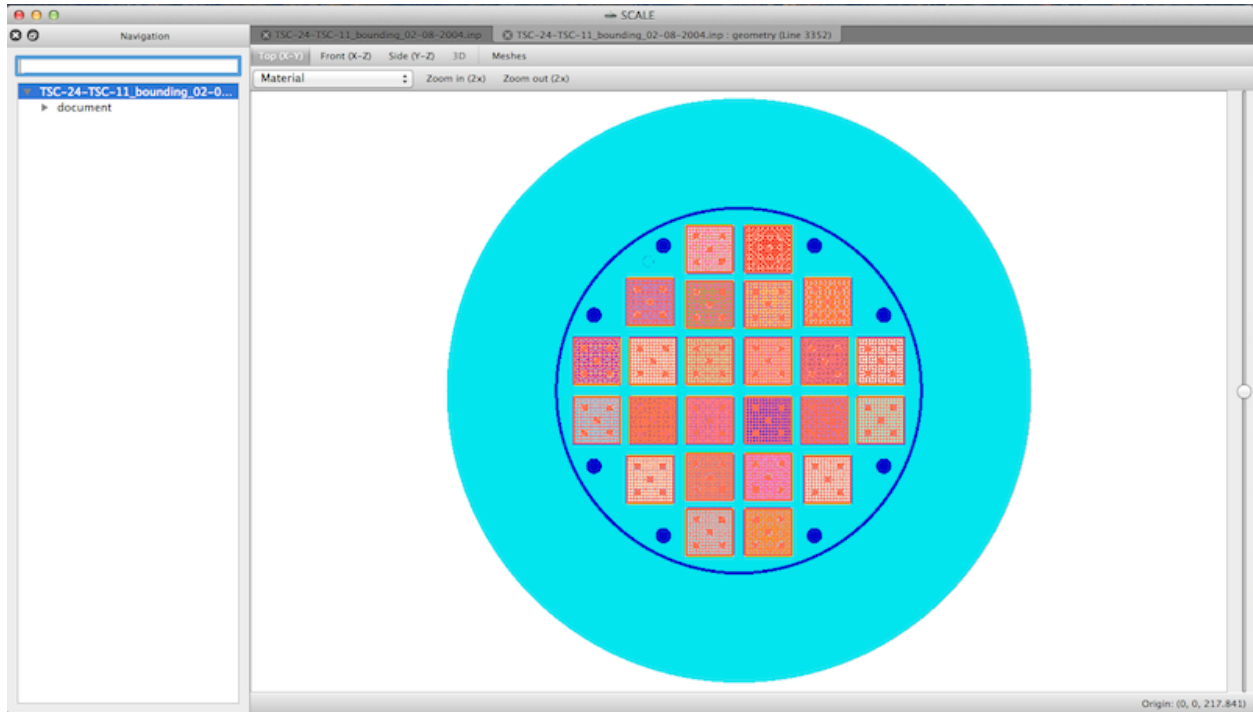


Fig. 3.26: Fulcrum editor displaying a criticality model.

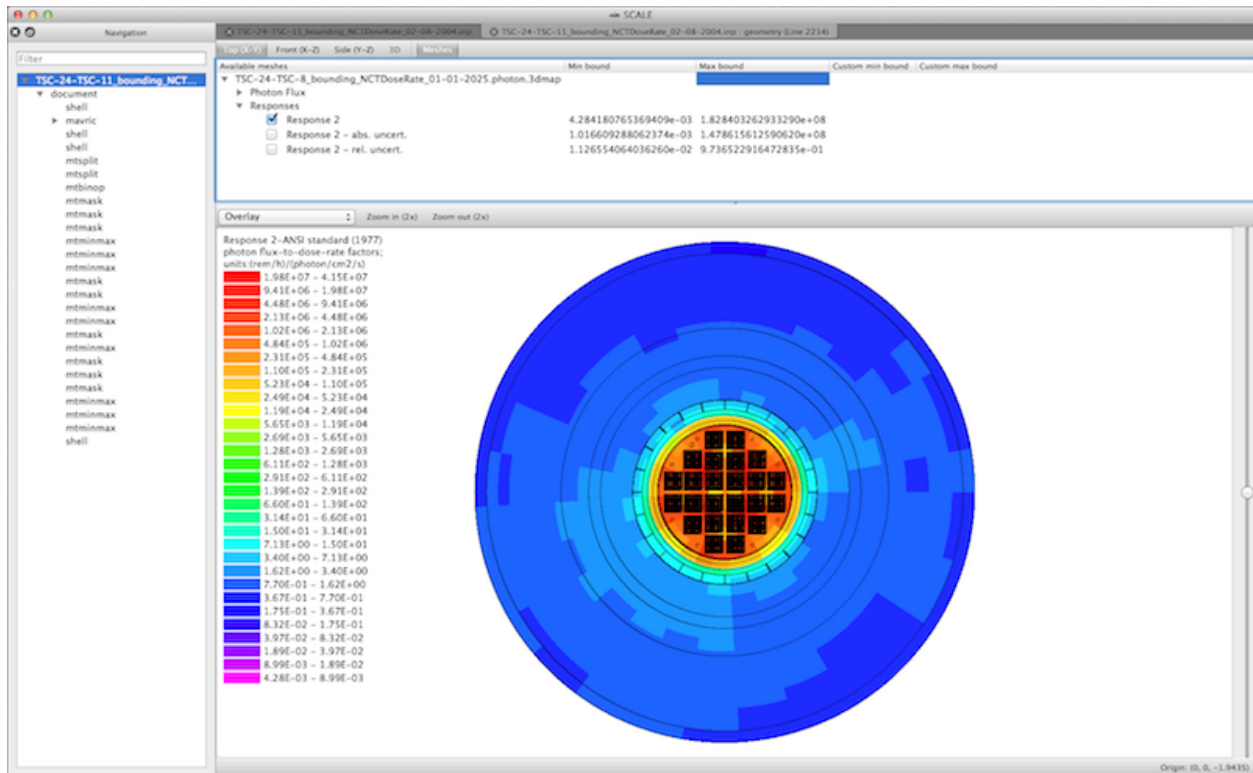


Fig. 3.27: Fulcrum editor displaying a superimposed dose map on a shielding model.

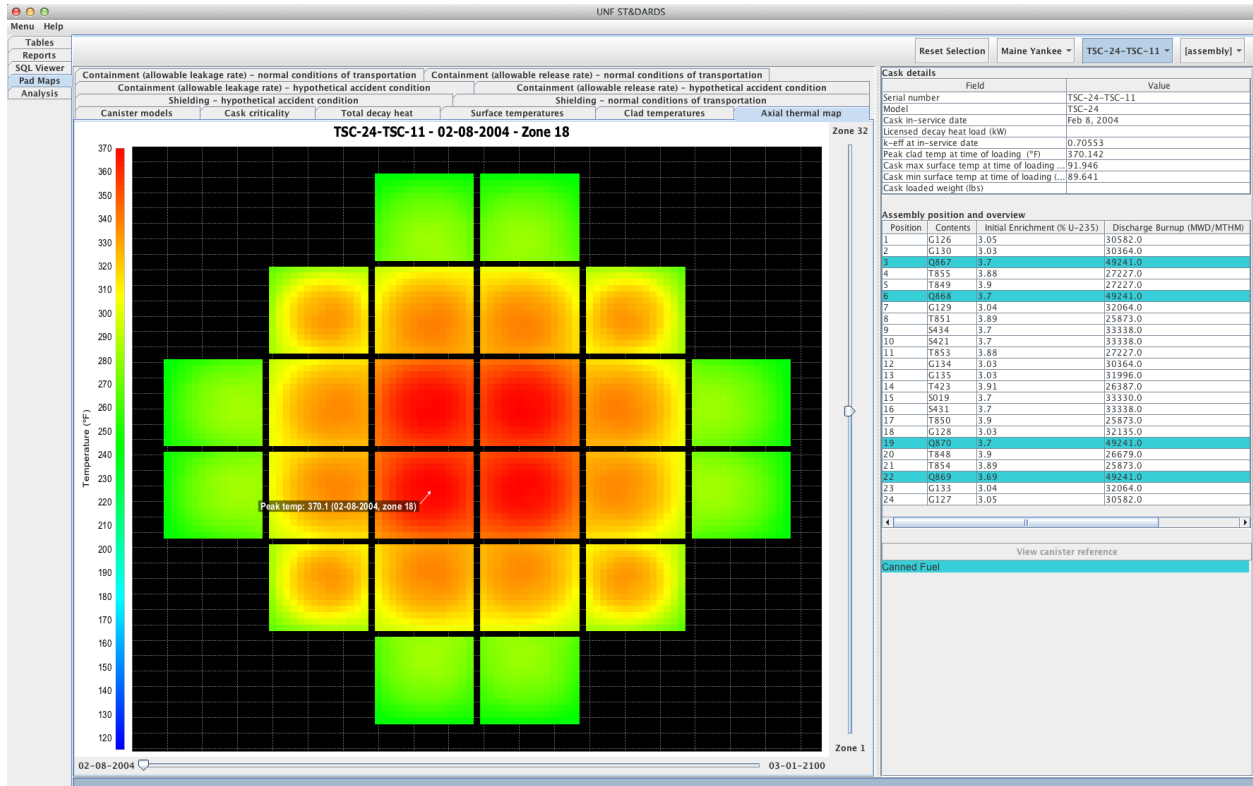


Fig. 3.28: Axial thermal map of a selected cask.

3.1.4 Plot Functionality

- **Zoom-in:** A left-click while dragging down and to the right will create a zoom box; releasing the left-click will zoom in.
- **Zoom-reset:** A left-click while dragging up and to the left will initiate the zoom-reset; releasing the left-click will reset the zoom.
- **Plot Properties:** A right-click will display a plot-context menu listing the following:
 - Properties - displays configurable plot properties (e.g., title, axis labels, text fonts/size, colors):
 - * Title Tab - Allows changing toggle display of title on or off, as well as change of text, font, and color.
 - * Plot Tab - Allows changing the domain and range axis label text, font, color, and tick labels, and the plot's appearance (e.g., outline stroke, outline color, background color, and plot orientation).
 - * Legend Tab - Allows changing the legend's fonts used to display the plot series names.
 - * Other Tab - Allows changing the anti-aliasing rendering used, as well as the

background color.

- Copy - copies the plot image to the system’s clipboard for subsequent pasting into emails or documents.
- Save as ... - Saves the plot as a portable network graphics (PNG) formatted image to the specified location.
- Print... - Allows for printing the plot to a selected printer.
- Export... - Allows for exporting the plot data as a Microsoft Excel (xlsx) formatted file for further data interrogation.

3.1.5 SQL Viewer

The SQL viewer provides direct access to all data stored in the Unified Database. Using SQL select statements, users can precisely define the data they want. The results are displayed in a table from which the user can save the results in a comma-separated value (CSV) file.

As shown in Fig. 3.29, the SQL viewer is divided into left and right panels. The left panel includes a list of table names on top and a tree view of table details at the bottom.

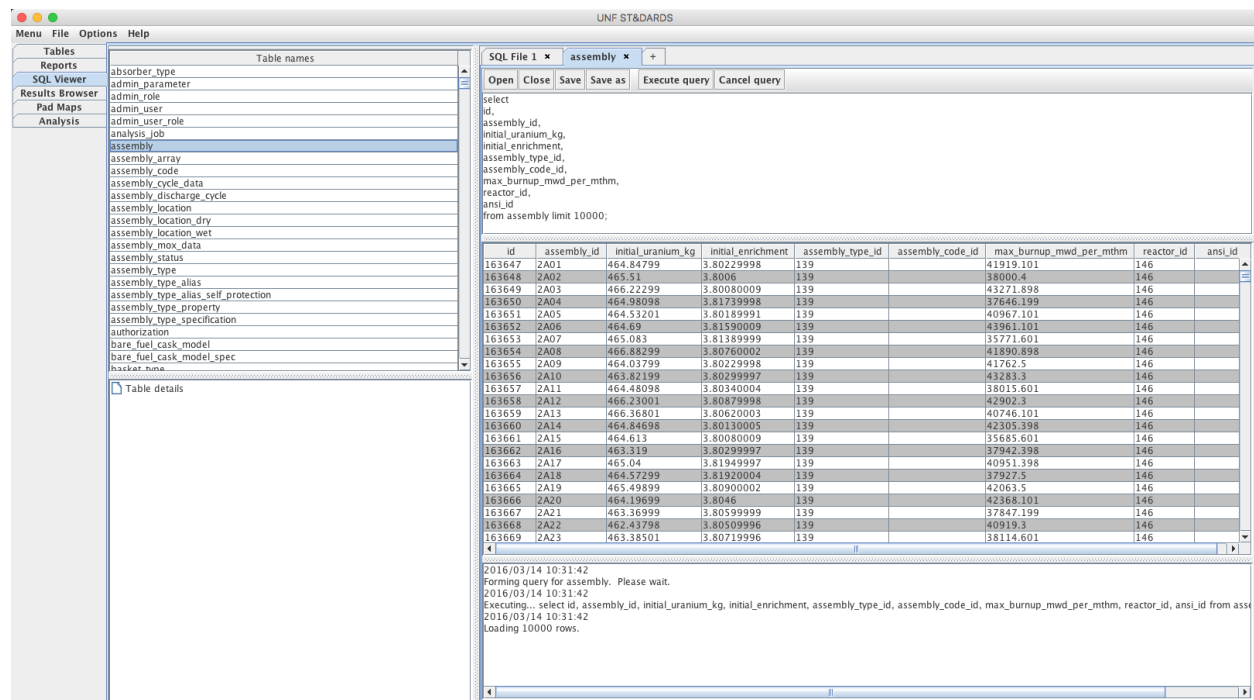


Fig. 3.29: UNF-ST&DARDS SQL viewer.

Table names: In the list of table names, right clicking on a table name opens a pop-up menu with options to “select rows” or “show table description.” Clicking “select rows” will

generate a SQL query in the query editor window and will then execute the query. Clicking “show table description” will add an entry to the table details tree.

Table details: The table details are displayed as an expandable/collapsible tree. The items in the tree describe the table structure in detail:

- A top-level item shows the table’s name. Clicking the icon next to the name (or double clicking the name) will expand the details of the table to reveal a description of the table’s contents and a tree item for each column.
- Each column also expands to offer a description and details about the units, datatype, nullability, and uniqueness. This information can be helpful when attempting to write a custom SQL query.

The right panel includes, from top to bottom, a query editor, results table, and message panel.

- Query editor: The query editor panel allows users to write ad hoc queries for retrieving custom data sets from the database. The query editor’s menu bar offers several actions:
 - New SQL Tab - Creates a blank window for writing a new query.
 - Open - Opens an SQL command from a file on your computer.
 - Close - Closes the query editor tab.
 - Save - Saves changes to the SQL file in the current tab.
 - Save as - Saves changes to the SQL file in the current tab with a new name.
 - Execute query - Sends the SQL command from the current tab to the database and displays the results. If there is more than one command, the selected command will be executed.
 - Cancel query - Attempts to interrupt the currently running query.
- Results table: The query results table displays the results and provides options to “save to CSV,” a common format for tabular data and allows users to view the data in most spreadsheet software, like Excel. When saving, specify the filename extension as “csv,” as in filename.csv.
- Message panel: The message panel displays information from the SQL Viewer as it communicates with the database.

3.1.6 Reports

The Reports tab provides UNF-ST&DARDS automated Characteristics of Potential Repository Waste report generation capability. The automated report generation capability uses data from the Unified Database to create tables, figures, and other form of data representations. [Fig. 3.30](#) presents the Reports tab. The users can select their favorite Latex

engine using the Latex Executable Directory window (A default is provided with UNF-ST&DARDS package). The users can select the location where the report will be saved from the Document Results Directory window (/Users/kb4/UNF_STANDARDS_reports in the figure below). The Generate PDF button is used to generate the report (in tex and pdf formats). The generated report can be found in the location selected in the Document Results directory window.

Note: The Reports tab is work in progress and mainly shows the automated report generation capability.

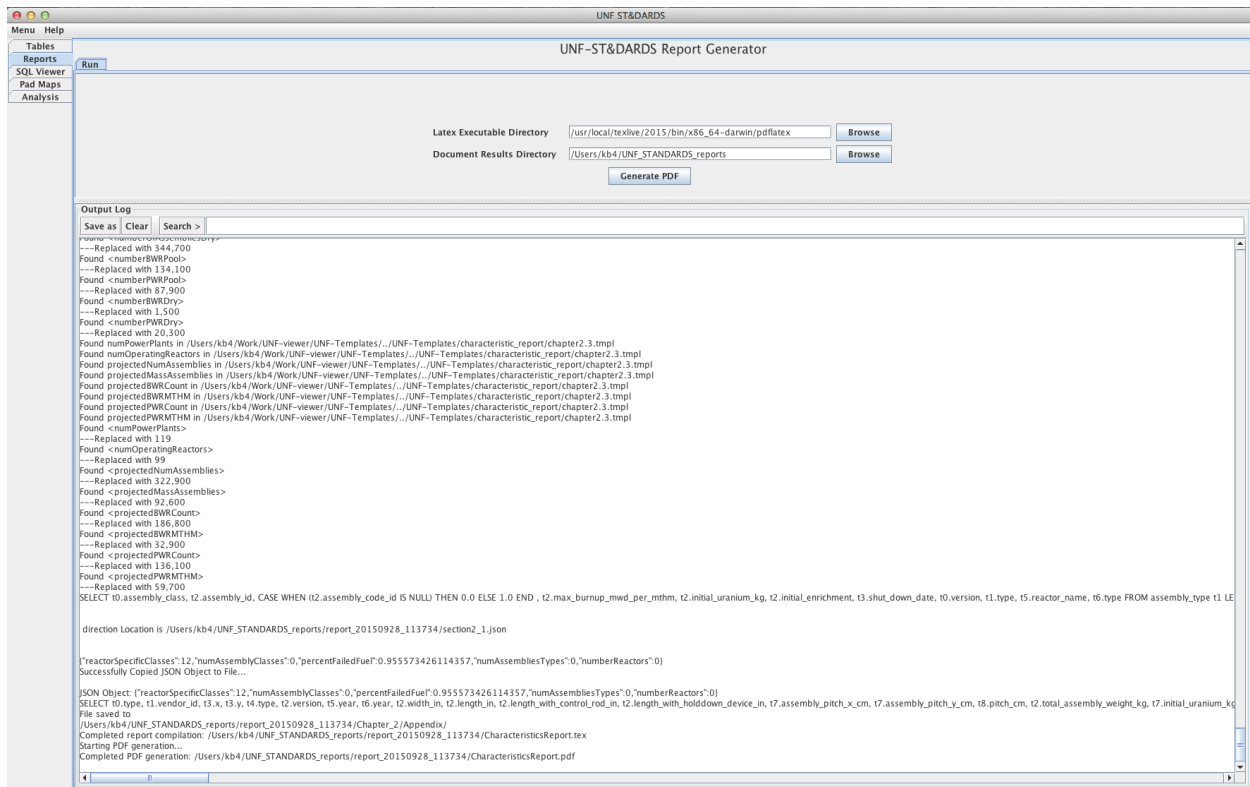


Fig. 3.30: UNF-ST&DARDS report generation tab.

4.1 Development Environment

Note: This section is under development. The purpose of this section is to provide a quick-start-guide to a new code developer.

UNF-ST&DARDS is comprised of 6 projects:

1. The **UNF-Db project** contains the JPA classes to interact with the Unified Database along with some other classes,
2. The **EIA859 project** contains the JPA classes to interact with the MySQL GC859 database (direct SQL dump of the MS Access database) and logic to normalize the data into Java objects,
3. The **TemplateEngine** expands analysis (e.g., depletion, criticality) templates from **UNF-Templates** with JSON files containing data from the database to create input files for SCALE and the other analysis codes,
4. The **UNF-Templates** contains analysis templates developed by various analysts,
5. The **UNF-Db-Model** contains the logic for populating JSON files, running the analysis codes, uploading results to the database, etc.,and
6. The **UNF-Db-Viewer** provides a GUI allowing the user to define analysis jobs, submit them for local execution, review results, and interact with the database.

The code repositories are:

UNF-Db: <<https://fogbugz.ornl.gov/kiln/Code/Used-Nuclear-Fuel-UNF/Group/UNF-Db>>

EIA859: <<https://fogbugz.ornl.gov/kiln/Code/Used-Nuclear-Fuel-UNF/Group/EIA859>>

TemplateEngine: <<https://fogbugz.ornl.gov/kiln/Code/TemplateEngine/Group/TemplateEngine>>

UNF-Templates: <<https://fogbugz.ornl.gov/kiln/Code/Used-Nuclear-Fuel-UNF/Group/UNF-Templates>>

UNF-Db-Model: <<https://fogbugz.ornl.gov/kiln/Code/Used-Nuclear-Fuel-UNF/Group/UNF-Db-Model>>

UNF-Db-Viewer: <<https://fogbugz.ornl.gov/kiln/Code/Used-Nuclear-Fuel-UNF/Group/UNF-Db-Viewer>>

4.1.1 Coding Standards

- Write well-documented, orthogonal code that produces the correct results and enables continuous integration with tests.
- Use clear formatting like the default NetBeans code formatter settings.

4.1.2 Test Infrastructure

Whenever code updates are pushed to the central repo, the continuous testing infrastructure runs the regression tests.

Test executions status can be seen @ **Jenkins:** <<http://ci.ornl.gov:8080/view/UNF/>>

Test results can be retrieved from **CDash:** <<http://ci.ornl.gov/CDash/index.php?project=UNF>>

4.1.3 Setting up UNF-ST&DARDS Development Environment

Tools, libraries, and accounts needed:

1. **Mercurial:** <<https://www.mercurial-scm.org/>>
2. **NetBeans:** <<https://netbeans.org/>>
3. **MySQL WorkBench:** <<https://www.mysql.com/products/workbench/>>
4. **MySQL Server:** (Optional) If you want to update,insert,delete data during development: <<https://dev.mysql.com/downloads/mysql/>>
5. Linux account on hulk cluster: email Ken Barker <barkerkr@ornl.gov>, cc Kaushik Banerjee <banerjee@ornl.gov> and John Scaglione <scaglionejm@ornl.gov>
6. FogBugz account on fogbugz.ornl.gov: email Tony Walsh <walshtd@ornl.gov>
7. CDash account on ci.ornl.gov: email Tony Walsh <walshtd@ornl.gov>
8. MySQL account(s) on hulk: Optional: email Paul Miller <millerpp@ornl.gov>
9. MacTex: (Optional) If you are working on Latex reporting features, for Mac: <<https://tug.org/mactex/>>

10. MikTeX: (Optional) If you are working on Latex reporting features, for Windows: <<http://miktex.org/>>

Make a parent directory for the projects in your home directory like `C:/Users/<user_id>/NetBeans_Projects`. In subsequent steps we will refer to this directory as the `<projects_directory>`.

Navigate into the `<projects_directory>`.

Clone the code from each of the repositories under `<projects_directory>` using the “**hg clone**” command.

Clone the Unified Database to a local MySQL Server is optional as mentioned above. The instructions are available at <<https://fogbugz.ornl.gov/default.asp?W97>>. Do not attempt to use MySQL Workbench to export the dump of the entire Unified Database from hulk:3306; you only need 1 GB ...not the full 30GB database. Additionally, it may be necessary to clone the EIA859 MySQL database depending on the work you are doing. In this case you can use the MySQL Workbench to export the dump. Then restore EIA859 database as detailed in the scripts from <<https://fogbugz.ornl.gov/default.asp?W97>> with modifications to use the `dump_EIA859.sql` as the source.

Step 1: Create the projects:

For each of the projects (**project setup order: UNF-Db, TemplateEngine, EIA859, UNF-Db-Model, UNF-Db-Viewer**):

In NetBeans, *left-click* the “File” menu. Select “New Project...”

In the dialog, *left-click* Java Project with Existing Sources, then click the “Next” button.

In the “New Java Project With Existing Sources” dialog...

Project Name should be provided according to the repository name, i.e. “UNF-Db”.

Project Folder should be `<projects_directory>/<project_name>/projects/<user_id>`.

Click the “Next” button.

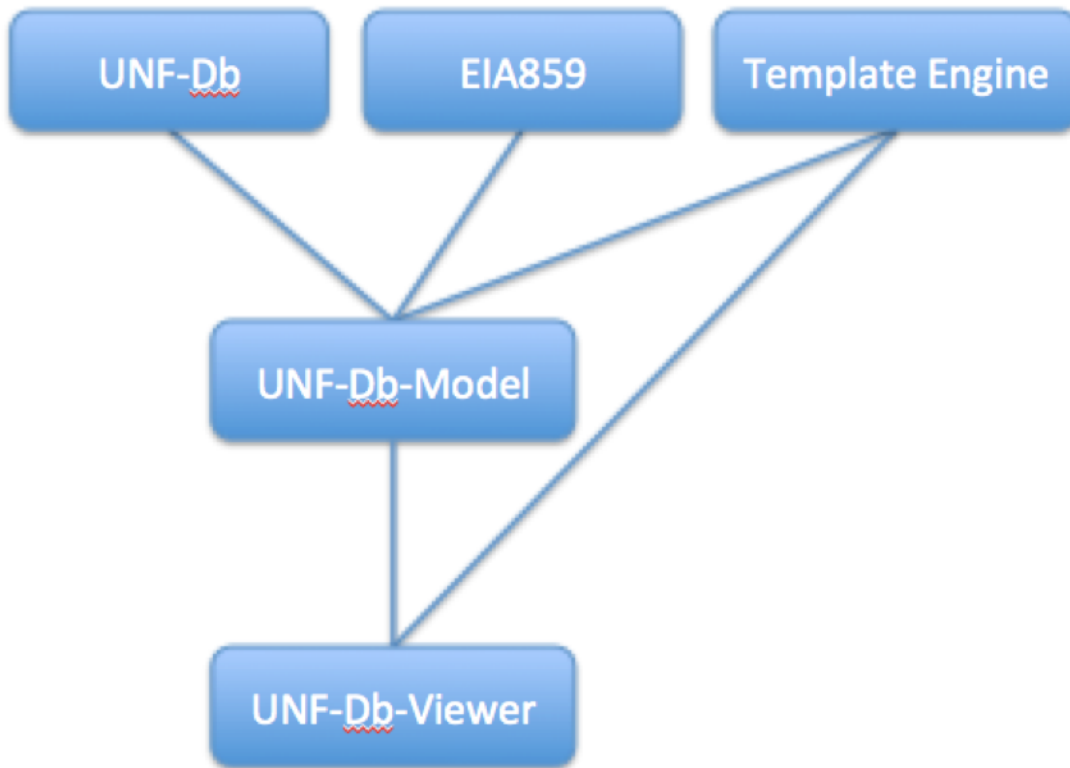
Next to the “Source Package Folders:” box, click the “Add Folder...” button.

In the “Browse Source Packages Folder” dialog, navigate to `<projects_directory>/<project_name>/src` and click the “Open” button.

In the “New Java Project With Existing Sources” dialog, click the “Finish” button.

Step 2: Link the projects and add the libraries:

Project dependency diagram:



For each of the projects (**project order: UNF-Db, TemplateEngine, EIA859, UNF-Db-Model, UNF-Db-Viewer**):

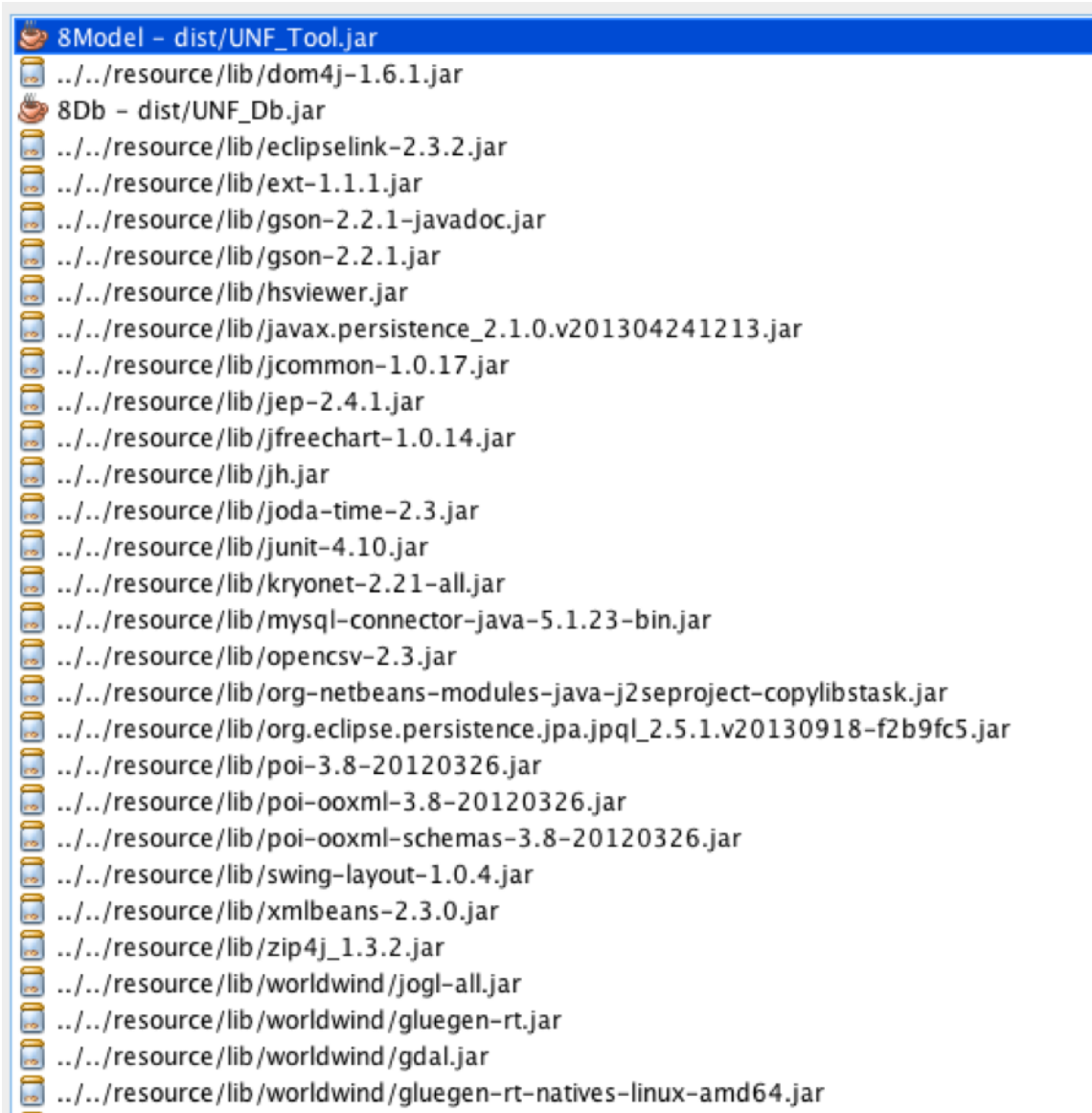
On the projects navigation panel on the left (NetBeans), *right-click* the project's "Source Packages" folder and select "Properties"

In the "Project Properties - <project_name>" dialog, select "Libraries" from the "Categories:" panel on the left.

Click the "Add Project..." button for each project listed earlier in the above diagram.

Click the "Add JAR/Folder" button and navigate to <projects_directory>/<project_name>/resource/lib and select all and click the "Choose" button.

The pictures below show any necessary ordering for the projects' libraries. Make sure that the order for your libraries match.



Step 3: Configure the resource file

UNF-ST&DARDS reads configuration from the file `.unf.rc` located in the user's home directory.

4.2 TemplateEngine

The TemplateEngine is a data-driven attribute substitution program. Data are stored in a hierarchical, lightweight format called [JavaScript Object Notation \(JSON\)](#). The TemplateEngine is written in the Java™ programming language, which means it is cross-platform and should run out of the box. Any question/comment/issue regarding TemplateEngine should be sent to [<UNFHelp@ornl.gov>](mailto:UNFHelp@ornl.gov) for further assistance. All of the examples pro-

vided in this appendix can be found and reproduced from the files located in the TemplateEngine's example directory. A MAC computer is used for the examples in this appendix.

4.2.1 TemplateEngine Requirements

The following system configurations are required for running TemplateEngine:

- Java 1.7 Runtime Environment (JRE 1.7) or newer (<http://www.oracle.com/java>),
- 1 GB RAM or greater, and
- Mercurial 2.5 or newer (<http://mercurial.selenic.com>).

The TemplateEngine is freely available within the Oak Ridge National Laboratory network. Using Mercurial Distributed Version Control System, the TemplateEngine repository can be easily cloned in a local machine as shown in Fig. 4.1. The Mercurial (hg) clone of the repository will download all files required by the TemplateEngine into the TemplateEngine directory. The distribution directory (TemplateEngine/dist) contains the executable java jar file that can now be invoked to expand templates, as shown in Fig. 4.2.

```
$hg clone https://fogbugz.ornl.gov/kiln/Code/TemplateEngine/Group/TemplateEngine
TemplateEngine
requesting all changes
adding changesets
adding manifests
adding file changes
added 47 changesets with 324 changes to 193 files
updating to branch default
142 files updated, 0 files merged, 0 files removed, 0 files unresolved
$
```

Fig. 4.1: Mercurial clone example.

```
$ls TemplateEngine/dist/
README.TXT      TemplateEngine.jar  lib
$java -jar TemplateEngine/dist/TemplateEngine.jar
***Usage:
  java -jar TemplateEngine.jar /path/to/template [/path/to/json/variables/file]? <-o
outputfile>
$
```

Fig. 4.2: Verification of TemplateEngine execution.

4.2.2 JSON Parameter Set

The TemplateEngine is driven by parameters which are defined in the JSON format. JSON can describe keyed-values, objects, arrays, and compositions of keyed-values, objects and

arrays. The JSON standard is defined at <<http://www.json.org>>. A parameter in its simplest form is a key-value pair delimited by a colon.

key : *value*

The key must be quoted, and is the name of the parameter. The value must be quoted if it is a string value. An object is delimited by left and right curly braces – {}; arrays are delimited by left and right square brackets – [], and commas delimit object and array members.

"key1" : *value1*, "key2" : *value2*, ...
[*value1*, *value2*, ...]

Here, the object contains unique keys, "key1" and "key2" ... where each key's value could be a primitive value (number, string, etc.) or a nested object or array. Fig. 4.3 depicts an example JSON describing a donut shop's donut options. The document object contains four components, two keyed-values—*type* and *name*—and two arrays—*batter* and *topping*. Batter contains 4 objects, and topping contains 7 objects. Each object in *batter* and *topping* contains two keyed-values, *id* and *type*. This example illustrates the use of keyed-values ("type": "donut", etc.), and arrays ("type", "batter").

```
{
  "type": "donut",
  "name": "Cake",
  "batter":
  [
    { "id": 1001, "type": "Regular" },
    { "id": 1002, "type": "Chocolate" },
    { "id": 1003, "type": "Blueberry" },
    { "id": 1004, "type": "Devil's Food" }
  ],
  "topping":
  [
    { "id": 5001, "type": "None" },
    { "id": 5002, "type": "Glazed" },
    { "id": 5005, "type": "Sugar" },
    { "id": 5007, "type": "Powdered Sugar" },
    { "id": 5006, "type": "Chocolate with Sprinkles" },
    { "id": 5003, "type": "Chocolate" },
    { "id": 5004, "type": "Maple" }
  ]
}
```

Fig. 4.3: Example donut shop JSON.

4.2.3 TemplateEngine Features

The TemplateEngine feature set is intended to facilitate template expansion involving formatted attribute substitution, expression evaluations, and subtemplate import. The templates can be described with the following components:

- static text,
- attributes,
- indirect attributes,
- scalar expression evaluation,
- iterative expression evaluation,
- subtemplate import,
- conditional action,
- iterative subtemplate import, and
- parameterized subtemplate import.

Template components are described below.

Static text

Static text is the most common template construct. The TemplateEngine conducts a pass-through of the text with no interpretations or evaluations. [Fig. 4.4](#) presents an example of static text in a template.

```
$cat examples/static.tpl
The brown fox jumped over the lazy dog
$java -jar dist/TemplateEngine.jar examples/static.tpl > static.txt
$cat static.txt
The brown fox jumped over the lazy dog
$
```

Fig. 4.4: Static text template.

Attributes

Attributes allow for static text to be parameterized to present alternatives. [Fig. 4.5](#) depicts a template containing two attributes, `jumper`, and `what`, which allows the attribute values to change, producing different results without modifying the template.

Indirect attributes

Indirect attributes allow attributes to be parameterized, further increasing the flexibility of the template. Arbitrary indirection is allowed. [Fig. 4.6](#) depicts a template in which the `jumper` attribute ([Fig. 4.5](#)) has been parameterized to incorporate the `what` attribute, allowing for an indirect attribute lookup based on the value of the `what` attribute. With this indirect attribute, the change of the `what` attribute also changes the `jumper[<what>]` attribute.

[Fig. 4.7](#) depicts how changing the value of `what` from `lazy yellow dog` to `lazy fat cat` produces a context change in the result. While the example is simple, the indirect attribute feature is another means of enhancing templates and increasing their flexibility. [Fig. 4.7](#)

```

$cat examples/attribute.tpl
The <jumper> jumped over the <what>
$cat examples/attribute.json
{
  "jumper" : "fluffy brown fox"
  ,"what" : "lazy yellow dog"
}
$java -jar dist/TemplateEngine.jar examples/attribute.tpl
examples/attribute.json > attribute.txt
Found jumper in examples/attribute.tpl
Found what in examples/attribute.tpl
Found <jumper>
---Replaced with fluffy brown fox
Found <what>
---Replaced with lazy yellow dog
$cat attribute.txt
The fluffy brown fox jumped over the lazy yellow dog
$

```

Fig. 4.5: Template with attributes.

```

$cat examples/indirect_attributes.tpl
The <jumper[<what>]> jumped over the <what>
$cat examples/indirect_attributes.json
{
  "jumper[lazy yellow dog]" : "fluffy brown fox"
  ,"jumper[lazy fat cat]" : "cute house mouse"
  ,"what" : "lazy yellow dog"
}
$java -jar dist/TemplateEngine.jar examples/indirect_attributes.tpl
examples/indirect_attributes.json > indirect_attributes.txt
Found what in examples/indirect_attributes.tpl
Found what in examples/indirect_attributes.tpl
Found <what>
---Replaced with lazy yellow dog
Found <what>
---Replaced with lazy yellow dog
Found <jumper[lazy yellow dog]>
---Replaced with fluffy brown fox
$cat indirect_attributes.txt
The fluffy brown fox jumped over the lazy yellow dog

```

Fig. 4.6: Indirect attribute example 1.

depicts reuse of the `examples/indirect_attributes.tpl` template with a change in the value of *what*, which changes the result of the template to involve a mouse and a cat instead of a fox and a dog.

```
$cat examples/indirect_attributes1.json
{
  "jumper[lazy yellow dog]" : "fluffy brown fox"
  ,"jumper[lazy fat cat]" : "cute house mouse"
  ,"what" : "lazy fat cat"
}
$java -jar dist/TemplateEngine.jar examples/indirect_attributes.tpl
examples/indirect_attributes1.json > indirect_attributes1.txt
Found what in examples/indirect_attributes.tpl
Found what in examples/indirect_attributes.tpl
Found <what>
---Replaced with lazy fat cat
Found <what>
---Replaced with lazy fat cat
Found <jumper[lazy fat cat]>
---Replaced with cute house mouse
$cat indirect_attributes1.txt
The cute house mouse jumped over the lazy fat cat
$
```

Fig. 4.7: Indirect attribute example 2.

Templates may require expressions to be evaluated. The TemplateEngine supports a full suite of mathematical operators and functions. Fig. 4.8 depicts scalar expression evaluations. A scalar expression is formatted as follows:

$$\#eval\,fmt = \,format(expression)as\,new_attribute\#$$

Where the `#eval` indicates the start of an expression; `fmt=format` is an optional format specification for the result; the parentheses delimit the expression; the expression is any operation, including a numeric or string-based result; `as new_attribute` is an optional component that will indicate the result of the expression to be saved for later use by the name `new_attribute`; and the final `#` closes the expression. On occasion it may be desired for the result of the expression evaluation to remain silent. This can be accomplished with the silent scalar expression using the following format:

$$\#\backslash eval\,fmt = \,format(expression)as\,new_attribute\#$$

Fig. 4.8 depicts a simple conversion of an attribute in *inches*, `howfar_inches`, to another attribute in *feet*, `howfar_feet`. The first expression evaluated uses the `as new_attribute` feature to store the result of the expression into a re-usable attribute, `howfar_feet`, which is used in the second sentence of the template.

Iterative expression evaluation

The ability to evaluate an expression iteratively as a function of some attribute in the system can be necessary to properly define a template result. The format for an iterative

```

$cat examples/scalar_expr.tpl
The <jumper> jumped #eval (howfar_inches/12) as howfar_feet# feet over the <what>.
<howfar_feet> feet!
$cat examples/scalar_expr.json
{
  "jumper" : "fluffy brown fox"
  , "what" : "lazy yellow dog"
  , "howfar_inches" : 78
}
$java -jar dist/TemplateEngine.jar examples/scalar_expr.tpl examples/scalar_expr.json >
scalar_expr.txt
Found jumper in examples/scalar_expr.tpl
Found what in examples/scalar_expr.tpl
Found howfar_feet in examples/scalar_expr.tpl
Storing evaluated expression howfar_feet as 6.5
Found #eval (howfar_inches/12) as howfar_feet#, replacing with 6.5
Found <jumper>
---Replaced with fluffy brown fox
Found <what>
---Replaced with lazy yellow dog
Found <howfar_feet>
---Replaced with 6.5
$cat scalar_expr.txt
The fluffy brown fox jumped 6.5 feet over the lazy yellow dog. 6.5 feet!
$

```

Fig. 4.8: Scalar expression example.

expression is as follows:

$$\#funcfmt = formatsep = separator(range)expression\#$$

where *#func* indicates the start of an iterative expression; *fmt=format* is an optional format specification for the results; *sep=separator* is an optional result separator defaulted to a single whitespace; parentheses delimit the *range*; and *expression* is the function to evaluate per iteration. The *range* is of the format:

$$attribute = start, end$$

where *attribute* is a new or existing attribute in the template parameters which is iterated from *start* through *end*; and *start* and *end* are numeric values. *Range* can be repeated, whitespace separated, to produce embedded loops.

Fig. 4.9 depicts the squaring of the *i* attribute, as *i* is iterated from 1 to 10 while using the default separator and the comma separator.

Subtemplate import

When dealing with large templates, it is best to split a single file into multiple subtemplates. The TemplateEngine provides a subtemplate import feature that allows import of a file using relative or absolute paths. The format for subtemplate import is as follows:

$$\#import filepath$$

```

$cat examples/iterative.tpl
#func (i=1,10) i*i#
#func sep="," (i=1,10) i*i#
$java -jar dist/TemplateEngine.jar examples/iterative.tpl > iterative.txt
Found #func (i=1,10) i*i#, replacing with 1.0 4.0 9.0 16.0 25.0 36.0 49.0 64.0 81.0 100.0
Found #func sep="," (i=1,10) i*i#, replacing with
1.0,4.0,9.0,16.0,25.0,36.0,49.0,64.0,81.0,100.0
$cat iterative.txt
1.0 4.0 9.0 16.0 25.0 36.0 49.0 64.0 81.0 100.0
1.0,4.0,9.0,16.0,25.0,36.0,49.0,64.0,81.0,100.0
$

```

Fig. 4.9: Iterative expression example.

Where the *#import* indicates the start of a subtemplate import and *filepath* is an absolute or relative file path. *#import* must start in the first column. Fig. 4.10 depicts the subtemplate import of the *static.tpl* file. Notice that in the activity message indicating Importing *static.tpl* using 'null,' 'null' is an empty parameter set. This will be discussed in parameterized subtemplate import section below. By default, an import of a subtemplate provides the subtemplate access to all attributes in the parent template's attribute set.

```

$cat examples/subtemplate_import.tpl
#import static.tpl
$java -jar dist/TemplateEngine.jar examples/subtemplate_import.tpl >
subtemplate_import.txt
Importing static.tpl using 'null'
$cat subtemplate_import.txt
The brown fox jumped over the lazy dog
$

```

Fig. 4.10: Subtemplate import example.

Some templates may require that subsections be expanded only when a certain condition applies. The TemplateEngine provides conditional actions to facilitate these types of template expansions. The condition action format is as follows:

```

#ifdefcondition
action
#endif

```

where the *#ifdef* or *#ifndef* indicate the start of a conditional action; *condition* is either an attribute name or value, or scalar expression that evaluations to 1 (true), 0 (false), or an attribute name; the *action* is any TemplateEngine feature described in this document; and *#endif* terminates the conditional action. Both *#ifdef* and *#endif* must occur in the first column.

Fig. 4.11 depicts two invocations of the TemplateEngine, exercising two paths through the *examples/conditional_action.tpl* file. The first invocation determines that the attribute *jumper* exists and then imports the *examples/attribute.tpl* subtemplate. The second invocation has no JSON parameters specified, so the attribute *jumper* does not exist, and the static text "Nothing jumping today" is printed.


```
$cat examples/conditional_action.tpl
#ifdef jumper
#import attribute.tpl
#endif
#ifndef jumper
Nothing jumping today.
#endif
$cat examples/attribute.json
{
  "jumper" : "fluffy brown fox"
  , "what" : "lazy yellow dog"
}
$java -jar dist/TemplateEngine.jar examples/conditional_action.tpl
examples/attribute.json > condition_action_true.txt
Importing attribute.tpl using 'null'
Found jumper in examples/attribute.tpl
Found what in examples/attribute.tpl
Found <jumper>
---Replaced with fluffy brown fox
Found <what>
---Replaced with lazy yellow dog
$cat condition_action_true.txt
The fluffy brown fox jumped over the lazy yellow dog

$java -jar dist/TemplateEngine.jar examples/conditional_action.tpl >
condition_action_false.txt
$cat condition_action_false.txt
Nothing jumping today.
$
```

Fig. 4.11: Conditional action example.

Iterative subtemplate import

The ability to import a subtemplate is powerful, but often a subtemplate may need to be imported repeatedly as a function of some attribute. The iterative subtemplate import is one means of repeating an import over a range. The iterative subtemplate import format is as follows:

```
#repeat filepathusingrange
```

where *#repeat* indicates the start of an iterative subtemplate import; the *filepath* indicates the absolute or relative path to the template to import; *using* delimits the filepath from the range; and *range* indicates the repeats to conduct. The *range* is of the format:

```
attribute = start, end
```

where *attribute* is a new or existing attribute in the template parameters, which is iterated from *start* through *end*; and *start* and *end* are numeric values. *Range* can be repeated, whitespace separated, to produce embedded loops.

Fig. 4.12 depicts the iterative import of the *examples/scalar_expr_fmt.tmpl*, which requires three attributes: *jumper*, *howfar_inches*, and *what*. The JSON parameter file, *examples/attribute.json*, provides attributes *jumper* and *what*, and the attribute *howfar_inches* is defined in the range statement of the iterative import where it is iterated from 73 through 75. While the example is simple, the iterative import feature provides a mechanism for generating identifiers, sampling mathematical expressions, and more.

```
$cat examples/iterative_import.tmpl
#repeat examples/scalar_expr_fmt.tmpl using howfar_inches=73,75
$cat examples/scalar_expr_fmt.tmpl
The <jumper> jumped #eval fmt=%.2f (howfar_inches/12) as howfar_feet# feet over the
<what>. <howfar_feet:fmt=%.2f> feet!
$java -jar dist/TemplateEngine.jar examples/iterative_import.tmpl examples/attribute.json
> iterative_import.txt
Repeating examples/scalar_expr_fmt.tmpl using 'howfar_inches=73,75'
<ACTIVITY OMMITTED FOR BREVITY>...
$cat iterative_import.txt
The fluffy brown fox jumped 6.08 feet over the lazy yellow dog. 6.08 feet!
The fluffy brown fox jumped 6.17 feet over the lazy yellow dog. 6.17 feet!
The fluffy brown fox jumped 6.25 feet over the lazy yellow dog. 6.25 feet!
$
```

Fig. 4.12: Iterative import example.

Parameterized subtemplate import

The TemplateEngine works from a data model consisting of a JSON parameter set. JSON describes hierarchical data by using objects and arrays. These hierarchies can be used by corresponding template hierarchies to scope attributes to subtemplates. The parameterized subtemplate import has the following format:

```
#import filepathusingparameter
```

where *#import* indicates a subtemplate import; *filepath* indicates the absolute or relative path to the template to import; *using* delimits the *filepath* from the *parameter*; and *parameter* is an attribute expression involving a JSON object or array. When the attribute is an object, the members of that object become available to the subtemplate and are accessible by name. When the attribute is an array, an implicit iterative import is conducted for each member of the array.

Fig. 4.13 depicts a parameterized subtemplate import, which implicitly iterates over both *batter* and *topping* arrays. Each member object of *batter* and *topping* is exposed to the subtemplate *example/parameterized_options.tmpl* where the attributes *id* and *type* are formatted as a bulleted option. Because the TemplateEngine's numerical storage is in floating point values, formatting can be used to remove the decimal portion of the number, creating an integer. This is done by using the format statement *fmt=%0f*, which states to use 0 decimal places for the floating value.

4.2.4 Attribute and Expression Format

For both attribute and expression evaluation, an optional format specification can be provided as discussed above. Formatting can be used for numeric or string attributes or for expression results. Format is specified in the following format:

$$fmt = \%T.DF$$

where *fmt* indicates a format is to follow; the percent symbol (%) is used to indicate the beginning of the format; *T* is the optional total space; the decimal (.) and *D* are the optional decimal space; and *F* indicates the format type of attribute being formatted.

The total space specification (T) can indicate the alignment. Default alignment is right. A negative sign (-), provides left alignment. A preceding zero (0) indicates that the preceding empty space is padded with zeros. A preceding plus sign (+) indicates to always show the number's sign even when positive.

The decimal space specification (D) can indicate the number of decimal points for numbers or the number of decimal points used as a harsh limiter for strings (can truncate strings). In Fig. 4.14, the first number formatted illustrates decimal truncation, creating an integer-formatted result. The second number formatted illustrates no truncation. The third formatted number indicates six total spaces with two decimal places. The fourth number formatted illustrates padding with zeros. The fifth number formatted illustrates left alignment.

4.2.5 Functions

The scalar and iterative expressions have access to the functions presented in the following tables.

```

$cat examples/parameterized.tpl
The available selection of <name> <type> is the following:
#eval fmt=%.0f (length(batter)) as batter_count# batters with #eval fmt=%.0f
(length(topping)) as topping_count# toppings.
Toppings (<topping_count>):
#import parameterized_options.tpl using <topping>
#func sep="-" (i=1,32)"-"#
Batters (<batter_count>):
#import parameterized_options.tpl using <batter>
$cat examples/parameterized_options.tpl
* <type> (<id:fmt=%.0f>)
$cat examples/donut_shop.json
{
  "type": "donut",
  "name": "Cake",
  "batter":
  [
    { "id": 1001, "type": "Regular" },
    { "id": 1002, "type": "Chocolate" },
    { "id": 1003, "type": "Blueberry" },
    { "id": 1004, "type": "Devil's Food" }
  ],
  "topping":
  [
    { "id": 5001, "type": "None" },
    { "id": 5002, "type": "Glazed" },
    { "id": 5005, "type": "Sugar" },
    { "id": 5007, "type": "Powdered Sugar" },
    { "id": 5006, "type": "Chocolate with Sprinkles" },
    { "id": 5003, "type": "Chocolate" },
    { "id": 5004, "type": "Maple" }
  ]
}
$java -jar dist/TemplateEngine.jar examples/parameterized.tpl examples/donut_shop.json >
parameterized.txt
<ACTIVITY OMITTED FOR BREVITY>...
$cat parameterized.txt
The available selection of Cake donut is the following:
4 batters with 7 toppings.
Toppings (7):
* None (5001)
* Glazed (5002)
* Sugar (5005)
* Powdered Sugar (5007)
* Chocolate with Sprinkles (5006)
* Chocolate (5003)
* Maple (5004)
-----
Batters (4):
* Regular (1001)
* Chocolate (1002)
* Blueberry (1003)
* Devil's Food (1004)

$

```

Fig. 4.13: Parameterized subtemplate import example.

```

$cat examples/formatting.tpl
#\eval (pi) as fl#\eval ("string") as str#
|<fl:fmt=%.0f>| |<fl>| |<fl:fmt=%6.2f>| |<fl:fmt=%06.2f>| |<fl:fmt=%-6.02f>|
|<str>| |<str:fmt=%8s>| |<str:fmt=%-8s>| |<str:fmt=%.4s>|
$java -jar dist/TemplateEngine.jar examples/formatting.tpl > formatting.txt
<ACTIVITY OMITTED FOR BREVITY>...
$cat formatting.txt
|3| |3.141592653589793| | 3.14| |003.14| |3.14 |
|string| | string| |string | |stri|

```

Fig. 4.14: Several formatting examples.

Table 4.1: Trigonometric functions

Description	Function Name
Sine	$\sin(x)$
Cosine	$\cos(x)$
Tangent	$\tan(x)$
Arc Sine	$\text{asin}(x)$
Arc Cosine	$\text{acos}(x)$
Arc Tangent	$\text{atan}(x)$
Arc Tan with 2 parameters	$\text{atan2}(y, x)$
Secant	$\text{sec}(x)$
Cosecant	$\text{cosec}(x)$
Co-tangent	$\text{cot}(x)$
Hyperbolic Sine	$\sinh(x)$
Hyperbolic Cosine	$\cosh(x)$
Hyperbolic Tangent	$\tanh(x)$
Inverse Hyperbolic Sine	$\text{asinh}(x)$
Inverse Hyperbolic Cosine1	$\text{acosh}(x)$
Inverse Hyperbolic Tangent1	$\text{atanh}(x)$

Table 4.2: Log and exponential functions

Description	Function Name
Natural Logarithm	$\ln(x)$
Logarithm base 10	$\log(x)$
Logarithm base 2	$\lg(x)$
Exponential (e^x)	$\exp(x)$
Power	$\text{pow}(x)$

Table 4.3: Statistical functions

Description	Function Name
Average	$\text{avg}(x_1, x_2, x_3, \dots)$
Minimum	$\text{min}(x_1, x_2, x_3, \dots)$
Maximum	$\text{max}(x_1, x_2, x_3, \dots)$

Table 4.4: Rounding functions

Description	Function Name
Round	round(x), round(x, p)
Floor	floor(x)
Ceiling	ceil(x)

Table 4.5: Miscellaneous functions

Description	Function Name
If	if(cond, trueval, falseval)
Str (convert number to string)	str(x)
Absolute Value / Magnitude	abs(x)
Random number (between 0 and 1)	rand()
Modulus	mod(x,y) = x % y
Square Root1	sqrt(x)
Sum	sum(x,y,...)
Binomial coefficients	binom(n, i)
Get object or array member by name or index	get(name,index)/ get(name,child_name)
Get the length of an array or string	length(attribute)
Acquire the left characters in a string	left(string, count)
Acquire the right characters in a string	right(string, count)
Find the index of substr in string	find(string, substr)
Replace all occurrences of X with Y in string	replace(string,x,y)
Determine if a filepath, relative or absolute to the template, exists	exists(filepath)
Get the current date milliseconds since January 1, 1970	current_date()
Get the milliseconds since January 1, 1970 and the provided date (mm-dd-yyyy)	date("mm-dd-yyyy")

A

Assembly decay heat
plot, 43

C

cask surface temperature
plot, 41
clad temperature
plot, 41
criticality
plot, 39

D

decay heat
plot, 39

F

Fulcrum, 54

H

HAC allowable leakage rate
plot, 51
HAC allowable release rate
plot, 51
HAC dose
plot, 42
hg
mercurial, 66

I

ISFSI
view, pad, 37

M

map

Pad Map View, 37

mercurial
hg, 66

N

NCT 2 m dose
plot, 42
NCT allowable leakage rate
plot, 49
NCT allowable release rate
plot, 49
NCT personnel boundary dose
plot, 43
NCT surface dose
plot, 42

P

pad
ISFSI view, 37
plot
Assembly decay heat, 43
cask surface temperature, 41
clad temperature, 41
criticality, 39
decay heat, 39
HAC allowable leakage rate, 51
HAC allowable release rate, 51
HAC dose, 42
NCT 2 m dose, 42
NCT allowable leakage rate, 49
NCT allowable release rate, 49
NCT personnel boundary dose, 43
NCT surface dose, 42
Self-protecting (Nationwide), 38
Self-protecting (selected site), 51

S

Self-protecting (Nationwide)
plot, [38](#)

Self-protecting (selected site)
plot, [51](#)

T

TemplateEngine

attributes, [68](#)

Conditional action, [72](#)

indirect attributes, [68](#)

Iterative expression evaluation, [70](#)

Iterative subtemplate import, [72](#)

Parameterized subtemplate import, [74](#)

Scalar expression evaluation, [70](#)

static text, [68](#)

Subtemplate import, [71](#)

V

view

pad ISFSI, [37](#)