ORNL/TM-2015/726

Nuclear Facility Accident (NFAC) Unit Test Report For HPAC Version 6.3

Ronald W. Lee
Robert W. Morris
C. David Sulfredge

Approved for public release.
Distribution is unlimited.

December, 2015

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE US DEPARTMENT OF ENERGY

DOCUMENT AVAILABILITY

Reports produced after January 1, 1996, are generally available free via the U.S. Department of Energy (DOE) SciTech Connect.

Website: `http:/\/www.osti.gov/scitech`

Reports produced before January 1, 1996, may be purchased by members of the public from the following source.

National Technical Information Service
5825 Port Royal Road
Springfield, VA 22161
Telephone: 703-605-6000 (1-800-553-6847)
TDD: 703-487-4639
Fax: 703-605-6900
E-mail: info@ntis.fedworld.gov
Website: `http:/\/www.ntis.gov/help/ordermethods.aspx`

Reports are available to DOE employees, DOE contractors, Energy Technology Data Exchange (ETDE) representatives, and International Nuclear Information System (INIS) representatives from the following source.

Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831
Telephone: 865-576-8401
Fax: 865-576-5728
E-mail: reports@osti.gov
Website: `http:/\/www.osti.gov/contact.html`

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ORNL/TM-2015/726

Computational Sciences and Engineering Division

NUCLEAR FACILITY ACCIDENT (NFAC) UNIT TEST REPORT FOR HPAC VERSION 6.3

Ronald W. Lee
Robert W. Morris
C. David Sulfredge

Date Published: December, 2015

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831-6285
managed by
UT-BATTELLE, LLC
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

ACRONYMS

AGR	Advanced Gas-Cooled Reactor
API	Application Programming Interface
ATR	Advanced Thermal Reactor
BWR	Boiling Water Reactor
DTRA	Defense Threat Reduction Agency
FBR	Fast Breeder Reactor
GCHWR	Gas-Cooled Heavy Water Reactor
GCR	Gas-Cooled Reactor
GUI	Graphical User Interface
HPAC	Hazard Prediction and Assessment Capability
HTGR	High-Temperature Gas Reactor
HWGCR	Heavy Water Gas-Cooled Reactor
HWLWR	Heavy Water Light Water Reactor
LGR	Light-Water Cooled Graphite-Moderated Reactor
LMFBR	Liquid-Metal Fast Breeder Reactor
MELCOR	Methods for Estimation of Leakages and Consequences of Releases
MWt	Megawatt thermal
NFAC	Nuclear Facility Accident Model
ORIGEN	Oak Ridge Isotope Generator
OSGi	Open Service Gateway Initiative
PHWR	Pressurized Heavy Water Reactor
PIR	Percent Inventory Release
RASCAL	Radiological Assessment Systems for Consequence Analysis
PWR	Pressurized Water Reactor
RBMK	(Russian) Reaktor Bolshoy Moschnosti Kanainy
RTH	Radiological Transport for HPAC
SCIPUFF	Second-order Closure Integrated Puff Model
SGHWR	Steam-Generating Heavy Water Reactor
SPCR	Software Problem/Change Report
VVER	(Russian) Water-Water Energy Reactor

1. OVERVIEW

NFAC's responsibility as an HPAC component is three-fold. First, it must present an interactive graphical user interface (GUI) by which users can view and edit the definition of an NFAC incident. Second, for each incident defined, NFAC must interact with RTH to create activity table inputs and associate them with pseudo materials to be transported via SCIPUFF. Third, NFAC must create SCIPUFF releases with the associated pseudo materials for transport and dispersion. The goal of NFAC unit testing is to verify that the inputs it produces are correct for the source term or model definition as specified by the user via the GUI. As shown in Fig. 1, NFAC contains 22 distinct source term models, all of which are tested.

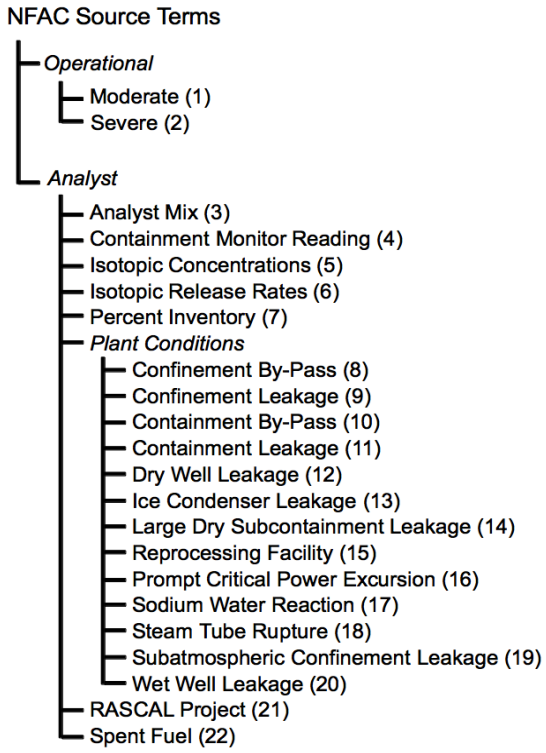

Fig. 1: NFAC Source Terms.

2. TEST APPROACH

Testing is divided into two paths: automated tests and interactive tests. Six automated JUnit4 test suites comprising 87 individual tests have been built and are described in detail below. Each individual test includes one or more assertions challenging specific results or consequences designed to exercise a key functional capability. The granularity of the capability being tested varies by test from an individual component method to an aggregate or composite test involving multiple components and methods. Success or failure of automated tests is an output of the test framework. They are run as unattended batch processes.

The automated tests aggregate information written by NFAC components to the log file. The log output contains all the data regarding activity table inputs fed to RTH and thus can be compared with baseline logs for equivalence. This represents a unit test of NFAC. However, in some test cases we go further to compare the radfile produced by RTH with a baseline radfile, providing in effect an integration test with RTH components. Since the HPAC architecture is such that NFAC is dependent on RTH for integration within HPAC, we feel this level of integration testing is justified in conjunction with corresponding NFAC unit tests that verify NFAC-produced inputs in the NFAC-RTH interface.

The GUI is not conducive to automated tests and thus must be tested interactively. All capabilities in the GUI are tested explicitly as described below.

3. NFACTEST OSGI BUNDLE

NFAC automated tests have been inherited from prior HPAC versions where testing could rely on Ant JUnit tasks. However, HPACs OSGi environment has made it necessary for NFAC to provide its own custom testing framework that allows NFAC's JUnit4 tests to run under OSGi. Further, NFAC's custom framework, the nfactest bundle, includes components for reading log files and radfiles so they can be compared as test assertions. These test framework components are described briefly below and listed as source in appendices.

3.1 NFACLOG

In HPAC-5.3, logging was added to NFAC components to document the activity table inputs fed to RTH for the ultimate generation of the radfile used by SCIPUFF. This was enhanced in HPAC-6.1 to be sufficient for extraction and comparison as part of a test assertion. Note this logging capability was built using java.util.logging and has been extended with the same logging framework. So much effort has been invested in this capability that it is now an integral, unremovable, part of NFAC. Fortunately, with version 6.2 HPAC has reverted to use of java.util.logging as the preferred logging mechanism. The NfacLog class provides the ability to read a log file, extract the activity table inputs logged by NFAC, encapsulate them, and provide equality comparison between NfacLog instances. Refer to Appendix B. for a source listing of this component.

3.2 RADFILE

This component was added to read radfiles generated by RTH and provide equality comparison between instances. The RadFile class accounts for material names that are specific to an individual radfile, matching material names between files so that comparison is possible. Refer to Appendix C. for a source list of this components.

3.3 NFACJUNIT4ADAPTER

This class provides the mechanics of finding and running JUnit4 test suites and cases, while using the OSGi environment for class loading. It borrows heavily from the Ant JUnit tasks and provides similar reporting output. The NfacJUnit4Adapter class uses the Junit4 API to run tests and receive events related to suite and individual test case runs. Refer to Appendix A. for a source listing.

4. AUTOMATED TESTS

4.1 EXTERNALFILEFINDERTEST

This suite is composed of four individual tests to verify correct function of NFAC's PropertyFinder class used to identify and locate ExternalFile instances to be (re)stored for a project. PropertyFinder instances are used by NfacImpl instances in exportFiles() and importFiles() methods now required for correct HPAC project export and import operations, respectively. Tests in alphabetical order are summarized below.

NFAC components tested:

mil.dtra.hpac.models.nfac.CAcomp.data.FacilityDef
mil.dtra.hpac.models.nfac.CAcomp.data.NfacIncident
mil.dtra.hpac.models.nfac.CAcomp.data.PropertyFinder

mil.dtra.hpac.models.nfac.CAcomp.impl.NfacImpl

Test: testAllIncidents

Loads a project containing multiple NfacImpl instances, assigns a custom inventory with an ExternalFile reference to each instance, and invokes the finder() method of a PropertyFinder instance to locate the ExternalFiles. Verifies all ExternalFiles are found.

Test: testFileReleases

Loads a project containing FileRelease instances and ensures all associated ExternalFiles are found.

Test: testNfacImpl

Tests correct behavior of the NfacImpl.exportFiles() method.

Test: testProject

Loads a project with an NFAC incident with a custom inventory and verifies that the findExternalFiles() method of PropertyFinder behaves correctly.

4.2 MODELDEFSTEST

This suite contains 42 individual tests designed to verify proper function of each of the NFAC source models. For each test there is a corresponding HPAC 6 project (*.hpac6*) file or a project export (*.zip*) file. For most tests a radfile (*.rad*) from the baseline project execution is a fixture, and when appropriate a log file (*.log*) is an additional fixture. A ProjectManager instance is used to open the project file, and a DispersionManager instance is used to calculate the project via calls to startDispersion() and waitForProcess(). The project is then saved via ProjectManager.saveProject(), and the radfile is extracted from the project archive (*.hzip*) file or directory, whichever exists. The radfile resulting from the test run and the baseline radfile are read into RadFile instances and compared with RadFile.equals(). For those tests including a log

file fixture, the fixture and log file resulting from the run are read into NfacLog objects and compared with the NfacLog.equals() method. Exact matches determine a successful test. Any difference in RadFile or NfacLog objects represents a test failure. Tests in alphabetical order are summarized below.

NFAC components tested:

mil.dtra.hpac.models.nfac.CAcomp.data.ActivityHash
mil.dtra.hpac.models.nfac.CAcomp.data.AmbClient
mil.dtra.hpac.models.nfac.CAcomp.data.Element
mil.dtra.hpac.models.nfac.CAcomp.data.Facility
mil.dtra.hpac.models.nfac.CAcomp.data.FacilityDB
mil.dtra.hpac.models.nfac.CAcomp.data.FacilityDBMgr
mil.dtra.hpac.models.nfac.CAcomp.data.FacilityDef
mil.dtra.hpac.models.nfac.CAcomp.data.MaterialMode
mil.dtra.hpac.models.nfac.CAcomp.data.ModelDefUtils
mil.dtra.hpac.models.nfac.CAcomp.data.ModelTimes
mil.dtra.hpac.models.nfac.CAcomp.data.ModerateModel
mil.dtra.hpac.models.nfac.CAcomp.data.NfacIncident
mil.dtra.hpac.models.nfac.CAcomp.data.NfacIncidentMgr
mil.dtra.hpac.models.nfac.CAcomp.data.NfacRelease
mil.dtra.hpac.models.nfac.CAcomp.data.Options
mil.dtra.hpac.models.nfac.CAcomp.data.ParticleGroups
mil.dtra.hpac.models.nfac.CAcomp.data.ParticleGroupsMgr
mil.dtra.hpac.models.nfac.CAcomp.data.ReproInventory
mil.dtra.hpac.models.nfac.CAcomp.data.SevereModel

mil.dtra.hpac.models.nfac.CAcomp.data.analyst.AnalystMix
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.AnalystModelUtils
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.ConcentrationUnits
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.ContainmentMonitorReading
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.IsotopeReleases
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.IsotopeValue
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.IsotopicConcentrations
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.IsotopicReleaseRates
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.PIRSubReleaseMode
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.PercentInventory
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.PercentInventoryConstants
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.PercentInventoryRelease
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.PercentInventoryReleaseTree
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.RascalFileReader
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.RascalProject
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.RascalXmlReader
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.ReleaseRate
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.ReleaseRateUnits
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.SpentFuel

mil.dtra.hpac.models.nfac.CAcomp.data.analyst.plantcond.PCConfinementByPass

Test: test10682GroupsExplicit

Tests the new capabilities added for SPCR 10682. Two material processing modes are now supported: the original *Simple* scheme where depositor and non-depositor materials are created for each release, and a new *Groups* mode where separate gas materials are created representing particle sizes and associated gas deposition velocities. Isotopes are apportioned according to ratios established for each MELCOR group. Two sets of apportionment ratios are provided, one for BWR reactors and another for PWRs (also applied to VVER). This test ensures the apportionment is correct and the corresponding materials are created with the Groups material mode and the Explicit percent inventory sub-release mode.

Facility:	Peach Bottom-2 (BWR Mk-1)
Operating Power:	3414.0 MWt
Material Mode:	Groups
Sub-release Mode:	Explicit
Source Term:	Percent Inventory
Shutdown Duration:	0
Release Duration:	7.0 h

Releases:

#	Duration	Percentages by Group
0	0.5 h	NobleGas=0.01, AlkaMetal=0.6, Chalcogen=0.35
1	2.0 h	NobleGas=0.09, AlkaMetal=1.7, AlkaEarth=0.03, Chalcogen=1.45
2	4.5 h	NobelGas=0.8, AlkaMetal=0.9, Chalcogen=0.45

Test: test10682GroupsNone

Tests proper activity apportionment and material creation for the Groups material mode and None percent inventory sub-release mode.

Facility:	Peach Bottom-2 (BWR Mk-1)
Operating Power:	3414.0 MWt
Material Mode:	Groups
Sub-release Mode:	None
Source Term:	Percent Inventory
Shutdown Duration:	0
Release Duration:	7.0 h

Releases:

#	Duration	Percentages by Group
0	0.5 h	NobleGas=0.01, AlkaMetal=0.6, Chalcogen=0.35
1	2.0 h	NobleGas=0.09, AlkaMetal=1.7, AlkaEarth=0.03, Chalcogen=1.45
2	4.5 h	NobelGas=0.8, AlkaMetal=0.9, Chalcogen=0.45

Test: test10682PwrGroupsNone

Tests proper activity apportionment and material creation for the Groups material mode and None percent inventory sub-release mode.

Facility:	Arkansas One-1 (PWR)
Operating Power:	2550.0 MWt
Material Mode:	Groups
Sub-release Mode:	None
Source Term:	Percent Inventory
Shutdown Duration:	1.0 h
Release Duration:	7.0 h

Releases:

#	Duration	Percentages by Group
0	1.0 h	0
1	5.0 h	NobleGas=3.0, AlkaMetal=0.003, Halogens=0.003, Chalcogen=0.03, Platinoid=1.68e-6, Tetravalent=4.0e-7, Trivalent=6.0e-7
2	2.0 h	NobleGas=17, AlkaMetal=0.07, AlkaEarth=8.95e-5, Halogens=0.089, Chalcogen=0.64,Platinoid=6.2e-6, Tetravalent=7.0e-7, Trivalent=1.6e-6

Test: test10682RascalGroupsNone

Tests proper activity apportionment and material creation for a RASCAL source term and the Groups material mode.

Facility:	Robinson-2 (PWR)
Operating Power:	2295.0 MWt
Material Mode:	Groups
Source Term:	Rascal Project
Units:	Ci
Release Times:	0.0, 2.0, 4.0, 6.0, 8.0 h
Isotopes:	Ba-140, Ce-141, Ce-144, Cs-137, I-131, Kr-85, La-140, Mo-99, Nb-95, Nd-147, Pr-143, Pu-239, Ru-103, Ru-106, Sr-89, Sr-90, Te-129, Xe-133, Xe-135, Y-91, Zr-95

Test: test10682RascalSimpleNone

Tests proper activity apportionment and material creation for a RASCAL source term and the Groups material mode.

Facility:	Robinson-2 (PWR)
Operating Power:	2295.0 MWt
Material Mode:	Simple
Source Term:	Rascal Project
Units:	Ci
Release Times:	0.0, 2.0, 4.0, 6.0, 8.0 h
Isotopes:	Ba-140, Ce-141, Ce-144, Cs-137, I-131, Kr-85, La-140, Mo-99, Nb-95, Nd-147, Pr-143, Pu-239, Ru-103, Ru-106, Sr-89, Sr-90, Te-129, Xe-133, Xe-135, Y-91, Zr-95

Test: test10682SimpleExplicit

Tests that activity apportionment and per-particle-size material creation do not occur for a scenario using the Explicit percent inventory sub-release mode.

Facility:	Peach Bottom-2 (BWR Mk-1)
Operating Power:	3414.0 MWt
Material Mode:	Simple
Sub-release Mode:	Explicit
Source Term:	Percent Inventory
Shutdown Duration:	0
Release Duration:	7.0 h

Releases:

#	Duration	Percentages by Group
0	0.5 h	NobleGas=0.01, AlkaMetal=0.6, Chalcogen=0.35,
1	2.0 h	NobleGas=0.09, AlkaMetal=1.7, AlkaEarth=0.03, Chalcogen=1.45,
2	4.5 h	NobleGas=0.8, AlkaMetal=0.9, Chalcogen=0.45

Test: test10682SimpleNone

Tests that activity apportionment and per-particle-size material creation do not occur for a scenario using the None percent inventory sub-release mode.

Facility: Peach Bottom-2 (BWR Mk-1)
Operating Power: 3414.0 MWt
Material Mode: Simple
Sub-release Mode: None
Source Term: Percent Inventory
Shutdown Duration: 0
Release Duration: 7.0 h
Releases:

#	Duration	Percentages by Group
0	0.5 h	NobleGas=0.01, AlkaMetal=0.6, Chalcogen=0.35,
1	2.0 h	NobleGas=0.09, AlkaMetal=1.7, AlkaEarth=0.03, Chalcogen=1.45,
2	4.5 h	NobleGas=0.8, AlkaMetal=0.9, Chalcogen=0.45

Test: test10682UnsupportedGroupsNone

Tests that activity apportionment and per-particle-size material creation do not occur even though requested via the Groups material mode for an unsupported facility type and the None percent inventory sub-release mode.

Facility: Hartlepool-1 (AGR)
Operating Power: 1785.0 MWt
Material Mode: *disabled*
Sub-release Mode: None
Source Term: Percent Inventory
Shutdown Duration: 1.0 h
Release Duration: 7.0 h
Releases:

#	Duration	Percentages by Group
0	1.0 h	0
1	2.0 h	NobleGas=3.0, AlkaMetal=0.003, Halogens=0.003, Chalcogen=0.03, Platinoid=1.68e-6, Tetravalent=4.0e-7, Trivalent=6.0e-7
2	5.0 h	NobleGas=17, AlkaMetal=0.07, AlkaEarth=8.95e-5, Chalcogen=0.64, Platinoid=6.2e-6, Tetravalent=7.0e-7, Trivalent=1.6e-6

Test: test10682VverGroupsNone

Tests proper activity apportionment and material creation occurs for a VVER facility based on the model for a PWR using the Groups material mode and the None percent inventory sub-release mode.

Facility:	Zaporozhe-1 (VVER-1000)
Operating Power:	2850.0 MWt
Material Mode:	Groups
Sub-release Mode:	None
Source Term:	Percent Inventory
Shutdown Duration:	1.0 h
Release Duration:	7.0 h

Releases:

#	Duration	Percentages by Group
0	1.0 h	0
1	2.0 h	NobleGas=3.0, AlkaMetal=0.003, Halogens=0.003, Chalcogen=0.03, Platinoid=1.68e-6, Tetravalent=4.0e-7, Trivalent=6.0e-7
2	5.0 h	NobleGas=17, AlkaMetal=0.07, AlkaEarth=8.95e-5, Chalcogen=0.64, Platinoid=6.2e-6, Tetravalent=7.0e-7, Trivalent=1.6e-6

Test: test11653

Explicitly tests the fix for SPCR 11653 to ensure a custom inventory is accounted for when determining the available plant conditions source terms.

Facility:	Arkansas One-1 (PWR)
Operating Power:	2550.0 MWt
Custom Inventory:	Created from kewaunee.f71 ORIGEN file
Material Mode:	Simple
Source Term:	Mix Specified by Analyst
Gross Release Rate:	1.0 Ci/s
Release Percentages:	
Nobel Gases:	98.0%
Halogens:	2.0%

Test: testAnalystMix

Tests an AnalystMix model instance with the following properties:

Facility:	Brunswick-1 (BWR Mk-1)		
Operating Power:	2949.0 MWt		
Material Mode:	Simple	Nobel Gases:	98.0%
Source Term:	Mix Specified by Analyst	Halogens:	2.0%
Gross Release Rate:	100.0 Ci/s		
Release Percentages:			

Test: testContainmentMonitorReading

Tests a ContainmentMonitorReading model instance with the following properties:

Facility:	Brunswick-1 (BWR Mk-1)
Operating Power:	2949.0 MWt
Material Mode:	Simple
Source Term:	Containment Monitor Reading
Leak Rate:	10%/h
Monitor Location:	Wet Well
Monitor Reading:	25.0 R/h
Release Path:	Unfiltered
Sprays:	Off

Test: testIsotopicConcentrations

Tests a IsotopicConcentrations model instance with the following properties:

Facility:	Brunswick-1 (BWR Mk-1)		
Operating Power:	2949.0 MWt	Cs-130:	130.0
Material Mode:	Simple	Cs-131:	131.0
Source Term:	Isotopic Concentrations	I-130:	130.0
Concentration Units:	kCi/cc	I-132:	132.0
Release Rate:	20.0 cc/s	I-133:	133.0
Isotope Values:			

Test: testIsotopicReleaseRates

Tests a IsotopicReleaseRates model instance with the following properties:

		Cs-130:	130.0
Facility:	Brunswick-1 (BWR Mk-1)	Cs-131:	131.0
Operating Power:	2949.0 MWt	Cs-132:	132.0
Material Mode:	Simple	Cs-134:	134.0
Source Term:	Isotopic Release Rates	I-130:	130.0
Release Units:	kCi/s	I-131:	131.0
Isotope Values:		I-132:	132.0
		I-133:	133.0

Test: testModerate

Tests an operational ModerateModel instance with properties:

Facility:	Brunswick-1 (BWR Mk-1)
Operating Power:	2949.0 MWt

Test: testMultiFive

Tests a project with five NFAC incidents.

Facility:	Calvert Cliffs-1 (PWR)
Operating Power:	2535.0 MWt
Materal Mode:	Simple
Source Term:	Steam Generator Tube Rupture (Coolant)
Partitioned Generator:	false
Release Rate:	1 Tube (35%/h)
Release Source:	Steam Jet Air Ejector

Facility:	Limerick-1 (BWR Mk-2)
Operating Power:	3165.0 MWt
Materal Mode:	Simple
Source Term:	Dry Well Leakage/Failure (BWR Containment)
Core Condition:	Vessel Melt Through
Leak Rate:	50%/h (release duration=2.0 h)
Filtered Release Path:	false
Sprays On:	false

Facility:	Oyster Creek (BWR Mk-1)
Operating Power:	1950.0 MWt
Materal Mode:	Simple
Source Term:	Containment Monitor Reading
Leak Rate:	50%/h (release duration=2.0 h)
Monitor Location:	Wet Well
Monitor Reading:	20.0 R/h
Release Path:	Unfiltered
Sprays:	Off

Facility:	Peach Bottom-2 (BWR Mk-1)
Operating Power:	3414.0 MWt
Materal Mode:	Simple
Source Term:	Spent Fuel/Spent Fuel Pool
Fuel Condition:	Fuel Cladding Failure
Number of Batches:	1
Release Path:	Unfiltered
Sprays:	Off
Time Last Batch in Pool:	*same as ReleaseToContainment*

Facility:	Salem-1 (PWR)
Operating Power:	3507.0 MWt
Materal Mode:	Simple
Source Term:	Large, Dry, or Subatmospheric Containment Leakage/Failure
Core Condition:	Gap Release
Filtered Release Path:	false
Leak Rate:	50%/h (release duration=2.0 h)
Sprays On:	false

Test: testPCConfinementByPass

Tests a PCConfinementByPass plant conditions model instance with properties:

Facility:	Hartlepool-1 (AGR)
Operating Power:	1785.0 MWt
Material Mode:	*disabled*
Source Term:	Plant Conditions: Bypass of Confinement
Core Condition:	Gap Release
Leak Rate:	25%/h (release duration=4.0 h)

Test: testPCConfinementLeakage

Tests a PCConfinementLeakage plant conditions model instance with properties:

Facility:	Novovoronezh-3 (VVER-400/230)
Operating Power:	1155.0 MWt
Material Mode:	Simple
Source Term:	Plant Conditions: Confinement Leakage/Failure
Core Condition:	Gap Release
Leak Rate:	50%/h (release duration=2.0 h)

Test: testPCContainmentByPass

Tests a PCContainmentByPass plant conditions model instance with properties:

Facility:	Hamaoka-3 (BWR Mk-1)
Operating Power:	3168.0 MWt
Material Mode:	Simple
Source Term:	Plant Conditions: Bypass of Containment
Core Condition:	Gap Release
Leak Rate:	100%/h (release duration=1.0 h)
Filtered Release Path:	false

Test: testPCContainmentLeakage

Tests a PCContainmentLeakage plant conditions model instance with properties:

Facility:	Beloyarski-3 (BN-600) (LMFBR)
Operating Power:	1680.0 MWt
Material Mode:	*disabled*
Source Term:	Plant Conditions: Containment Leakage/Failure
Core Condition:	Gap Release
Leak Rate:	50%/h (release duration=2.0 h)
Filtered Release Path:	false

Test: testPCDryWellLeakage

Tests a PCDryWellLeakage plant conditions model instance with properties:

Facility:	Hamaoka-3 (BWR Mk-1)
Operating Power:	3168.0 MWt
Material Mode:	Simple
Source Term:	Plant Conditions: Dry Well Leakage/Failure (BWR Containment)
Core Condition:	Gap Release
Filtered Release Path:	false
Leak Rate:	100%/h (release duration=1.0 h)
Sprays On:	false

Test: testPCIceCondenserContainmentCoreDamage

Tests a PCIceCondenserContainment plant conditions model instance with properties:

Facility:	Watts Bar-1 (PWR)
Operating Power:	3465.0 MWt
Material Mode:	Simple
Source Term:	Plant Conditions: Ice Condenser Containment Leakage/Failure
Core Condition:	In-Vessel Severe Core Damage
Fans On:	true
Filtered Release Path:	true
Ice Bed Exhausted:	true
Leak Rate:	50%/h (release duration=2.0 h)
Sprays On:	true

Test: testPCIceCondenserContainmentGapRelease

Tests a PCIceCondenserContainment plant conditions model instance with properties:

Facility:	Watts Bar-1 (PWR)
Operating Power:	3465.0 MWt
Material Mode:	Simple
Source Term:	Plant Conditions: Ice Condenser Containment Leakage/Failure
Core Condition:	Gap Release
Fans On:	false
Filtered Release Path:	false
Ice Bed Exhausted:	false
Leak Rate:	50%/h (release duration=2.0 h)
Sprays On:	false

Test: testPCIceCondenserContainmentMeltThrough

Tests a PCIceCondenserContainment plant conditions model instance with properties:

Facility:	Watts Bar-1 (PWR)
Operating Power:	3465.0 MWt
Material Mode:	Simple
Source Term:	Plant Conditions: Ice Condenser Containment Leakage/Failure
Core Condition:	Vessel Melt Through
Fans On:	false
Filtered Release Path:	false
Ice Bed Exhausted:	false
Leak Rate:	50%/h (release duration=2.0 h)
Sprays On:	false

Test: testPCLargeDrySubcontainmentLeakage

Tests a PCLargeDrySubcontainmentLeakage plant conditions model instance with properties:

Facility:	Beaver Valley-1 (PWR)
Operating Power:	2733.0 MWt
Material Mode:	Simple
Source Term:	Plant Conditions: Large, Dry, or Subatmospheric Containment Leakage/Failure
Core Condition:	Gap Release
Filtered Release Path:	false
Leak Rate:	100%/h (release duration=1.0 h)
Sprays On:	false

Test: testPCPromptCriticalPowerExcursion

Tests a PCPromptCriticalPowerExcursion plant conditions model instance with properties:

Facility:	Bilibino Unit A (RBMK)
Operating Power:	33.0 MWt
Material Mode:	*disabled*
Source Term:	Plant Conditions: Prompt Critical Power Excursion
Core Involvement:	Total

Test: testPCReproFacility

Tests a PCReproFacility plant conditions model instance with properties:

Facility:	Savannah River (Reprocessing)
Material Mode:	*disabled*
Source Term:	Plant Conditions: Reprocessing Facility
Plant Throughput:	1.0 tonnes/day
Release Rate:	100%/h (release duration=1.0 h)
Release Fraction by Component:	
Aqueous Waste Treatment:	0.5
Dissolution: :	1.0
Feed Adjustment and Accountability: :	1.0
Pu Recovery: :	0.5
Solvent Treatment: :	0.5
U-Pu Co-Decontamination, Partitioning, and U Purification:	1.0

Test: testPCSodiumWaterReaction

Tests a PCSodiumWaterReaction plant conditions model instance with properties:

Facility:	Phenix (LMFBR)
Custom Inventory:	stlauren_1.avc
Operating Power:	699.0 MWt
Material Mode:	*disabled*
Source Term:	Plant Conditions: Sodium-Water Reaction
Core Condition:	Gap Release
Leak Rate:	50%/h (release duration=2.0 h)

Test: testPCSteamTubeRuptureCoreDamage

Tests a PCSteamTubeRupture plant conditions model instance with properties:

Facility:	Dungeness B-1 (AGR)
Operating Power:	1560.0 MWt
Material Mode:	*disabled*
Source Term:	Plant Conditions: Steam Generator Tube Rupture (Coolant)
Coolant Concentration:	In-Vessel Severe Core Damage
Partitioned Generator:	false
Release Rate:	1 Tube (35%/h)
Release Source:	Safety Valve

Test: testPCSteamTubeRuptureGapRelease

Tests a PCSteamTubeRupture plant conditions model instance with properties:

Facility:	Dungeness B-1 (AGR)
Operating Power:	1560.0 MWt
Material Mode:	*disabled*
Source Term:	Plant Conditions: Steam Generator Tube Rupture (Coolant)
Coolant Concentration:	Gap Release
Partitioned Generator:	false
Release Rate:	1 Tube (35%/h)
Release Source:	Safety Valve

Test: testPCSubatmosphericConfinementLeakage

Tests a PCSubatmosphericConfinementLeakage plant conditions model instance with properties:

Facility:	Rovno-1 (VVER-440/213)
Operating Power:	1143.0 MWt
Material Mode:	Simple
Source Term:	Plant Conditions: Subatmospheric Confinement Leakage/Failure
Core Condition:	Gap Release
Leak Rate:	100%/h (release duration=1.0 h)
Pool Suppression System:	true

Test: testPCWetWellLeakage

Tests a PCWetWellLeakage plant conditions model instance with properties:

Facility:	Hamaoka-3 (BWR Mk-1)
Operating Power:	3168.0 MWt
Material Mode:	Simple
Source Term:	Plant Conditions: Wet Well Leakage/Failure (BWR Containment)
Core Condition:	Gap Release
Leak Rate:	100%/h (release duration=1.0 h)
Filtered Release Path:	false
Wet Well:	Saturated

Test: testPercentInventoryNoShutdownAsIs

Tests a PercentInventory model instance with no shutdown duration defined in the releases and no override of the model event times.

Facility:	Watts Bar-1 (PWR)
Operating Power:	3465.0 MWt
Material Mode:	Simple
Sub-release Mode:	Average
Source Term:	Percent Inventory
Shutdown Duration:	0
Release Duration:	44.9 h

Releases:

#	Duration	Percentages by Group
0	1.9 h	NobleGas=3.0, AlkaMetal=0.003, AlkaEarth=1e-5, Halogens=0.003, Chalcogen=0.03, Platinoid=1.2e-7
1	32.0 h	NobleGas=3.0, AlkaMetal=0.003, Halogens=0.003, Chalcogen=0.03, Platinoid=1.68e-6, Tetravalent=4.0e-7, Trivalent=6.0e-7
2	11.0 h	NobleGas=17, AlkaMetal=0.07, AlkaEarth=8.95e-5, Halogens=8.9e-2, Chalcogen=0.64, Platinoid=6.2e-6, Tetravalent=7.0e-7, Trivalent=1.6e-6

Test: testPercentInventoryNoShutdownForced

Tests a PercentInventory model instance with no shutdown duration defined in the releases, but model event times are overridden to force a shutdown time.

Facility:	Watts Bar-1 (PWR)
Operating Power:	3465.0 MWt
Material Mode:	Simple
Sub-release Mode:	Average
Source Term:	Percent Inventory
Shutdown Duration:	10 m
Release Duration:	44.9 h

Releases:

#	Duration	Percentages by Group
0	1.9 h	NobleGas=3.0, AlkaMetal=0.003, AlkaEarth=1e-5, Halogens=0.003, Chalcogen=0.03, Platinoid=1.2e-7
1	32.0 h	NobleGas=3.0, AlkaMetal=0.003, Halogens=0.003, Chalcogen=0.03, Platinoid=1.68e-6, Tetravalent=4.0e-7, Trivalent=6.0e-7
2	11.0 h	NobleGas=17, AlkaMetal=0.07, AlkaEarth=8.95e-5, Halogens=8.9e-2, Chalcogen=0.64, Platinoid=6.2e-6, Tetravalent=7.0e-7, Trivalent=1.6e-6

Test: testPercentInventoryWithShutdownAsIs

Tests a PercentInventory model instance with a shutdown duration defined in the releases and no override of the model event times.

Facility: Peach Bottom-2 (BWR Mk-1)
Operating Power: 3414.0 MWt
Material Mode: Simple
Sub-release Mode: Average
Source Term: Percent Inventory
Shutdown Duration: 41.7 m
Release Duration: 11.3 h
Releases:

#	Duration	Percentages by Group
0	41.7 m	0
1	41.7 m	NobelGas=0.01, AlkaMetal=0.6, Chalcogen=0.35
2	83.33 m	NobelGas=0.09, AlkaMetal=1.7, AlkaEarth=0.03, Chalcogen=1.45
3	250.0 m	NobelGas=0.8, AlkaMetal=0.9, Chalcogen=0.45
4	303.0 m	NobelGas=0.3, AlkaMetal=0.1, Chalcogen=0.22, LessVolatile=0.27

Test: testPercentInventoryWithShutdownForced

Tests a PercentInventory model instance with a shutdown duration defined in the releases, but model event times are overridden to force no shutdown time.

Facility: Peach Bottom-2 (BWR Mk-1)
Operating Power: 3414.0 MWt
Material Mode: Simple
Sub-release Mode: Average
Source Term: Percent Inventory
Shutdown Duration: 41.7 m
Release Duration: 11.3 h
Releases:

#	Duration	Percentages by Group
0	41.7 m	0
1	41.7 m	NobelGas=0.01, AlkaMetal=0.6, Chalcogen=0.35
2	83.33 m	NobelGas=0.09, AlkaMetal=1.7, AlkaEarth=0.03, Chalcogen=1.45
3	250.0 m	NobelGas=0.8, AlkaMetal=0.9, Chalcogen=0.45
4	303.0 m	NobelGas=0.3, AlkaMetal=0.1, Chalcogen=0.22, LessVolatile=0.27

Test: testPIRSubReleasesAverage

Tests a PercentInventory model instance with the Average PIR sub-release mode.

Facility: Peach Bottom-2 (BWR Mk-1)
Operating Power: 3414.0 MWt
Material Mode: Simple
Sub-release Mode: Average
Source Term: Percent Inventory
Shutdown Duration: 30.0 m
Release Duration: 48.0 h
Releases:

#	Duration	Percentages by Group
0	48.0 h	NobelGas=100

Test: testPIRSubReleasesExplicit

Tests a PercentInventory model instance with the Explicit PIR sub-release mode.

Facility: Peach Bottom-2 (BWR Mk-1)
Operating Power: 3414.0 MWt
Material Mode: Simple
Sub-release Mode: Explicit
Source Term: Percent Inventory
Shutdown Duration: 30.0 m
Release Duration: 48.0 h
Releases:

#	Duration	Percentages by Group
0	48.0 h	NobelGas=100

Test: testPIRSubReleasesNone

Tests a PercentInventory model instance with the None PIR sub-release mode.

Facility: Peach Bottom-2 (BWR Mk-1)
Operating Power: 3414.0 MWt
Material Mode: Simple
Sub-release Mode: None
Source Term: Percent Inventory
Shutdown Duration: 30.0 m
Release Duration: 48.0 h
Releases:

#	Duration	Percentages by Group
0	48.0 h	NobelGas=100

Test: testRascalProject

Tests a RascalProject model instance with properties:

Facility:	Brunswick-1 (BWR Mk-1)
Operating Power:	2949.0 MWt
Material Mode:	Simple
Source Term:	Rascal Project
Units:	Ci
Release Times:	0.0, 0.25, 0.5, 0.75, ..., 8.0 h
Isotopes:	Ba-139, Ba-140, Cs-134, Cs-136, Cs-137, Cs-138, I-131, I-132, I-133, I-134, I-135, Kr-83m, Kr-85, Kr-85m, Kr-87, Kr-88, La-140, Mo-99, Rb-86, Rb-88, Rh-103, Rh-105, Ru-103, Ru-105, Ru-106, Sb-127, Sb-129, Sr-89, Sr-90, Sr-91, Sr-92, Tc-99m, Tc-127, Tc-127m, Tc-129, Tc-129m, Tc-131, Tc-131m, Tc-132, Xc-131m, Xc-133, Xc-133m, Xc-135, Xc-135m, Xc-138, Y-90, Y-91, Y-91m, Y-92

Test: testSevere

Tests an operational SevereModel instance with properties:

Facility:	Brunswick-1 (BWR Mk-1)
Operating Power:	2949.0 MWt

Test: testSpentFuel

Tests an operational SpentFuel instance with properties:

Facility:	Brunswick-1 (BWR Mk-1)
Operating Power:	2949.0 MWt
Source Term:	Spent Fuel/Spent Fuel Pool
Fuel Condition:	Fuel Cladding Failure
Number of Batches:	1
Release Path:	Unfiltered
Sprays:	Off
Time Last Batch in Pool:	*6 months prior to ReleaseToContainment*

4.3 MODELTIMESTEST

One of the most critical aspects of NFAC processing is correct management of event times within a scenario. There are six possible events whose times define or are defined by associated durations:

Event	Associated Duration	From Event
Shutdown / Start of Decay		
Release to Containment	Containment	Start of Decay
Release to Environment	Shutdown	Start of Decay
	Holdup	Release to Containment
End of Release	Release	Release to Environment
End of Dispersion	Dispersion	End of Release
End of Exposure	Exposure	End of Dispersion

For some NFAC source models, individual events have no meaning and thus are ignored and not displayed in the GUI. Some times are computed directly from the model definition and thus are made uneditable (although displayed) in the GUI. Which times are displayed and/or editable are determined based on the model, and uneditable times must be computed based on the model parameters. Finally, uneditable times can be forced to be editable via the Advanced Mode toggle button in the When tab. The purpose of this test suite is to exercise event times management against various models and conditions. The eight tests are described below. Each unique times management class is tested.

NFAC components tested:

mil.dtra.hpac.models.nfac.CAcomp.data.ModelTimes

mil.dtra.hpac.models.nfac.CAcomp.data.times.AnalystMixTimesMgr
mil.dtra.hpac.models.nfac.CAcomp.data.times.IsotopicConcentrationsTimesMgr
mil.dtra.hpac.models.nfac.CAcomp.data.times.IsotopicReleaseRatesTimesMgr
mil.dtra.hpac.models.nfac.CAcomp.data.times.ModelTimesMgr
mil.dtra.hpac.models.nfac.CAcomp.data.times.ModelTimesMgrFactory
mil.dtra.hpac.models.nfac.CAcomp.data.times.OperationalTimesMgr
mil.dtra.hpac.models.nfac.CAcomp.data.times.PCContainmentByPassTimesMgr
mil.dtra.hpac.models.nfac.CAcomp.data.times.PercentInventoryTimesMgr
mil.dtra.hpac.models.nfac.CAcomp.data.times.RascalProjectTimesMgr
mil.dtra.hpac.models.nfac.CAcomp.data.times.SpentFuelTimesMgr

Test: testAnalystMix

This test verifies the following:

- The ModelTimesMgrFactory.getInstance() method correctly returns an AnalystMixTimesMgr instance for an AnalystMix model.
- The default times based on test input times are correctly set by the AnalystMixTimesMgr instance.
- After explicit settings of editable times, the Release to Environment and Release to Containment times are equivalent.

Test: testIsotopicConcentrations

This test verifies the following:

- The ModelTimesMgrFactory.getInstance() method correctly returns an IsotopicConcentrationsTimesMgr instance for an IsotopicConcentrations model.
- The default times based on test input times are correctly set by the IsotopicConcentrationsTimesMgr instance.
- After explicit settings of editable times, the Start of Decay, Release to Containment, and Release to Environment times are equivalent.

Test: testIsotopicReleaseRates

This test verifies the following:

- The ModelTimesMgrFactory.getInstance() method correctly returns an IsotopicReleaseRatesTimesMgr instance for an IsotopicReleaseRates model.
- The default times based on test input times are correctly set by the IsotopicReleaseRatesTimesMgr instance.
- After explicit settings of editable times, the Start of Decay, Release to Containment, and Release to Environment times are equivalent.

Test: testPCContainmentByPass

This test verifies the following:

- The ModelTimesMgrFactory.getInstance() method correctly returns a PCContainmentByPassTimesMgr instance for an PCContainmentByPass model.
- The default times based on test input times are correctly set by the PCContainmentByPassTimesMgr instance.
- After explicit settings of editable times, the Start of Decay and Release to Containment times are equivalent, and the Shutdown, Release, Dispersion, and Exposure durations are all correct.

Test: testPercentInventoryNoShutdown

This test verifies the following:

- The ModelTimesMgrFactory.getInstance() method correctly returns a PercentInventoryTimesMgr instance for a PercentInventory model.
- After explicit setting of editable times, the Release to Containment and Release to Environment times are equivalent, the Shutdown Duration is zero, and the Release, Dispersion, and Exposure durations are correct.
- After explicit setting of editable times in lenient mode, the Release to Containment and Release to Environment times are equivalent, and the Shutdown, Release, Dispersion, and Exposure durations are correct for the input times.

Test: testPercentInventoryShutdown

This test verifies the following:

- The ModelTimesMgrFactory.getInstance() method correctly returns a PercentInventoryTimesMgr instance for a PercentInventory model.
- After explicit setting of editable times, the Release to Containment and Release to Environment times are equivalent, and the Shutdown, Release, Dispersion, and Exposure durations are correct.
- After explicit setting of editable times in lenient mode, the Release to Containment and Release to Environment times are equivalent, and the Shutdown, Release, Dispersion, and Exposure durations are correct for the input times.

Test: testRascalProject

This test verifies the following:

- The ModelTimesMgrFactory.getInstance() method correctly returns a RascalProjectTimesMgr instance for an RascalProject model.
- After explicit setting of editable times, the Start of Decay, Release to Containment and Release to Environment times are equivalent.

Test: testSpentFuel

This test verifies the following:

- The ModelTimesMgrFactory.getInstance() method correctly returns a SpentFuelTimesMgr instance for an SpentFuel model.
- After explicit setting of editable times, the Start of Decay and Release to Containment times are equivalent, and the Holdup, Shutdown, Release, Dispersion, and Exposure durations are correct.

4.4 PARTICLEGROUPSMGRTEST

The major enhancement for NFAC between versions 6.2 and 6.3 was the addition of a new mode for processing materials. Previously, NFAC represented each activity vector with two materials, a depositor and an optional non-depositor for any noble gas isotopes. As per SPCR 10682, in addition to the Simple depositor/non-depositor approach, a Groups mode was added in which up to eleven materials can be created for each release. Each material represents a particle size group with an associated gas deposition velocity. Tables specifying apportionments into the particle size groups for each MELCOR group are used to assign activities in each activity vector by MELCOR group. Apportionment tables for supported facility types are specified in the NFAC *defaults.properties* data file.

This test suite is comprised of seven tests that ensure the apportionment tables are properly read and activity apportionment and material creation are correctly processsed.

NFAC components tested:

mil.dtra.hpac.models.nfac.CAcomp.data.ParticleGroups
mil.dtra.hpac.models.nfac.CAcomp.data.ParticleGroupsMgr

Test: testBWRCounts

Checks that the number of particle size groups, MELCOR groups, and table entries for each MELCOR group associated with the BWR facility type, as read from *defaults.properties*, are correct.

Test: testBWRValues

Checks that the BWR table entries giving apportionments for each MELCOR group and particle group, as read from *defaults.properties* are correct.

Test: testPWRCounts

Checks that the number of particle size groups, MELCOR groups, and table entries for each MELCOR group associated with the PWR facility type, as read from *defaults.properties*, are correct.

Test: testPWRValues

Checks that the PWR table entries giving apportionments for each MELCOR group and particle group, as read from *defaults.properties*, are correct.

Test: testReadCounts

Checks that the number of particle group tables and the list of associated facility types, as read from *defaults.properties*, are correct.

Test: testTypeMap

Verifies that the table associated with each reactor type and subtype are correct if the type/subtype should be supported and are not specified otherwise.

Test: testWriteAndRead

Verifies that the java.util.Properties entries as read from *defaults.properties* can be written and read back with the same resulting deserialized objects.

4.5 SOURCETERMTABLETEST

Another critical piece of NFAC functionality is determination of which of the 22 source models are valid or can be applied to a specific facility definition. A facility definition includes the facility selected from NFAC's facility "database" as well as a reference to any custom inventory file. NFAC's SourceTermTable and PlantConditionTables classes encapsulate the capability of determining available source terms. They are "data driven" in that they read the *source_terms.data* and *plntcond.dat* files, respectively, to drive the determination process.

This test suite includes 25 individual tests to verify that the source model availability process functions correctly.

NFAC components tested:

mil.dtra.hpac.models.nfac.CAcomp.data.Facility
mil.dtra.hpac.models.nfac.CAcomp.data.FacilityDB
mil.dtra.hpac.models.nfac.CAcomp.data.FacilityDBMgr
mil.dtra.hpac.models.nfac.CAcomp.data.FacilityDef

mil.dtra.hpac.models.nfac.CAcomp.data.analyst.PlantConditionTables
mil.dtra.hpac.models.nfac.CAcomp.data.analyst.SourceTermTable

Test: testAGRCompatibility

Explicitly tests for individual model compatibility via the SourceTermTable.checkCompatability() method. The Dungeness B-1 and Windscale AGR facilities (without custom inventories) are tested against the following models with expected compatibility:

Dungeness B-1

AnalystMix	yes
ContainmentMonitorReading	no
IsotopicConcentrations	yes
IsotopicReleaseRates	yes
ModerateModel	no
PCLargeDrySubContainment	no
PercentInventory	yes
SevereModel	no
SpentFuel	no

Windscale AGR

PCConfinementByPass	yes
PCConfinementLeakage	yes
PCContainmentByPass	no
PCContainmentLeakage	no
PCDryWellLeakage	no
PCIceCondenserContainment	no
PCLargeDrySubContainmentLeakage	no
PCPromptCriticalPowerExcursion	no
PCReproFacility	no
PCSodiumWaterReaction	no
PCSteamTubeRupture	yes
PCSubatmosphericConfinementLeakage	no
PCWetWellLeakage	no

Test: testAGRSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following AGR facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Dungeness B-1	Windscale AGR[1]	Hunterston B-1[2]

1 Inventory filename set explicitly
2 Inventory filename cleared explicitly

Test: testBWR0SourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following BWR type-0 facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Leibstadt	Brunsbuettel (kkb)	Leibstadt[2]

2 Inventory filename cleared explicitly

Test: testBWR1SourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following BWR type-1 facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Santa Maria De Garona	Fukushima-Daiichi-1	Santa Maria De Garona[2]

2 Inventory filename cleared explicitly

Test: testBWR2SourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following BWR type-2 facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Fukushima-Daini-1	Fukushima-Daini-1[3]	Fukushima-Daini-1[2]

2 Inventory filename cleared explicitly
3 Explicitly set inactive

Test: testBWR3SourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following BWR type-3 facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Cofrentes	Cofrentes[3]	Cofrentes[2]

2 Inventory filename cleared explicitly
3 Explicitly set inactive

Test: testFBRSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following FBR facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Monju[1]	KNK II[1]	Monju

1 Inventory filename set explicitly

Test: testGCHWRSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following GCHWR facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Monts D'Arree[1]	Monts D'Arree	Monts D'Arree[4]

1 Inventory filename set explicitly
4 Explicitly set active

Test: testGCRSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following GCR facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Oldbury-1	Oldbury-2	Latina

Test: testHTGRSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following HTGR facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Peach Bottom[1,4]	Peach Bottom[1]	Peach Bottom

1 Inventory filename set explicitly
4 Explicitly set active

Test: testHWGCRSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following HWGCR facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
A-1 Bohunice[1,4]	A-1 Bohunice[1]	A-1 Bohunice

1 Inventory filename set explicitly
4 Explicitly set active

Test: testHWLWRSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following HWLWR facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Fugen ATR[1,4]	Fugen ATR[1]	Fugen ATR

1 Inventory filename set explicitly
4 Explicitly set active

Test: testLGRSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following LGR facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Hanford - N[1,4]	Toitsk E	Troitsk F[4]

1 Inventory filename set explicitly
4 Explicitly set active

Test: testLMFBRSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following LMFBR facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Beloyarsky-3(BN-600)[1]	Beloyarsky-4(BN-800)[1,3]	Beloyarsky-3(BN-600)

1 Inventory filename set explicitly
3 Explicitly set inactive

Test: testLWRSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following LWR facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Generic LWR Site	Generic LWR Site[3]	Generic LWR Site[2]

2 Inventory filename cleared explicitly
3 Explicitly set inactive

Test: testPHWRSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following PHWR facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Bruce-1	Gentilly-1	Agesta

Test: testPWRSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following PWR facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Doel-1	BR-3	Doel-2[2]

2 Inventory filename cleared explicitly

Test: testRBMKSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following RBMK facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Bilibino Unit A	Beloyarsky-1	Bilibino Unit A[2]

2 Inventory filename cleared explicitly

Test: testReproSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following Repro facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Idaho Falls-1	Mol	Idaho Falls-1[2]

2 Inventory filename cleared explicitly

Test: testResearchSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following Research facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
AFRRI TRIGA Reactor	BR-3 (Mol)	ATR

Test: testSGHWRSourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following SGHWR facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Winfrith SGHWR[1,4]	Winfrith SGHWR[1]	Winfrith SGHWR[4]

1 Inventory filename set explicitly
4 Explicitly set active

Test: testVVER0SourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following VVER type-0 facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Bushehr-1	Bushehr-1[3]	Bushehr-1[2]

2 Inventory filename cleared explicitly
3 Explicitly set inactive

Test: testVVER1SourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following VVER type-1 facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Metzamor-2	Metzamor-1	Kola-1[2]

2 Inventory filename cleared explicitly

Test: testVVER2SourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following VVER type-2 facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Dukovany-1	Nord (Greifswald)-5	Dukovany-2[2]

2 Inventory filename cleared explicitly

Test: testVVER3SourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following VVER type-3 facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	With inventory + Inactive	No inventory
Kozloduy-5	Kozloduy-6[3]	Kalinin-3[2]

2 Inventory filename cleared explicitly
3 Explicitly set inactive

4.6 SOURCETERMTABLETEST2

This test suite is distinguished from SourceTermTableTest for cases in which no plant condition models apply. For now, this is limited to the new HWRESRC research facility type.

Test: testHWRESRC3SourceTerms

Calls SourceTermTable.isOperationalSupported(), SourceTermTable.getFacilityTerms(), and PlantConditionTables.getPlantConditionTerms() with the following HWRESRC facilities and conditions, verifying the correct source term names are returned:

With inventory + Active	No inventory
IR-40	IR-40[2]

2 Inventory filename cleared explicitly

5. INTERACTIVE TESTS

The purpose of the interactive tests is to exercise each piece of functionality in the Nuclear Facility Incident Edit dialog and its constituent components. Verification of results is by visual inspection of the interface and/or examination of project files after being saved and are recorded in the test descriptions below.

5.1 TOP LEVEL INTERACTIONS

Test: Set Incident Name

Set the incident name via the *Name* edit field.
Verification:

- Incident name correctly (de)serialized from/to the project file.

5.2 WHERE TAB

Test: Select Facility

Select a facility via the *Select* button in the *Facility* group. Examine the facility tree to verify correct representation of the facilities defined in NFAC data files.
Verification:

- The *All Facilities* tree lists all the countries (and generic) represented in the *wcountry.dat* file.
- Each country tree node expands to the proper subtrees (Power Reactor, Reprocessing Facility, and/or Research Reactor) based on facilities available in that country.
- Verify special statuses *decommissioned*, *not yet completed*, and *no inventory* are accurately represented.
- Each leaf node lists the available facilities correctly as per the associated *lwr_??.dat* file.
- The selected facility is correctly represented and (de)serialized from/to the project file.

Test: Select Default Facility Location

Choose the *Facility Location* radio button in the *Location* group for the default facility location.
Verification:

- The coordinate for the facility as stored in the facility database is (de)serialized from/to the project file.

Test: Enter Explicit Facility Location

Selected the *Customized Location* radio button and enter an explicit coordinate for the facility location.
Verification:

- The entered coordinate is (de)serialized from/to the project file as the facility location.

Test: Define Facility Location by Dragging the Incident Icon

With the *Customized Location* radio button selected, drag the incident icon on the map to set the location.
Verification:

- The dragged-to coordinate is (de)serialized from/to the project file as the facility location.

Test: Select the Default Inventory File

Choose the *Default for Facility* radio button in the *Inventory* group.
Verification:

- The inventory file referenced in the facility database entry is (de)serialized from/to the project file.

Test: Select a Custom Inventory File

Choose the *Customized* radio button in the *Inventory* group and specify an inventory (i.e., activity vector or *.avc*) file.
Verification:

- The specified file is (de)serialized from/to the project file as the custom inventory.

Test: Use the ORIGEN File Import Utility to Create a Custom Inventory File

Activate the *Origen Import* button to run the ORIGEN Importer.
Verification:

- After selecting a F71 file, the *ORIGEN File Import* utility appears and displays the cases in the file.
- Activating the *Show File Overview* button brings up an overview windowing showing information about each case in the F71 file.
- After selecting a case and activating the *Show Case Overview* button, a window giving a summary of the selected case appears.

- After selecting a case and activating the *Show Case Details* button, a windowing displaying details of the selected case appears.
- After selecting a case, selecting and entering values in the *Inventory File Units* group, and activating the *Create Inventory File* button, and entering a target filename, an inventory file is created.
- The created inventory file can be specified as the custom inventory for the facility.

5.3 WHAT TAB

Test: Enter an Explicit Operating Power

Change the value of the *Operating Power* field.
Verification:

- The specified operating power is (de)serialized from/to the project file.

Test: Select a Moderate Incident

Choose the *Moderate Incident* radio button.
Verification:

- The Moderate model appropriate for the selected facility is (de)serialized from/to the project file and displayed in the report resulting from activating the *View Source Term* button (compared to the associated *acmxxx.dat* file).

Test: Select a Severe Incident

Choose the *Severe Incident* radio button.
Verification:

- The Severe model appropriate for the selected facility is (de)serialized from/to the project file and displayed in the report resulting from activating the *View Source Term* button (compared to the associated *acs*xxx.dat file).

Test: Define an Isotopic Release Rates Incident

Choose the *Technical Analysis* radio button. Activate the *Define Incident* button and choose *Isotopic Release Rates* from the *Source Terms Available* list. Click the *Next* button. From the *Define Isotopic Release Rates* dialog, choose *Release Units* and enter values under *Isotope Values*.
Verification:

- The defined isotope values and release units are (de)serialized from/to the project file and displayed in the report resulting from activating the *View Source Term* button.

Test: Define an Isotopic Concentrations Incident

Choose the *Technical Analysis* radio button. Activate the *Define Incident* button and choose *Isotopic Concentrations* from the *Source Terms Available* list. Click the *Next* button. From the *Define Isotopic Release Rates* dialog, choose *Release Units* and enter values under *Isotope Values*.
Verification:

- The defined isotope values and release units are (de)serialized from/to the project file and displayed in the report resulting from activating the *View Source Term* button.

Test: Define an Percent Inventory Incident

Choose the *Technical Analysis* radio button. Activate the *Define Incident* button and choose *Percent Inventory* from the *Source Terms Available* list. Click the *Next* button. From the *Define Percent Inventory Releases* dialog, activate the *Load* button and choose *From Moderate Model*. Then, activate the *Load* button and choose *From Severe Model*. Enter explicit values in the release duration and group percentage cells. Activate the *New* button to add a release. Change the duration units via the *Duration Units* combo box. Select a cell in any release column and activate *Delete* to remove that release.

Verification:

- The release definitions for the moderate and severe models are correct for the type of facility *acmxxx.dat* and *acsxxx.dat* files, respectively).
- The (first) totals column is accurately updated after cell edits.
- All duration cell values are converted upon selection of new duration units.
- A new, release column is added when *New* is activated with cell values set to the remainder required to reach 100% release of each group.
- *Delete*d releases are moved.
- The defined releases are (de)serialized from/to the project file and displayed in the report resulting from activating the *View Source Term* button.

Test: Define an Analyst Mix Incident

Choose the *Technical Analysis* radio button. Activate the *Define Incident* button and choose *Mix Specified by Analyst* from the *Source Terms Available* list. Click the *Next* button. From the *Define Mix Specified by Analyst* dialog, enter a *Gross Release Rate* and various values for the percentages by MELCOR group.

Verification:

- The *Gross Release Rate* value is (de)serialized from/to the project file.
- Entering release percentages totaling more than 100% results in a warning dialog.
- Release percentages for each group are (de)serialized from/to the project file.

Test: Define a Containment Monitor Reading Incident

Choose the *Technical Analysis* radio button. Activate the *Define Incident* button and choose *Mix Specified by Analyst* from the *Source Terms Available* list. Click the *Next* button. From the *Define Containment Monitor Reading Incident* dialog, enter various values for: *Representative Operating Power*, *Monitor Reading and Location*, *Sprays*, *Release Path*, and *Leak Rate*.

Verification:

- All entered values are (de)serialized from/to the project file.
- A warning appears if 100% leak rate is selected with a filtered release path.
- The release duration is correctly calculated from the specified leak rate.

Test: Define a Spent Fuel Incident

Choose the *Technical Analysis* radio button. Activate the *Define Incident* button and choose *Spent Fuel* from the *Source Terms Available* list. Click the *Next* button. From the *Define Spent Fuel/Spent Fuel Pool Incident* dialog, enter various values for: *Representative Operating Power*, *Number of Batches*, *Fuel Condition*, *Sprays*, *Release Path*, *Last Batch in Pool* datetime, and *Leak Rate*.

Verification:

- All entered values are (de)serialized from/to the project file.
- A warning appears if 100% leak rate is selected with a filtered release path.
- The *Last Batch in Pool* time is forced to be no later than *Shutdown/Start of Decay* on the *When* tab.
- The release duration is correctly calculated from the specified leak rate.

Test: Define a RASCAL Project Incident

Choose the *Technical Analysis* radio button. Activate the *Define Incident* button and choose *RASCAL Project* from the *Source Terms Available* list. Click the *Next* button. From the *RASCAL Project* dialog, activate the *Read File* button to locate and select a RASCAL project file.

Verification:

- The contained *tadspecs.tmp* file is read correctly to get the case name.
- The contained *NucName.tmp* file is read correctly to get the list of isotopes.
- The contained *STC_Rel.tmp* file is read correctly to get release durations and activity values for each isotope at each release time.
- The isotope activities and durations for each release are correctly (de)serialized from/to the project file.

Test: Define a Confinement Bypass Plant Conditions Incident

In the *Where* tab, choose a facility for which the Confinement Bypass plant condition is available, such as Hartlepool-1 in Great Britain (an AGR). On the *What* tab, choose the *Technical Analysis* radio button. Activate the *Define Incident* button, choose *Plant Conditions* from the *Source Terms Available* list, activate the *Next* button, and then choose *Bypass of Confinement* from the *Plant Conditions Available* list. From the *Bypass of Confinement* dialog, enter or select various values for *Representative Operating Power*, *Core Condition*, and *Leak Rate*.

Verification:

- Values entered or selected are (de)serialized from/to the project file.
- The release duration is correctly calculated from the specified leak rate.

Test: Define a Confinement Leakage/Failure Plant Conditions Incident

In the *Where* tab, choose a facility for which the Confinement Bypass plant condition is available, such as Novovoronezh-3 in Russia (a VVER-440/230). On the *What* tab, choose the *Technical Analysis* radio button. Activate the *Define Incident* button, choose *Plant Conditions* from the *Source Terms Available* list, activate the *Next* button, and then choose *Confinement*

Leakage/Failure from the *Plant Conditions Available* list. From the *Confinement Leakage/Failure* dialog, enter or select various values for *Representative Operating Power*, *Core Condition*, and *Leak Rate*.

Verification:

- Values entered or selected are (de)serialized from/to the project file.
- The release duration is correctly calculated from the specified leak rate.

Test: Define a Containment Bypass Plant Conditions Incident

In the *Where* tab, choose a facility for which the Containment Bypass plant condition is available, such as Hamaoka-3 in Japan (a BWR Mk-1). On the *What* tab, choose the *Technical Analysis* radio button. Activate the *Define Incident* button, choose *Plant Conditions* from the *Source Terms Available* list, activate the *Next* button, and then choose *Bypass of Containment* from the *Plant Conditions Available* list. From the *Bypass of Containment* dialog, enter or select various values for *Representative Operating Power*, *Core Condition*, *Release Path*, and *Leak Rate*.

Verification:

- Values entered or selected are (de)serialized from/to the project file.
- The release duration is correctly calculated from the specified leak rate.

Test: Define a Containment Leakage/Failure Plant Conditions Incident

In the *Where* tab, choose a facility for which the Containment Leakage plant condition is available, such as Beloyarsky-3(BN-600) in Russia (an LMFBR, requires a custom inventory file). On the *What* tab, choose the *Technical Analysis* radio button. Activate the *Define Incident* button, choose *Plant Conditions* from the *Source Terms Available* list, activate the *Next* button, and then choose *Containment Leakage/Failure* from the *Plant Conditions Available* list. From the *Containment Leakage/Failure* dialog, enter or select various values for *Representative Operating Power*, *Core Condition*, *Release Path*, and *Leak Rate*.

Verification:

- Values entered or selected are (de)serialized from/to the project file.
- The release duration is correctly calculated from the specified leak rate.

Test: Define a Dry Well Leakage/Failure Plant Conditions Incident

In the *Where* tab, choose a facility for which the Dry Well Leakage plant condition is available, such as Hamaoka-3 in Japan (a BWR Mk-1). On the *What* tab, choose the *Technical Analysis* radio button. Activate the *Define Incident* button, choose *Plant Conditions* from the *Source Terms Available* list, activate the *Next* button, and then choose *Dry Well Leakage/Failure* from the *Plant Conditions Available* list. From the *Dry Well Leakage/Failure* dialog, enter or select various values for *Representative Operating Power*, *Core Condition*, *Release Path*, *Sprays*, and *Leak Rate*.

Verification:

- Values entered or selected are (de)serialized from/to the project file.
- The release duration is correctly calculated from the specified leak rate.

Test: Define an Ice Condenser Containment Leakage/Failure Plant Conditions Incident

In the *Where* tab, choose a facility for which the Ice Condenser Containment plant condition is available, such as Watts Bar-1 in the United States (a PWR). On the *What* tab, choose the *Technical Analysis* radio button. Activate the *Define Incident* button, choose *Plant Conditions* from the *Source Terms Available* list, activate the *Next* button, and then choose *Ice Condenser Containment Leakage/Failure* from the *Plant Conditions Available* list. From the *Ice Condenser Containment Leakage/Failure* dialog, enter or select various values for *Representative Operating Power*, *Core Condition*, *Release Path*, *Sprays*, *Fans*, *Ice Bed Condition Before Core Damage*, and *Leak Rate*.

Verification:

- Values entered or selected are (de)serialized from/to the project file.
- The release duration is correctly calculated from the specified leak rate.

Test: Define an Large Dry Subcontainment Leakage/Failure Plant Conditions Incident

In the *Where* tab, choose a facility for which the Large Dry Subcontainment Leakage plant condition is available, such as Beaver Valley-1 in the United States (a PWR). On the *What* tab, choose the *Technical Analysis* radio button. Activate the *Define Incident* button, choose *Plant Conditions* from the *Source Terms Available* list, activate the *Next* button, and then choose *Large, Dry, or Subatmospheric Containment Leakage/Failure* from the *Plant Conditions Available* list. From the *Large, Dry, or Subatmospheric Containment Leakage/Failure* dialog, enter or select various values for *Representative Operating Power*, *Core Condition*, *Release Path*, *Sprays*, and *Leak Rate*.

Verification:

- Values entered or selected are (de)serialized from/to the project file.
- The release duration is correctly calculated from the specified leak rate.

Test: Define an Prompt Critical Power Excursion Plant Conditions Incident

In the *Where* tab, choose a facility for which the Prompt Critical Power Excursion plant condition is available, such as Bilibino Unit A in Russia (an RBMK). On the *What* tab, choose the *Technical Analysis* radio button. Activate the *Define Incident* button, choose *Plant Conditions* from the *Source Terms Available* list, activate the *Next* button, and then choose *Prompt Critical Power Excursion* from the *Plant Conditions Available* list. From the *Prompt Critical Power Excursion* dialog, enter or select various values for *Representative Operating Power*, and *Core Involvement*.

Verification:

- Values entered or selected are (de)serialized from/to the project file.

Test: Define an Reprocessing Facility Plant Conditions Incident

In the *Where* tab, choose a facility for which the Reprocessing Facility plant condition is available, such as Savannah River in the United States. On the *What* tab, choose the *Technical Analysis* radio button. Activate the *Define Incident* button, choose *Plant Conditions* from the *Source Terms Available* list, activate the *Next* button, and then choose *Reprocessing Facility Plant Conditions* from the *Plant Conditions Available* list. From the *Reprocessing Facility Plant Conditions* dialog, enter or select various values for *Plant Throughput*, *Release Rate*, and the values in the *Release Fraction by Component* group.

Representative Operating Power, and *Core Involvement*.
Verification:

- Values entered or selected are (de)serialized from/to the project file.

Test: Define an Steam Generator Tube Rupture Plant Conditions Incident

In the *Where* tab, choose a facility for which the Steam Tube Rupture plant condition is available, such as Dungeness B-1 in Great Britain (an AGR). On the *What* tab, choose the *Technical Analysis* radio button. Activate the *Define Incident* button, choose *Plant Conditions* from the *Source Terms Available* list, activate the *Next* button, and then choose *Steam Generator Tube Rupture (Coolant)* from the *Plant Conditions Available* list. From the *Steam Generator Rube Rupture (Coolant)* dialog, enter or select various values for *Coolant Concentration*, *Steam Generator Conditions*, *Release is From*, and *Release Rate*.
Verification:

- Values entered or selected are (de)serialized from/to the project file.

Test: Define an Subatmospheric Confinement Leakage/Failure Plant Conditions Incident

In the *Where* tab, choose a facility for which the Subatmospheric Confinement Leakage/Failure plant condition is available, such as Rovno-1 in the Ukraine (VVER-440/213). On the *What* tab, choose the *Technical Analysis* radio button. Activate the *Define Incident* button, choose *Plant Conditions* from the *Source Terms Available* list, activate the *Next* button, and then choose *Subatmospheric Confinement Leakage/Failure* from the *Plant Conditions Available* list. From the *Subatmospheric Confinement Leakage/Failure* dialog, enter or select various values for *Representative Operating Power*, *Core Condition*, *Pool Suppression System*, and *Leak Rate*.
Verification:

- Values entered or selected are (de)serialized from/to the project file.
- The release duration is correctly calculated from the specified leak rate.

Test: Define a Wet Well Leakage/Failure Plant Conditions Incident

In the *Where* tab, choose a facility for which the Wet Well Leakage plant condition is available, such as Hamaoka-3 in Japan (a BWR Mk-1). On the *What* tab, choose the *Technical Analysis* radio button. Activate the *Define Incident* button, choose *Plant Conditions* from the *Source Terms Available* list, activate the *Next* button, and then choose *Wet Well Leakage/Failure* from the *Plant Conditions Available* list. From the *Wet Well Leakage/Failure* dialog, enter or select various values for *Representative Operating Power*, *Core Condition*, *Release Path*, *Wet Well*, and *Leak Rate*.
Verification:

- Values entered or selected are (de)serialized from/to the project file.
- The release duration is correctly calculated from the specified leak rate.

Test: View Source Term

Activate the *View Source Term* button to generate and display an HTML report giving details of the source term as currently defined.

Verification:

- Source term HTML report generated for all source models.

5.4 WHEN TAB

The *When* tab in the NFAC GUI is relatively complex compared to other incident source models. Refer to Section 3.4 ModelTimesTest for a discription of the six event times possible in NFAC incidents. Which of the event times are visible and/or editable for a particular NFAC source model depends on that model and which of the event times are defined or computed based on the model definition. Further, the new *Advanced Mode* toggle button allows the user to override these settings and enter explicit times for all events, possibly redefining the model in the process. The interactive tests examine each NFAC source model to verify the proper times are visible and editable and are correctly computed if appropriate. The *Reset Times* button is activated to ensure the default times are applied.

Test: Moderate Model Times Edit

Verification:

Shutdown/Start of Decay:	visible, editable
Release to Containment:	invisible
Release to Environment:	visible, uneditable, calculated correctly
End of Release:	visible, uneditable, calculated correctly
End of Dispersion:	visible, editable
End of Exposure:	visible, editable

After reset, times set correctly

Test: Severe Model Times Edit

Verification:

Shutdown/Start of Decay:	visible, editable
Release to Containment:	invisible
Release to Environment:	visible, uneditable, calculated correctly
End of Release:	visible, uneditable, calculated correctly
End of Dispersion:	visible, editable
End of Exposure:	visible, editable

After reset, times set correctly

Test: Analyst Mix Model Times Edit

Verification:

Shutdown/Start of Decay: visible, editable
Release to Containment: invisible
Release to Environment: visible, editable
End of Release: visible, editable
End of Dispersion: visible, editable
End of Exposure: visible, editable
After reset, times set correctly

Test: Containment Monitor Reading Model Times Edit

Verification:

Shutdown/Start of Decay: invisible
Release to Containment: visible, editable
Release to Environment: visible, editable
End of Release: visible, editable[1]
End of Dispersion: visible, editable
End of Exposure: visible, editable
After reset, times set correctly

1 Time can be manually set less than computed end time based on leak rate

Test: Isotopic Concentrations Model Times Edit

Verification:

Shutdown/Start of Decay: invisible
Release to Containment: invisible
Release to Environment: visible, editable
End of Release: visible, editable
End of Dispersion: visible, editable
End of Exposure: visible, editable
After reset, times set correctly

Test: Isotopic Release Rates Model Times Edit

Verification:

Shutdown/Start of Decay: invisible
Release to Containment: invisible
Release to Environment: visible, editable
End of Release: visible, editable
End of Dispersion: visible, editable
End of Exposure: visible, editable
After reset, times set correctly

Test: Percent Inventory Model Times Edit

Verification:

Shutdown/Start of Decay: visible, editable
Release to Containment: invisible
Release to Environment: visible, uneditable, calculated correctly
End of Release: visible, uneditable, calculated correctly
End of Dispersion: visible, editable
End of Exposure: visible, editable
After reset, times set correctly

Test: Containment Bypass Plant Conditions Model Times Edit

Verification:

Shutdown/Start of Decay: visible, editable
Release to Containment: invisible
Release to Environment: visible, editable
End of Release: visible, editable[1]
End of Dispersion: visible, editable
End of Exposure: visible, editable
After reset, times set correctly

1 Time can be manually set less than computed end time based on leak rate

Test: Prompt Critical Power Excursion Plant Conditions Model Times Edit

Verification:

Shutdown/Start of Decay: invisible
Release to Containment: invisible
Release to Environment: visible, editable
End of Release: visible, editable
End of Dispersion: visible, editable
End of Exposure: visible, editable
After reset, times set correctly

Test: Reprocessing Facility Plant Conditions Model Times Edit

Verification:

Shutdown/Start of Decay: invisible
Release to Containment: invisible
Release to Environment: visible, editable
End of Release: visible, editable
End of Dispersion: visible, editable
End of Exposure: visible, editable
After reset, times set correctly

Test: Confinement Bypass Plant Conditions Model Times Edit

Times management for this source model is shared by all other Plant Conditions models except for the three identified in the preceeding tests.

Verification:

Shutdown/Start of Decay:	visible, editable
Release to Containment:	invisible
Release to Environment:	visible, editable
End of Release:	visible, editable[1]
End of Dispersion:	visible, editable
End of Exposure:	visible, editable

After reset, times set correctly

1 Time can be manually set less than computed end time based on leak rate

Test: Spent Fuel Model Times Edit

Verification:

Shutdown/Start of Decay:	invisible
Release to Containment:	visible, editable
Release to Environment:	visible, editable
End of Release:	visible, editable[1]
End of Dispersion:	visible, editable
End of Exposure:	visible, editable

After reset, times set correctly

1 Time can be manually set less than computed end time based on leak rate

Test: RASCAL Project Model Times Edit

Verification:

Shutdown/Start of Decay:	invisible
Release to Containment:	invisible
Release to Environment:	visible, editable
End of Release:	visible, uneditable, calculated correctly
End of Dispersion:	visible, editable
End of Exposure:	visible, editable

After reset, times set correctly

Test: Detect Event Times Out of Chronological Order

After modification of editable time fields and activation of the *OK* button for the *Nuclear Facility Incident Edit* dialog, if times are not in chronological order, a "Times are not in chronological order" error dialog is displayed, and the model definition remains unchanged.

Verification:

- Out of order times are detected, a warning dialog is displayed, and the model definition is not updated.

Test: Apply Time Field Changes for All Events

Enter a field value and activate the corresponding button in the *Change Field for All Events* group.

Verification:

- Month (*MM*) fields are updated.
- Day (*DD*) fields are updated.
- Year fields are updated.
- Hours (*hh*) fields are updated.
- Minutes (*mm*) fields are updated.
- Seconds (*ss*) fields are updated.
- Calculated fields are updated correctly.

Test: Override Times in Advanced Mode

Activate the *Advanced Mode* toggle button to override editability of fields.

Verification:

- All visible fields are editable.
- Chronological order checks are still applied.

5.5 NOTES TAB

Test: Edit Notes

Verification:

- Notes are (de)serialized from/to the project file.
- Notes can be modified (changed, deleted, augmented).

6. RESULTS

6.1 AUTOMATED TEST RESULTS

NfacJUnit4Adapter consolidates all test results into a single report. The report is listed below.

```
-------------------------------------------------------------------------
2015-12-02T17:03:35-05
Test suite: mil.dtra.hpac.models.nfac.test.junit.ExternalFileFinderTest
Tests run: 4, Failures: 0, Errors: 0, Time elapsed: 2.480 sec

Test case: testNfacImpl(mil.dtra.hpac.models.nfac.test.junit.ExternalFileFinderTest)
took 1.123 sec
Passed

Test case: testFileReleases(mil.dtra.hpac.models.nfac.test.junit.ExternalFileFinderTest)
took 0.874 sec
Passed

Test case: testProject(mil.dtra.hpac.models.nfac.test.junit.ExternalFileFinderTest)
took 0.265 sec
Passed
```

```
Test case: testAllIncidents(mil.dtra.hpac.models.nfac.test.junit.ExternalFileFinderTest)
took 0.218 sec
Passed

-------------------------------------------------------------------------------
2015-12-02T17:03:37-05
Test suite: mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest
Tests run: 42, Failures: 0, Errors: 0, Time elapsed: 13589.501 sec

Test case: testPercentInventoryNoShutdownAsIs(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 301.330 sec
Passed

Test case: testPCIceCondenserContainmentCoreDamage(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 51.646 sec
Passed

Test case: testPIRSubReleasesAverage(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 125.219 sec
Passed

Test case: testPCContainmentLeakage(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 29.626 sec
Passed

Test case: testPCDryWellLeakage(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 11.650 sec
Passed

Test case: testPCLargeDrySubContainmentLeakage(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 26.088 sec
Passed

Test case: test10682GroupsNone(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 631.498 sec
Passed

Test case: testAnalystMix(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 15.569 sec
Passed

Test case: testPCSteamTubeRuptureGapRelease(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 18.596 sec
Passed

Test case: testPercentInventoryWithShutdownAsIs(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 311.171 sec
Passed

Test case: testPCIceCondenserContainmentGapRelease(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 43.976 sec
Passed

Test case: test11653(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 15.669 sec
Passed

Test case: test10682GroupsExplicit(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 4524.318 sec
Passed

Test case: testPercentInventoryNoShutdownForced(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 298.039 sec
Passed
```

```
Test case: testSpentFuel(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 15.746 sec
Passed

Test case: testMultiFive(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 166.247 sec
Passed

Test case: testPCSubatmosphericConfinementLeakage(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 23.412 sec
Passed

Test case: testContainmentMonitorReading(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 52.711 sec
Passed

Test case: test10682SimpleNone(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 77.965 sec
Passed

Test case: testPCContainmentByPass(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 12.168 sec
Passed

Test case: testPCPromptCriticalPowerExcursion(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 22.519 sec
Passed

Test case: testPCSodiumWaterReaction(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 21.731 sec
Passed

Test case: testPCConfinementByPass(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 44.606 sec
Passed

Test case: testModerate(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 109.682 sec
Passed

Test case: test10682SimpleExplicit(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 375.591 sec
Passed

Test case: testIsotopicConcentrations(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 9.498 sec
Passed

Test case: test10682UnsupportedGroupsNone(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 58.758 sec
Passed

Test case: testPCWetWellLeakage(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 11.558 sec
Passed

Test case: testPCConfinementLeakage(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 23.711 sec
Passed

Test case: test10682RascalGroupsNone(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 680.833 sec
Passed

Test case: testPIRSubReleasesNone(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
```

```
took 108.681 sec
Passed

Test case: testPCReproFacility(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 11.597 sec
Passed

Test case: testRascalProject(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 336.176 sec
Passed

Test case: test10682VverGroupsNone(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 1263.811 sec
Passed

Test case: testPercentInventoryWithShutdownForced(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 313.360 sec
Passed

Test case: testPCIceCondenserContainmentMeltThrough(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 45.884 sec
Passed

Test case: testIsotopicReleaseRates(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 9.462 sec
Passed

Test case: testSevere(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 43.617 sec
Passed

Test case: testPCSteamTubeRuptureCoreDamage(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 20.033 sec
Passed

Test case: test10682RascalSimpleNone(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 83.051 sec
Passed

Test case: test10682PwrGroupsNone(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 242.527 sec
Passed

Test case: testPIRSubReleasesExplicit(mil.dtra.hpac.models.nfac.test.junit.ModelDefsTest)
took 3000.156 sec
Passed

-------------------------------------------------------------------------------
2015-12-02T20:49:52-05
Test suite: mil.dtra.hpac.models.nfac.test.junit.ModelTimesTest
Tests run: 8, Failures: 0, Errors: 0, Time elapsed: 0.453 sec

Test case: testAnalystMix(mil.dtra.hpac.models.nfac.test.junit.ModelTimesTest)
took 0.047 sec
Passed

Test case: testSpentFuel(mil.dtra.hpac.models.nfac.test.junit.ModelTimesTest)
took 0.031 sec
Passed

Test case: testPercentInventoryShutdown(mil.dtra.hpac.models.nfac.test.junit.ModelTimesTest)
took 0.047 sec
Passed

Test case: testPCContainmentByPass(mil.dtra.hpac.models.nfac.test.junit.ModelTimesTest)
```

```
took 0.031 sec
Passed

Test case: testPercentInventoryNoShutdown(mil.dtra.hpac.models.nfac.test.junit.ModelTimesTest)
took 0.032 sec
Passed

Test case: testIsotopicConcentrations(mil.dtra.hpac.models.nfac.test.junit.ModelTimesTest)
took 0.046 sec
Passed

Test case: testRascalProject(mil.dtra.hpac.models.nfac.test.junit.ModelTimesTest)
took 0.188 sec
Passed

Test case: testIsotopicReleaseRates(mil.dtra.hpac.models.nfac.test.junit.ModelTimesTest)
took 0.031 sec
Passed

-------------------------------------------------------------------------
2015-12-02T20:49:52-05
Test suite: mil.dtra.hpac.models.nfac.test.junit.ParticleGroupsMgrTest
Tests run: 7, Failures: 0, Errors: 0, Time elapsed: 0.015 sec

Test case: testReadCounts(mil.dtra.hpac.models.nfac.test.junit.ParticleGroupsMgrTest)
took 0.000 sec
Passed

Test case: testBWRCounts(mil.dtra.hpac.models.nfac.test.junit.ParticleGroupsMgrTest)
took 0.000 sec
Passed

Test case: testWriteAndRead(mil.dtra.hpac.models.nfac.test.junit.ParticleGroupsMgrTest)
took 0.015 sec
Passed

Test case: testTypeMap(mil.dtra.hpac.models.nfac.test.junit.ParticleGroupsMgrTest)
took 0.000 sec
Passed

Test case: testPWRCounts(mil.dtra.hpac.models.nfac.test.junit.ParticleGroupsMgrTest)
took 0.000 sec
Passed

Test case: testBWRValues(mil.dtra.hpac.models.nfac.test.junit.ParticleGroupsMgrTest)
took 0.000 sec
Passed

Test case: testPWRValues(mil.dtra.hpac.models.nfac.test.junit.ParticleGroupsMgrTest)
took 0.000 sec
Passed

-------------------------------------------------------------------------
2015-12-02T20:49:52-05
Test suite: mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest
Tests run: 25, Failures: 0, Errors: 0, Time elapsed: 0.000 sec

Test case: testHTGRSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testRBMKSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed
```

```
Test case: testPHWRSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testHWGCRSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testGCRSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testHWLWRSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testLWRSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testLGRSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testVVER0SourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testFBRSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testLMFBRSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testVVER1SourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testVVER2SourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testVVER3SourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testAGRCompatibility(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testAGRSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testBWR0SourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testResearchSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testBWR1SourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
```

```
took 0.000 sec
Passed

Test case: testBWR2SourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testReproSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testPWRSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testBWR3SourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testSGHWRSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

Test case: testGCHWRSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest)
took 0.000 sec
Passed

-------------------------------------------------------------------------
2015-12-02T20:49:52-05
Test suite: mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest2
Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.000 sec

Test case: testHWRESRCSourceTerms(mil.dtra.hpac.models.nfac.test.junit.SourceTermTableTest2)
took 0.000 sec
Passed
```

6.2 INTERACTIVE TEST RESULTS

Interactive tests results are as follows.

Test	**Result**
Set Incident Name	Success
Select default facility location	Success
Enter explicit facility location	Success
Define facility location by dragging the incident Icon	Success
Select the default inventory file	Success
Select a custom inventory file	Success
Use the ORIGEN File Import Utility to create a custom inventory file	Success
Enter an explicit operating power	Success
Select a moderate incident	Success
Select a severe incident	Success
Define an Isotopic Release Rates incident	Success
Define an Isotopic Concentrations incident	Success
Define a Percent Inventory incident	Success
Define an Analyst Mix incident	Success
Define a Containment Monitor Reading incident	Success
Define a RASCAL Project incident	Success
Define a Confinement Bypass Plant Conditions incident	Success
Define a Confinement Leakage/Failure Plant Conditions incident	Success
Define a Containment Bypass Plant Conditions incident	Success
Define a Containment Leakage/Failure Plant Conditions incident	Success
Define a Dry Well Leakage/Failure Plant Conditions incident	Success
Define an Ice Condenser Containment Leakage/Failure Plant Conditions incident	Success
Define a Large Dry Subcontainmet Leakage/Failure Plant Conditions incident	Success
Define a Prompt Critical Power Excursion Plant Condition incident	Success
Define a Reprocessing Facility Plant Condition incident	Success
Define a Steam Generator Tube Rupture Plant Condition incident	Success
Define a Subatmospheric Confinement Leakage/Failure Plant Conditions incident	Success
Define a Wet Well Leakage/Failure Plant Conditions incident	Success
View source term	Success
Moderate model times edit	Success
Severe model times edit	Success
Analyst Mix model times edit	Success
Containment Monitor Reading model times edit	Success
Isotopic Concentrations model times edit	Success
Isotopic Release Rates model times edit	Success
Percent Inventory model times edit	Success
Containment Bypass Plant Conditions model Times edit	Success
Prompt Critical Power Excursion Plant Conditions model times edit	Success
Reprocessing Facility Plant Conditions model times edit	Success
Confinement Bypass Plant Conditions model times edit	Success
RASCAL Project model times edit	Success
Spent Fuel model times edit	Success
Detect event times out of chronological order	Success
Apply time field changes for all events	Success
Override times in advanced mode	Success
Edit notes	Success

References

1. R.I. Sykes, C.P. Cerasoli, and D.S. Henn. The representation of dynamic flow effects in a lagrangian puff dispersion model. *J. Haz. Mat.*, 64:223–247, 1999.

2. R.I. Sykes and R.S. Gabruk. A second-order closure model for the effect of averaging time on turbulent plume dispersion. *J. Appl. Met.*, 36:165–184, 1997.

A. NFACJUNIT4ADAPTER SOURCE LISTING

```
 1 // $Id$
 2 //------------------------------------------------------------------------
 3 //      NAME:           NfacJUnit4Adapter.java                            -
 4 //      HISTORY:                                                          -
 5 //              2015-10-06      leerw@ornl.gov
 6 //        SPCR 14517.
 7 //              2015-07-15      leerw@ornl.gov
 8 //        Added timestamp to start of test suite in report.
 9 //              2015-03-04      leerw@ornl.gov
10 //        Added RunReporter and reporting to a file.
11 //              2014-12-30      leerw@ornl.gov
12 //        Calling Bundle.loadClass().
13 //              2014-12-29      leerw@ornl.gov
14 //        Saving BundleContext for shutting down after run.
15 //              2014-11-13      leerw@ornl.gov
16 //------------------------------------------------------------------------
17 package mil.dtra.hpac.models.nfac.test.junit;
18
19 import java.io.FileOutputStream;
20 import java.io.IOException;
21 import java.io.PrintStream;
22
23 import java.text.DateFormat;
24 import java.text.SimpleDateFormat;
25
26 import java.util.ArrayList;
27 import java.util.Arrays;
28 import java.util.Date;
29 import java.util.List;
30 import java.util.TreeSet;
31
32 import java.util.logging.Level;
33 import java.util.logging.Logger;
34
35 import org.junit.runner.Description;
36 import org.junit.runner.JUnitCore;
37 import org.junit.runner.Request;
38 import org.junit.runner.Result;
39 import org.junit.runner.Runner;
40
41 import org.junit.runner.manipulation.Filter;
42
43 import org.junit.runner.notification.Failure;
44 import org.junit.runner.notification.RunListener;
45
46 import org.osgi.framework.Bundle;
47 import org.osgi.framework.BundleContext;
48 //import org.osgi.framework.FrameworkUtil;
49
50
51 //------------------------------------------------------------------------
52 //      CLASS:          NfacJUnit4Adapter                                 -
53 /**************************************************************************
54 * Tests are defined in a comma-separated list of test class names with
55 * an optional colon-separated list of methods following each test class
56 * name.
57 **************************************************************************/
58 public class
59 NfacJUnit4Adapter
60     implements Runnable
61   {
62                 // Constants
```

```
63                //
64
65    public final static
66      String              BASE_NAME = "nfac.test.junit.NfacJUnit4Adapter";
67
68    public final static
69      String              PROP_report = BASE_NAME + ".report";
70
71    public final static
72      String              PROP_tests = BASE_NAME + ".tests";
73
74
75                // Class Attributes
76                //
77
78    protected final static
79      Logger              fLogger_ =
80          Logger.getLogger( NfacJUnit4Adapter.class.getPackage().getName() );
81
82
83                // Object Attributes
84                //
85
86    private
87      BundleContext       fBundleContext;
88
89    private
90      String              fReportFilename;
91
92    private
93      ArrayList<Runner>   fRunnerList;
94
95
96                // Object Methods
97                //
98
99
100   //------------------------------------------------------------------------
101   //    METHOD:         NfacJUnit4Adapter()                                -
102   /************************************************************************
103   ************************************************************************/
104   public
105   NfacJUnit4Adapter()
106     {
107     fLogger_.log( Level.INFO, "instantiated" );
108     fRunnerList = new ArrayList<>( 64 );
109     } // NfacJUnit4Adapter
110
111
112   //------------------------------------------------------------------------
113   //    METHOD:         activate()                                         -
114   /************************************************************************
115   * @return             object reference
116   ************************************************************************/
117   //@Activate
118   public void
119   activate( BundleContext context )
120     {
121     fLogger_.log( Level.INFO, "activated" );
122
123     setBundleContext( context );
124     new Thread( this ).start();
125
126     System.err.println( "[NfacJUnit4Adapter.activate] run thread launched" );
127     fLogger_.log( Level.INFO, "run thread launched" );
```

```
128     } // activate
129
130
131   //-------------------------------------------------------------------
132   //    METHOD:         deactivate()                                  -
133   /*******************************************************************
134   * @return            object reference
135   *******************************************************************/
136   //@Deactivate
137   public void
138   deactivate( BundleContext context )
139     {
140     fLogger_.log( Level.INFO, "deactivated" );
141     setBundleContext( null );
142     } // deactivate
143
144
145   //-------------------------------------------------------------------
146   //    METHOD:         getBundleContext()                            -
147   /*******************************************************************
148   * @return            object reference
149   *******************************************************************/
150   public BundleContext
151   getBundleContext()
152     {
153     return  fBundleContext;
154     } // getBundleContext
155
156
157   //-------------------------------------------------------------------
158   //    METHOD:         getReportFilename()                           -
159   /*******************************************************************
160   *******************************************************************/
161   public String
162   getReportFilename()
163     {
164     return  fReportFilename;
165     } // getReportFilename
166
167
168   //-------------------------------------------------------------------
169   //    METHOD:         getRunnerList()                               -
170   /*******************************************************************
171   * @return            object reference
172   *******************************************************************/
173   protected List<Runner>
174   getRunnerList()
175     {
176     return  fRunnerList;
177     } // getRunnerList
178
179
180   //-------------------------------------------------------------------
181   //    METHOD:         loadClass()                                   -
182   /*******************************************************************
183   *******************************************************************/
184   public Class<?>
185   loadClass( String class_name )
186       throws ClassNotFoundException
187     {
188     getLogger().info( "class_name=" + class_name );
189
190                 // Assert on Bundle
191                 //
192     Bundle  bundle;
```

```
193     if (
194         getBundleContext() == null ||
195         (bundle = getBundleContext().getBundle()) == null
196         )
197       throw  new IllegalStateException( "BundleContext has not been set" );
198
199     return  bundle.loadClass( class_name );
200     } // loadClass
201
202
203   //---------------------------------------------------------------------
204   //    METHOD:         run()                                           -
205   /**********************************************************************
206   **********************************************************************/
207   public void
208   run()
209     {
210                 // Get Output Filename
211                 //
212     setReportFilename( System.getProperty( PROP_report ) );
213
214                 // Get List of Tests/Methods to Run
215                 //
216     getRunnerList().clear();
217     String  defs_str = System.getProperty( PROP_tests, "" );
218
219     for ( String item : defs_str.split( "," ) )
220       {
221                       // Remove Trailing Colons, Then Split
222                       //
223       item = item.replaceAll( ":*$", "" );
224       String[]  tokens = item.split( ":" );
225
226                       // Class Name Cannot be Blank
227                       //
228       if ( tokens[ 0 ].length() > 0 )
229         {
230         try
231           {
232           getLogger().info( "loading class: " + tokens[ 0 ] );
233           Class<?>  test_class = loadClass( tokens[ 0 ] );
234           Request  request;
235
236                                 // Methods Specified?
237                                 //
238           if ( tokens.length > 1 )
239             {
240             String[]  methods = Arrays.copyOfRange( tokens, 1, tokens.length );
241             getLogger().info( "methods: " + Arrays.toString( methods ) );
242             Filter  filter = new MethodsFilter( test_class, methods );
243             request = Request.aClass( test_class ).filterWith( filter );
244             } // if methods specified
245
246           else
247             request = Request.aClass( test_class );
248
249           getLogger().info( "running test(s)" );
250           getRunnerList().add( request.getRunner() );
251           } // try
252
253         catch ( ClassNotFoundException ex )
254           {
255           getLogger().log(
256               Level.WARNING,
257               String.format( "Error loading class '%s'", tokens[ 0 ] ),
```

```
258              ex
259              );
260          }
261        } // if class specified
262      } // for each class item
263
264                // Anything to Run?
265                //
266    if ( getRunnerList().size() > 0 )
267      {
268      JUnitCore  junit = new JUnitCore();
269      RunReporter  reporter = new RunReporter( getReportFilename() );
270
271      junit.addListener( new RunReporter( getReportFilename() ) );
272      try
273        {
274        for ( Runner runner : getRunnerList() )
275          runOne( junit, runner );
276        }
277      finally
278        {
279        reporter.close();
280        }
281      } // if runners to run
282
283                // Quit
284                //
285    Bundle  sys_bundle;
286    if (
287        getBundleContext() != null &&
288        (sys_bundle = getBundleContext().getBundle( 0L )) != null
289        )
290      {
291      System.err.println( "[NfacJUnit4Adapter.run] stopping bundle 0" );
292      fLogger_.info( "stopping bundle 0" );
293
294      try { sys_bundle.stop(); }
295      catch ( Exception ex )
296        {
297        String  message = "Error stopping system bundle";
298        System.err.println( "[NfacJUnit4Adapter.run] " + message );
299        fLogger_.log( Level.WARNING, message, ex );
300        }
301      }
302    } // run
303
304
305  //-------------------------------------------------------------------------
306  //    METHOD:         runOne()                                           -
307  /*************************************************************************
308  *************************************************************************/
309  protected void
310  runOne( JUnitCore junit, Runner runner )
311    {
312    Result  result = junit.run( runner );
313    System.out.printf(
314        "%n%n%s  (%.3fs)%nTests: %3d  Failures: %3d  %s%n",
315        runner.getDescription().getClassName(),
316        result.getRunTime() / 1000.,
317        result.getRunCount(),
318        result.getFailureCount(),
319        result.wasSuccessful() ? "OK" : "Failed"
320        //result.getFailureCount() == 0 ? "OK" : "Failed"
321        );
322
```

```
323     for ( Failure failure : result.getFailures() )
324       System.out.printf( "%n%s%n", failure );
325 
326     System.out.println();
327     } // runOne
328 
329 
330   //------------------------------------------------------------------------
331   //    METHOD:         runOne_0()                                        -
332   /*************************************************************************
333   *************************************************************************/
334   protected void
335   runOne_0( JUnitCore junit, Runner runner )
336     {
337     Result  result = junit.run( runner );
338 
339     PrintStream  report = System.out;
340     if ( getReportFilename() != null )
341       {
342       try
343         {
344         report = new PrintStream(
345             new FileOutputStream( getReportFilename(), true ),
346             true
347             );
348         }
349       catch ( IOException ex )
350         {
351         report = System.out;
352         }
353       } // if report filename specified
354 
355     try
356       {
357       report.printf(
358           "%n%n%s  (%.3fs)%nTests: %3d  Failures: %3d  %s%n",
359           runner.getDescription().getClassName(),
360           result.getRunTime() / 1000.,
361           result.getRunCount(),
362           result.getFailureCount(),
363           result.wasSuccessful() ? "OK" : "Failed"
364           //result.getFailureCount() == 0 ? "OK" : "Failed"
365           );
366 
367       for ( Failure failure : result.getFailures() )
368         report.printf( "%n%s%n", failure );
369 
370       report.println();
371       } // try
372 
373     finally
374       {
375       if ( report != System.out )
376         report.close();
377       }
378     } // runOne_0
379 
380 
381   //------------------------------------------------------------------------
382   //    METHOD:         setBundleContext()                                -
383   /*************************************************************************
384   * @param  value       object reference to store
385   *************************************************************************/
386   protected void
387   setBundleContext( BundleContext value )
```

```
388     {
389     fBundleContext = value;
390     } // setBundleContext
391
392
393   //---------------------------------------------------------------------
394   //    METHOD:         setReportFilename()                            -
395   /***********************************************************************
396   ***********************************************************************/
397   protected void
398   setReportFilename( String value )
399     {
400     fReportFilename = value;
401     } // setReportFilename
402
403
404                 // Class Methods
405                 //
406
407
408   //---------------------------------------------------------------------
409   //    METHOD:         getLogger()                                    -
410   /***********************************************************************
411   * @return            object reference
412   ***********************************************************************/
413   public static Logger
414   getLogger()
415     {
416     return  fLogger_;
417     } // getLogger
418
419
420   //---------------------------------------------------------------------
421   //    METHOD:         loadClass_pojo()                               -
422   /***********************************************************************
423   ***********************************************************************/
424   public static Class<?>
425   loadClass_pojo( String class_name )
426       throws ClassNotFoundException
427     {
428     getLogger().info( "class_name=" + class_name );
429
430     try
431       {
432       getLogger().info( "trying class.classLoader" );
433       return  NfacJUnit4Adapter.class.getClassLoader().loadClass( class_name );
434       }
435
436     catch ( ClassNotFoundException ex )
437       {
438       try
439         {
440         getLogger().info( "trying thread.contextClassLoader" );
441         return
442         Thread.currentThread().getContextClassLoader().loadClass( class_name );
443         }
444
445       catch ( ClassNotFoundException ex2 )
446         {
447         getLogger().info( "trying Class.forName()" );
448         return  Class.forName( class_name );
449         }
450       }
451     } // loadClass_pojo
452
```

```
453
454   //-------------------------------------------------------------------
455   //    METHOD:         main()                                       -
456   /*******************************************************************
457   *******************************************************************/
458   public static void
459   main( String[] argv )
460     {
461     try
462       {
463       new NfacJUnit4Adapter().run();
464       }
465
466     catch ( Exception ex )
467       {
468       ex.printStackTrace( System.err );
469       }
470     } // main
471
472
473                 // Inner Classes
474                 //
475
476
477   //-------------------------------------------------------------------
478   //    CLASS:          MethodsFilter                                  -
479   /*******************************************************************
480   *******************************************************************/
481   protected final static class
482   MethodsFilter
483       extends Filter
484     {
485     private
486       TreeSet<String>   fMethodNames;
487
488     private
489       Class<?>          fTestClass;
490
491
492     //-------------------------------------------------------------------
493     //  METHOD:         MethodsFilter.MethodsFilter()                  -
494     /*******************************************************************
495     *******************************************************************/
496     public
497     MethodsFilter( Class<?> test_class, String[] method_names )
498       {
499       fTestClass = test_class;
500
501       fMethodNames = new TreeSet<String>();
502       if ( method_names != null && method_names.length > 0 )
503         {
504         for ( String name : method_names )
505           fMethodNames.add( name );
506         }
507       } // MethodsFilter
508
509
510     //-------------------------------------------------------------------
511     //  METHOD:         MethodsFilter.describe()                       -
512     /*******************************************************************
513     *******************************************************************/
514     public String
515     describe()
516       {
517       StringBuilder  buffer = new StringBuilder( 256 );
```

```
518       buffer.append( fTestClass.getName() );
519
520       if ( fMethodNames.size() > 0 )
521         {
522 /*Java8
523         buffer.
524             append( ":" ).
525             append( String.join( ":", fMethodNames ) );
526 */
527         buffer.append( ":" );
528         for ( String name : fMethodNames )
529           buffer.append( ":" ).append( name );
530         }
531
532       return  buffer.toString();
533       } // describe
534
535
536     //-------------------------------------------------------------------
537     //  METHOD:         MethodsFilter.shouldRun()                       -
538     /*******************************************************************
539     *******************************************************************/
540     public boolean
541     shouldRun( Description descr )
542       {
543       boolean  result = false;
544
545       if ( descr.isTest() )
546         {
547         result =
548             fTestClass.getName().equals( descr.getClassName() ) &&
549             descr.getMethodName() != null &&
550             fMethodNames.contains( descr.getMethodName() );
551         }
552
553       else
554         {
555         for ( Description child : descr.getChildren() )
556           {
557           if ( shouldRun( child ) )
558             {
559             result = true;
560             break;
561             }
562           }
563         } // else not a test
564
565       return  result;
566       } // shouldRun
567     } // MethodsFilter
568
569
570   //-------------------------------------------------------------------
571   //    CLASS:          RunReporter()                                   -
572   /*******************************************************************
573   *******************************************************************/
574   public static class
575   RunReporter
576       extends RunListener
577     {
578     private
579       StringBuilder     fBuffer;
580
581     private transient
582       SimpleDateFormat  fDateFormat;
```

```
583
584     private
585       List<Failure>     fFailures;
586
587     private
588       PrintStream       fPrint;
589
590     private
591       long              fTestStart;
592
593
594     //---------------------------------------------------------------------
595     //  METHOD:         RunReporter.RunReporter()                         -
596     /**********************************************************************
597     **********************************************************************/
598     public
599     RunReporter( String report_filename )
600       {
601       fBuffer = new StringBuilder( 32768 );
602       fDateFormat = new SimpleDateFormat( "yyyy-MM-dd'T'HH:mm:ssX" );
603       fFailures = new ArrayList<>( 128 );
604       fPrint = null;
605
606       if ( report_filename != null )
607         {
608         try
609           {
610           fPrint = new PrintStream(
611               new FileOutputStream( report_filename, true ),
612               true
613               );
614           }
615         catch ( IOException ex ) {}
616         } // if report filename specified
617
618       if ( fPrint == null )
619         fPrint = System.out;
620       } // RunReporter
621
622
623     //---------------------------------------------------------------------
624     //  METHOD:         RunReporter.close()                               -
625     /**********************************************************************
626     * @return           object reference
627     **********************************************************************/
628     public void
629     close()
630       {
631       fPrint.close();
632       } // close
633
634
635     //---------------------------------------------------------------------
636     //  METHOD:         RunReporter.finalize()                            -
637     /**********************************************************************
638     * @return           object reference
639     **********************************************************************/
640     @Override
641     protected void
642     finalize()
643       {
644       close();
645       } // finalize
646
647
```

```
648     //---------------------------------------------------------------------
649     //  METHOD:         RunReporter.getBuffer()                          -
650     /***********************************************************************
651     * @return           object reference
652     ***********************************************************************/
653     protected StringBuilder
654     getBuffer()
655       {
656       return  fBuffer;
657       } // getBuffer
658
659
660     //---------------------------------------------------------------------
661     //  METHOD:         RunReporter.getDateFormat()                      -
662     /***********************************************************************
663     * @return           object reference
664     ***********************************************************************/
665     protected DateFormat
666     getDateFormat()
667       {
668       return  fDateFormat;
669       } // getDateFormat
670
671
672     //---------------------------------------------------------------------
673     //  METHOD:         RunReporter.getFailures()                        -
674     /***********************************************************************
675     * @return           object reference
676     ***********************************************************************/
677     protected List<Failure>
678     getFailures()
679       {
680       return  fFailures;
681       } // getFailures
682
683
684     //---------------------------------------------------------------------
685     //  METHOD:         RunReporter.getPrint()                           -
686     /***********************************************************************
687     * @return           object reference
688     ***********************************************************************/
689     public PrintStream
690     getPrint()
691       {
692       return  fPrint;
693       } // getPrint
694
695
696     //---------------------------------------------------------------------
697     //  METHOD:         RunReporter.getTestStart()                       -
698     /***********************************************************************
699     ***********************************************************************/
700     public long
701     getTestStart()
702       {
703       return  fTestStart;
704       } // getTestStart
705
706
707     //---------------------------------------------------------------------
708     //  METHOD:         RunReporter.setTestStart()                       -
709     /***********************************************************************
710     ***********************************************************************/
711     protected void
712     setTestStart( long value )
```

```
713       {
714       fTestStart = value;
715       } // setTestStart
716
717
718     //---------------------------------------------------------------------
719     //  METHOD:         RunReporter.testAssumptionFailure()             -
720     /*********************************************************************
721     *********************************************************************/
722     @Override
723     public void
724     testAssumptionFailure( Failure failure )
725       {
726       testFailure( failure );
727       } // testAssumptionFailure
728
729
730     //---------------------------------------------------------------------
731     //  METHOD:         RunReporter.testFailure()                       -
732     /*********************************************************************
733     *********************************************************************/
734     @Override
735     public void
736     testFailure( Failure failure )
737       {
738       getFailures().add( failure );
739       //report.printf( "%n%s%n", failure );
740       } // testFailure
741
742
743     //---------------------------------------------------------------------
744     //  METHOD:         RunReporter.testFinished()                      -
745     /*********************************************************************
746     *********************************************************************/
747     @Override
748     public void
749     testFinished( Description descr )
750       {
751       long  now = System.currentTimeMillis();
752       getBuffer().append( String.format(
753           "%nTest case: %s took %.3f sec%n\t%s%n",
754           descr.getDisplayName(),
755           (now - getTestStart()) / 1000.,
756           getFailures().size() > 0 ? "FAILED" : "Passed"
757           ) );
758
759       for ( Failure failure : getFailures() )
760         getBuffer().append( String.format( "  %s%n", failure ) );
761       } // testFinished
762
763
764     //---------------------------------------------------------------------
765     //  METHOD:         RunReporter.testIgnored()                       -
766     /*********************************************************************
767     *********************************************************************/
768     @Override
769     public void
770     testIgnored( Description descr )
771       {
772       } // testIgnored
773
774
775     //---------------------------------------------------------------------
776     //  METHOD:         RunReporter.testRunFinished()                   -
777     /*********************************************************************
```

```
778     ************************************************************************/
779     @Override
780     public void
781     testRunFinished( Result result )
782       {
783       fPrint.printf(
784           "Tests run: %d, Failures: %d, Errors: %d, Time elapsed: %.3f sec%n",
785           result.getRunCount(),
786           result.getFailureCount(),
787           0, //result.getRunCount() - result.getFailureCount(),
788           result.getRunTime() / 1000.
789           );
790
791       fPrint.println( getBuffer().toString() );
792       } // testRunFinished
793
794
795     //------------------------------------------------------------------
796     //  METHOD:         RunReporter.testRunStarted()                    -
797     /************************************************************************
798     ************************************************************************/
799     @Override
800     public void
801     testRunStarted( Description descr )
802       {
803       getBuffer().setLength( 0 );
804       getFailures().clear();
805
806       fPrint.println( "-------------------------------------------------------------------------------" );
807       fPrint.printf(
808           "%s%nTest suite: %s%n",
809           getDateFormat().format( new Date() ),
810           descr.getDisplayName()
811           );
812       fPrint.flush();
813       } // testRunStarted
814
815
816     //------------------------------------------------------------------
817     //  METHOD:         RunReporter.testStarted()                       -
818     /************************************************************************
819     ************************************************************************/
820     @Override
821     public void
822     testStarted( Description descr )
823       {
824       getFailures().clear();
825       setTestStart( System.currentTimeMillis() );
826       //report.printf( "Starting %s%n", descr.getDisplayName() );
827       } // testStarted
828     } // RunReporter
829   } // NfacJUnit4Adapter
```

B. NFACLOG SOURCE LISTING

```
 1 // $Id$
 2 //--------------------------------------------------------------------------
 3 //      NAME:           NfacLog.java                                       -
 4 //      HISTORY:                                                           -
 5 //              2015-01-12      leerw@ornl.gov
 6 //        Eight chars in material name, matching decay time to ATIs.
 7 //              2014-03-03      leerw@ornl.gov
 8 //        New outputs.
 9 //              2014-02-19      leerw@ornl.gov
10 //--------------------------------------------------------------------------
11 package mil.dtra.hpac.models.nfac.test;
12
13 import java.io.BufferedReader;
14 import java.io.File;
15 import java.io.FileInputStream;
16 import java.io.IOException;
17 import java.io.InputStream;
18 import java.io.InputStreamReader;
19 import java.io.PrintStream;
20
21 import java.text.DecimalFormat;
22
23 import java.util.ArrayList;
24 import java.util.Arrays;
25 import java.util.Collections;
26
27 //import java.util.logging.Level;
28 //import java.util.logging.Logger;
29
30 import java.util.regex.Matcher;
31 import java.util.regex.Pattern;
32
33 import java.util.zip.GZIPInputStream;
34
35 import mil.dtra.hpac.models.nfac.CAcomp.data.ModelTimes;
36
37 import mil.dtra.hpac.util.DataUtils;
38 import mil.dtra.hpac.util.Time;
39
40
41 //--------------------------------------------------------------------------
42 //      CLASS:          NfacLog                                            -
43 /**************************************************************************
44 * Captures the NFAC log "FINE" level output showing inventories and
45 * activities.
46 **************************************************************************/
47 public class
48 NfacLog
49   {
50               // Constants
51               //
52
53   public final static
54     String            BASE_NAME = "nfac.test.NfacLog";
55
56   public final static
57     Pattern           PATTERN_ati =
58         Pattern.compile( "^activityTableInput: name=[^,]*, matID=(.*)$" );
59
60   public final static
61     Pattern           PATTERN_releaseBaseActivity =
62         Pattern.compile( "^\\Q++\\Erelease_base_activity, pir_ndx=([0-9]+):$" );
```

```
 63
 64   public final static
 65     Pattern             PATTERN_ws = Pattern.compile( "[\\s,]+" );
 66
 67   public final static
 68     String              PROP_debug = BASE_NAME + ".debug";
 69
 70
 71                 // Object Attributes
 72                 //
 73
 74   private
 75     ActivityInputRec[]  fActivityInputs;
 76
 77   private
 78     IsotopeRec[]        fFacilityInventory;
 79
 80   private
 81     ModelTimes          fModelTimes;
 82
 83   private
 84     ReleaseRec[]        fReleases;
 85
 86
 87                 // Object Methods
 88                 //
 89
 90
 91   //-----------------------------------------------------------------------
 92   //    METHOD:         NfacLog()                                         -
 93   /************************************************************************
 94   ************************************************************************/
 95   public
 96   NfacLog( File file )
 97       throws IOException
 98     {
 99     read( file );
100     } // NfacLog
101
102
103   //-----------------------------------------------------------------------
104   //    METHOD:         NfacLog()                                         -
105   /************************************************************************
106   ************************************************************************/
107   public
108   NfacLog( InputStream input )
109       throws IOException
110     {
111     read( input );
112     } // NfacLog
113
114
115   //-----------------------------------------------------------------------
116   //    METHOD:         equals()                                          -
117   /************************************************************************
118   ************************************************************************/
119   @Override
120   public boolean
121   equals( Object that )
122     {
123     return  !(that instanceof NfacLog) ? false : equals( (NfacLog)that );
124     } // equals
125
126
127   //-----------------------------------------------------------------------
```

```
128   //    METHOD:         equals()                                        -
129   /*********************************************************************
130   *********************************************************************/
131   public boolean
132   equals( NfacLog that )
133     {
134     boolean  equal = false;
135
136     if ( Boolean.getBoolean( PROP_debug ) )
137       {
138       boolean[]  eq = new boolean[ 4 ];
139
140       eq[ 0 ] = DataUtils.equals( this.getModelTimes(), that.getModelTimes() );
141       System.err.printf( "[NfacLog.equals] modelTimes=%b%n", eq[ 0 ] );
142
143       eq[ 1 ] = Arrays.
144           equals( this.getFacilityInventory(), that.getFacilityInventory() );
145       System.err.printf( "[NfacLog.equals] facilityInventory=%b%n", eq[ 1 ] );
146
147       eq[ 2 ] = Arrays.equals( this.getReleases(), that.getReleases() );
148       System.err.printf( "[NfacLog.equals] releases=%b%n", eq[ 2 ] );
149
150       eq[ 3 ] = Arrays.
151           equals( this.getActivityInputs(), that.getActivityInputs() );
152       System.err.printf( "[NfacLog.equals] activityInputs=%b%n", eq[ 3 ] );
153
154       equal = true;
155       for ( int i = 0; i < eq.length && equal; i++ )
156         equal &= eq[ i ];
157       }
158
159     else
160       {
161       equal =
162         DataUtils.equals( getModelTimes(), that.getModelTimes() ) &&
163         Arrays.equals( getFacilityInventory(), that.getFacilityInventory() ) &&
164         Arrays.equals( getReleases(), that.getReleases() ) &&
165         Arrays.equals( getActivityInputs(), that.getActivityInputs() );
166       }
167
168     return  equal;
169     } // equals
170
171
172   //-------------------------------------------------------------------
173   //    METHOD:         getActivityInputs()                             -
174   /*********************************************************************
175   * @return            array reference
176   *********************************************************************/
177   public ActivityInputRec[]
178   getActivityInputs()
179     {
180     return  fActivityInputs;
181     } // getActivityInputs
182
183
184   //-------------------------------------------------------------------
185   //    METHOD:         getFacilityInventory()                          -
186   /*********************************************************************
187   * @return            array reference
188   *********************************************************************/
189   public IsotopeRec[]
190   getFacilityInventory()
191     {
192     return  fFacilityInventory;
```

```
193     } // getFacilityInventory
194
195
196   //------------------------------------------------------------------------
197   //    METHOD:         getModelTimes()                                   -
198   /*************************************************************************
199   * @return            object reference
200   *************************************************************************/
201   public ModelTimes
202   getModelTimes()
203     {
204     return  fModelTimes;
205     } // getModelTimes
206
207
208   //------------------------------------------------------------------------
209   //    METHOD:         getReleases()                                     -
210   /*************************************************************************
211   * @return            array reference
212   *************************************************************************/
213   public ReleaseRec[]
214   getReleases()
215     {
216     return  fReleases;
217     } // getReleases
218
219
220   //------------------------------------------------------------------------
221   //    METHOD:         read()                                            -
222   /*************************************************************************
223   * Calls {@link #read( InputStream )}.
224   *************************************************************************/
225   public void
226   read( File file )
227       throws IOException
228     {
229     InputStream  input = new FileInputStream( file );
230
231     if ( file.getName().endsWith( ".gz" ) )
232       input = new GZIPInputStream( input );
233
234     try
235       { read( input ); }
236     finally
237       { input.close(); }
238     } // read
239
240
241   //------------------------------------------------------------------------
242   //    METHOD:         read()                                            -
243   /*************************************************************************
244   *************************************************************************/
245   public void
246   read( InputStream input )
247       throws IOException
248     {
249     BufferedReader  reader =
250         new BufferedReader( new InputStreamReader( input ) );
251     String  line, state = null;
252
253     IsotopeRec[]  facility_inv = null;
254     ArrayList<ReleaseRec>  release_recs = new ArrayList<>( 16 );
255     ActivityInputRec[]  ati_recs = null;
256
257                 // Outer Read Loop
```

```
258                 //
259     while ( (line = reader.readLine()) != null )
260       {
261                         // Model Times
262                         //
263       if ( line.startsWith( "[ModelTimes.begin]:" ) )
264         setModelTimes( readModelTimes( reader ) );
265
266                         // Facility Inventory
267                         //
268       else if ( line.startsWith( "+facility_inventory:" ) )
269         {
270         facility_inv = IsotopeRec.readArray( reader );
271
272         float[]  decay_times = readResolvedDecayTimes( reader );
273
274         ReleaseRec  release_rec;
275         while ( (release_rec = ReleaseRec.readNext( reader )) != null )
276           release_recs.add( release_rec );
277         }
278
279                         // Facility Inventory
280                         //
281       else if ( line.startsWith( "FINE: +activityTableInputs:" ) )
282         ati_recs = ActivityInputRec.readArrayEnd( reader, null );
283       else if ( line.startsWith( "activityTableInput:" ) )
284         ati_recs = ActivityInputRec.readArrayEnd( reader, line );
285       } // while
286
287     setFacilityInventory( facility_inv );
288     setReleases( release_recs.toArray( new ReleaseRec[ release_recs.size() ] ) );
289     setActivityInputs( ati_recs );
290     } // read
291
292
293   //---------------------------------------------------------------------
294   //    METHOD:         readModelTimes()                                -
295   /**********************************************************************
296   **********************************************************************/
297   public ModelTimes
298   readModelTimes( BufferedReader reader )
299       throws IOException
300     {
301     String  line;
302     Time
303         start = new Time(),
304         rel_cont = new Time(),
305         rel_env = new Time(),
306         end_rel = new Time(),
307         end_disp = new Time(),
308         end_exp = new Time();
309
310     while (
311         (line = reader.readLine()) != null &&
312         ! line.startsWith( "[ModelTimes.end]" )
313         )
314       {
315       int  comma_ndx = line.indexOf( "," );
316       if ( comma_ndx > 0 )
317         {
318         line = line.substring( 0, comma_ndx ).trim();
319         if ( line.startsWith( "start=" ) )
320           start.valueOf( line.substring( 6 ) );
321         else if ( line.startsWith( "relCont=" ) )
322           rel_cont.valueOf( line.substring( 8 ) );
```

```
323         else if ( line.startsWith( "relEnv=" ) )
324           rel_env.valueOf( line.substring( 7 ) );
325         else if ( line.startsWith( "endRel=" ) )
326           end_rel.valueOf( line.substring( 7 ) );
327         else if ( line.startsWith( "endDisp=" ) )
328           end_disp.valueOf( line.substring( 8 ) );
329         else if ( line.startsWith( "endExp=" ) )
330           end_exp.valueOf( line.substring( 7 ) );
331         }
332       } // while
333
334     return
335         new ModelTimes( start, rel_cont, rel_env, end_rel, end_disp, end_exp );
336     } // readModelTimes
337
338
339   //------------------------------------------------------------------------
340   //    METHOD:         readResolvedDecayTimes()                          -
341   /*************************************************************************
342   *************************************************************************/
343   public float[]
344   readResolvedDecayTimes( BufferedReader reader )
345       throws IOException
346     {
347     float[]  times = null;
348     String  line;
349     int  eq_ndx;
350
351     while (
352         (line = reader.readLine()) != null &&
353         ! line.contains( "resolved decay_times=" )
354         )
355       ;
356
357     if ( line != null && (eq_ndx = line.indexOf( "=" )) >= 0 )
358       {
359       line = line.substring( eq_ndx + 2, line.length() - 1 );
360       String[]  tokens = PATTERN_ws.split( line );
361
362       times = new float[ tokens.length ];
363       for ( int i = 0; i < tokens.length; i++ )
364         times[ i ] = DataUtils.parseFloat( tokens[ i ] );
365       } // if
366
367     return  times == null ? new float[ 0 ] : times;
368     } // readResolvedDecayTimes
369
370
371   //------------------------------------------------------------------------
372   //    METHOD:         setActivityInputs()                               -
373   /*************************************************************************
374   * @param  value       array reference to store
375   *************************************************************************/
376   public void
377   setActivityInputs( ActivityInputRec[] value )
378     {
379     fActivityInputs = value;
380     } // setActivityInputs
381
382
383   //------------------------------------------------------------------------
384   //    METHOD:         setFacilityInventory()                            -
385   /*************************************************************************
386   * @param  value       array reference to store
387   *************************************************************************/
```

```
388   public void
389   setFacilityInventory( IsotopeRec[] value )
390     {
391     fFacilityInventory = value;
392     } // setFacilityInventory
393
394
395   //-----------------------------------------------------------------------
396   //    METHOD:         setModelTimes()                                   -
397   /*************************************************************************
398   * @param  value       object reference to store
399   *************************************************************************/
400   public void
401   setModelTimes( ModelTimes value )
402     {
403     fModelTimes = value;
404     } // setModelTimes
405
406
407   //-----------------------------------------------------------------------
408   //    METHOD:         setReleases()                                     -
409   /*************************************************************************
410   * @param  value       array reference to store
411   *************************************************************************/
412   public void
413   setReleases( ReleaseRec[] value )
414     {
415     fReleases = value;
416     } // setReleases
417
418
419   //-----------------------------------------------------------------------
420   //    METHOD:         write()                                           -
421   /*************************************************************************
422   *************************************************************************/
423   public void
424   write( PrintStream writer )
425       throws IOException
426     {
427     DecimalFormat  value_fmt = new DecimalFormat( "0.0##E00" );
428
429     if ( getFacilityInventory() != null )
430       {
431       writer.printf(
432           "+facility_inventory:%n%ss%n",
433           IsotopeRec.toHeaderString()
434           );
435       for ( IsotopeRec iso_rec : getFacilityInventory() )
436         writer.println( iso_rec.toPrettyString( value_fmt ) );
437       }
438
439     for ( ReleaseRec release_rec : getReleases() )
440       {
441       writer.printf( "%n%n" );
442       release_rec.write( writer );
443       }
444
445     for ( ActivityInputRec ati_rec : getActivityInputs() )
446       {
447       writer.printf( "%n%n" );
448       ati_rec.write( writer );
449       }
450     } // write
451
452
```

```
453                 // Class Methods
454                 //
455
456
457   //---------------------------------------------------------------------
458   //    METHOD:         main()                                         -
459   /*********************************************************************
460   *********************************************************************/
461   public static void
462   main( String[] argv )
463     {
464     try
465       {
466       if ( argv.length < 2 )
467         System.err.println( "Usage: NfacLog { -i file | file1 file2 }" );
468
469       else if ( "-i".equals( argv[ 0 ] ) )
470         new NfacLog( new File( argv[ 1 ] ) ).write( System.out );
471
472       else
473         {
474         NfacLog
475             one = new NfacLog( new File( argv[ 0 ] ) ),
476             two = new NfacLog( new File( argv[ 1 ] ) );
477
478         System.out.println( one.equals( two ) ? "EQUAL" : "not equal" );
479         }
480       } // try
481
482     catch ( Exception ex )
483       {
484       ex.printStackTrace( System.err );
485       }
486     } // main
487
488
489                 // Inner Classes
490                 //
491
492
493   //---------------------------------------------------------------------
494   //    CLASS:          ActivityInputRec                               -
495   /*********************************************************************
496   *********************************************************************/
497   public static class
498   ActivityInputRec
499       implements Comparable<ActivityInputRec>
500     {
501     public float        decaytime;
502
503     public IsotopeRec[] isotopes;
504
505     public String       material;
506
507
508     //-------------------------------------------------------------------
509     //  METHOD:         ActivityInputRec()                             -
510     /*******************************************************************
511     *******************************************************************/
512     public
513     ActivityInputRec( String material, IsotopeRec[] isotopes, float decay_time )
514       {
515       this.decaytime = decay_time;
516       this.isotopes = isotopes;
517       this.material = material;
```

```
518       } // ActivityInputRec
519
520
521     //-----------------------------------------------------------------
522     //  METHOD:         compareTo()                                   -
523     /*****************************************************************
524     *****************************************************************/
525     @Override
526     public int
527     compareTo( ActivityInputRec that )
528       {
529       String
530           this_name = RadFile.getMaterialIndexName( this.material ),
531           that_name = RadFile.getMaterialIndexName( that.material );
532
533       return
534           this_name == null ? (that_name == null ? 0 : -1) :
535           that_name == null ? 1 :
536           this_name.compareTo( that_name );
537       } // compareTo
538
539
540     //-----------------------------------------------------------------
541     //  METHOD:         equals()                                      -
542     /*****************************************************************
543     *****************************************************************/
544     @Override
545     public boolean
546     equals( Object that )
547       {
548       boolean  equal = false;
549
550       if ( that instanceof ActivityInputRec )
551         {
552         ActivityInputRec  rthat = (ActivityInputRec)that;
553
554         equal =
555             DataUtils.equals( this.material, rthat.material ) &&
556             Arrays.equals( this.isotopes, rthat.isotopes );
557         } // if a ReleaseRec
558
559       return  equal;
560       } // equals
561
562
563     //-----------------------------------------------------------------
564     //  METHOD:         write()                                       -
565     /*****************************************************************
566     *****************************************************************/
567     public void
568     write( PrintStream writer )
569         throws IOException
570       {
571       DecimalFormat  value_fmt = new DecimalFormat( "0.0##E00" );
572
573       writer.printf(
574           "+activity_table_input name=%s decay=%g:%n%s%n",
575           this.material, this.decaytime, IsotopeRec.toHeaderString()
576           );
577       for ( IsotopeRec iso_rec : this.isotopes )
578         writer.println( iso_rec.toPrettyString( value_fmt ) );
579       } // write
580
581
582     //-----------------------------------------------------------------
```

```
583     //  METHOD:         readArray()                                     -
584     /*******************************************************************
585     *******************************************************************/
586     public static ActivityInputRec[]
587     readArray( BufferedReader reader, float[] decay_times )
588         throws IOException
589       {
590       String  line;
591       ArrayList<ActivityInputRec>  rec_list = new ArrayList<>( 16 );
592
593       while (
594           (line = reader.readLine()) != null &&
595           ! line.contains( "+activityTableInputs:" )
596           )
597         ;
598
599       int  decay_ndx = 0;
600       while (
601           (line = reader.readLine()) != null &&
602           (line = line.trim()).length() > 0
603           )
604         {
605         int  ndx = line.lastIndexOf( "=" );
606         String  name = line.substring( ndx + 1 ).substring( 0, 10 );
607
608         line = reader.readLine();
609         IsotopeRec[]  isotopes = IsotopeRec.readArray( reader );
610
611         float  decay_time =
612             decay_ndx < decay_times.length ? decay_times[ decay_ndx ] : -1.f;
613         rec_list.add( new ActivityInputRec( name, isotopes, decay_time ) );
614
615         decay_ndx++;
616         } // while reading blocks
617
618       Collections.sort( rec_list );
619       return  rec_list.toArray( new ActivityInputRec[ rec_list.size() ] );
620       } // readArray
621
622
623     //-----------------------------------------------------------------
624     //  METHOD:         readArrayEnd()                                  -
625     /*******************************************************************
626     *******************************************************************/
627     public static ActivityInputRec[]
628     readArrayEnd( BufferedReader reader, String line )
629         throws IOException
630       {
631       //String  line;
632       ArrayList<ActivityInputRec>  rec_list = new ArrayList<>( 16 );
633
634       if ( line == null )
635         line = reader.readLine();
636
637       while (
638           //(line = reader.readLine()) != null &&
639           line != null &&
640           ! line.contains( "ModelDefUtils logIncidentObjects" )
641           )
642         {
643         Matcher  m = PATTERN_ati.matcher( line );
644         if ( m.matches() )
645           {
646           String  name = m.group( 1 ).substring( 0, 10 );
647           reader.readLine();
```

```
648
649           IsotopeRec[]  isotopes = IsotopeRec.readArray( reader );
650           rec_list.add( new ActivityInputRec( name, isotopes, -1.f ) );
651           }
652
653         line = reader.readLine();
654         } // outer while
655
656       Collections.sort( rec_list );
657       return  rec_list.toArray( new ActivityInputRec[ rec_list.size() ] );
658       } // readArrayEnd
659     } // ActivityInputRec
660
661
662   //---------------------------------------------------------------------
663   //    CLASS:          IsotopeRec                                      -
664   /*********************************************************************
665   *********************************************************************/
666   public static class
667   IsotopeRec
668       implements Comparable<IsotopeRec>
669     {
670     public double       activity;
671     public int          dep;
672     public int          group;
673     public String       name;
674
675
676     //---------------------------------------------------------------------
677     //  METHOD:         IsotopeRec()                                      -
678     /*********************************************************************
679     *********************************************************************/
680     public
681     IsotopeRec( String line )
682       {
683       valueOf( line );
684       } // IsotopeRec
685
686
687     //---------------------------------------------------------------------
688     //  METHOD:         IsotopeRec()                                      -
689     /*********************************************************************
690     *********************************************************************/
691     public
692     IsotopeRec( String name, int dep, double activity )
693       {
694       this( name, dep, -1, activity );
695       } // IsotopeRec
696
697
698     //---------------------------------------------------------------------
699     //  METHOD:         IsotopeRec()                                      -
700     /*********************************************************************
701     *********************************************************************/
702     public
703     IsotopeRec( String name, int dep, int group, double activity )
704       {
705       this.name = name;
706       this.dep = dep;
707       this.group = group;
708       this.activity = activity;
709       } // IsotopeRec
710
711
712     //---------------------------------------------------------------------
```

```
713     //  METHOD:         compareTo()                                     -
714     /*********************************************************************
715     *********************************************************************/
716     @Override
717     public int
718     compareTo( IsotopeRec that )
719       {
720       return  this.name.compareTo( that.name );
721       } // compareTo
722
723
724     //--------------------------------------------------------------------
725     //  METHOD:         equals()                                        -
726     /*********************************************************************
727     *********************************************************************/
728     @Override
729     public boolean
730     equals( Object that )
731       {
732       boolean  equal = false;
733
734       if ( that instanceof IsotopeRec )
735         {
736         IsotopeRec  rthat = (IsotopeRec)that;
737         equal =
738             DataUtils.equals( this.name, rthat.name ) &&
739             this.dep == rthat.dep &&
740             this.group == rthat.group &&
741             this.activity == rthat.activity;
742         }
743
744       return  equal;
745       } // equals
746
747
748     //--------------------------------------------------------------------
749     //  METHOD:         toPrettyString()                                -
750     /*********************************************************************
751     *********************************************************************/
752     public String
753     toPrettyString( DecimalFormat value_fmt )
754       {
755       return
756       String.format(
757           "%-8s %3d %9s  %02d",
758           this.name, this.dep, value_fmt.format( this.activity ), this.group
759           );
760       } // toPrettyString
761
762
763     //--------------------------------------------------------------------
764     //  METHOD:         toString()                                      -
765     /*********************************************************************
766     *********************************************************************/
767     @Override
768     public String
769     toString()
770       {
771       return
772       String.format(
773           "%s,%d,%-8g,%d",
774           this.name, this.dep, this.activity, this.group
775           );
776       } // toString
777
```

```
778
779     //---------------------------------------------------------------
780     //  METHOD:         valueOf()                                    -
781     /*****************************************************************
782     *****************************************************************/
783     public void
784     valueOf( String line )
785       {
786       String[]  tokens = line.split( "[\\s,]+" );
787
788       this.name = tokens.length > 0 ? tokens[ 0 ] : "";
789       this.dep = tokens.length > 1 ? DataUtils.parseInt( tokens[ 1 ], 0 ) : 0;
790       this.activity =
791           tokens.length > 2 ? DataUtils.parseDouble( tokens[ 2 ], 0.0 ) : 0.0;
792       this.group = tokens.length > 3 ?
793           DataUtils.parseInt( tokens[ 3 ], -1 ) : -1;
794       } // valueOf
795
796
797     //---------------------------------------------------------------
798     //  METHOD:         readArray()                                  -
799     /*****************************************************************
800     *****************************************************************/
801     public static IsotopeRec[]
802     readArray( BufferedReader reader )
803         throws IOException
804       {
805       ArrayList<IsotopeRec>  rec_list = new ArrayList<>( 1024 );
806       String  line;
807
808       while (
809           (line = reader.readLine()) != null &&
810           (line = line.trim()).length() > 0
811           )
812         {
813         if ( ! line.startsWith( "Isotope" ) )
814           rec_list.add( new IsotopeRec( line ) );
815         }
816
817       Collections.sort( rec_list );
818       return  rec_list.toArray( new IsotopeRec[ rec_list.size() ] );
819       } // readArray
820
821
822     //---------------------------------------------------------------
823     //  METHOD:         toHeaderString()                             -
824     /*****************************************************************
825     *****************************************************************/
826     public static String
827     toHeaderString()
828       {
829       return
830       String.format(
831           "%-8s %3s %9s %3s",
832           "Isotope", "Dep", "Activity", "Grp"
833           );
834       } // toHeaderString
835     } // IsotopeRec
836
837
838   //---------------------------------------------------------------
839   //    CLASS:          ReleaseRec                                 -
840   /*****************************************************************
841   *****************************************************************/
842   public static class
```

```
843   ReleaseRec
844     {
845     public IsotopeRec[] base;
846
847     public IsotopeRec[] depositors;
848
849     public IsotopeRec[] nondepositors;
850
851     public int          index = -1;
852
853
854     //---------------------------------------------------------------------
855     //  METHOD:         ReleaseRec()                                      -
856     /*********************************************************************
857     *********************************************************************/
858     public
859     ReleaseRec(
860         int             index,
861         IsotopeRec[]    base,
862         IsotopeRec[]    depositors,
863         IsotopeRec[]    nondepositors
864         )
865         throws IOException
866       {
867       this.index = index;
868       this.base = base;
869       this.depositors = depositors;
870       this.nondepositors = nondepositors;
871       } // ReleaseRec
872
873
874     //---------------------------------------------------------------------
875     //  METHOD:         equals()                                          -
876     /*********************************************************************
877     *********************************************************************/
878     @Override
879     public boolean
880     equals( Object that )
881       {
882       boolean  equal = false;
883
884       if ( that instanceof ReleaseRec )
885         {
886         ReleaseRec  rthat = (ReleaseRec)that;
887
888         equal =
889             this.index == rthat.index &&
890             Arrays.equals( this.base, rthat.base ) &&
891             Arrays.equals( this.depositors, rthat.depositors ) &&
892             Arrays.equals( this.nondepositors, rthat.nondepositors );
893         } // if a ReleaseRec
894
895       return  equal;
896       } // equals
897
898
899     //---------------------------------------------------------------------
900     //  METHOD:         write()                                           -
901     /*********************************************************************
902     *********************************************************************/
903     public void
904     write( PrintStream writer )
905         throws IOException
906       {
907       DecimalFormat  value_fmt = new DecimalFormat( "0.0##E00" );
```

```
908
909       writer.printf(
910           "++release_base_activity pir_ndx=%d:%n%s%n",
911           this.index, IsotopeRec.toHeaderString()
912           );
913       for ( IsotopeRec iso_rec : this.base )
914         writer.println( iso_rec.toPrettyString( value_fmt ) );
915
916       writer.printf(
917           "%n+++depositors %d:%n%s%n",
918           this.index, IsotopeRec.toHeaderString()
919           );
920       for ( IsotopeRec iso_rec : this.depositors )
921         writer.println( iso_rec.toPrettyString( value_fmt ) );
922
923       writer.printf(
924           "%n+++non-depositors %d:%n%s%n",
925           this.index, IsotopeRec.toHeaderString()
926           );
927       for ( IsotopeRec iso_rec : this.nondepositors )
928         writer.println( iso_rec.toPrettyString( value_fmt ) );
929       } // write
930
931
932     //-----------------------------------------------------------------
933     //  METHOD:         readNext()                                     -
934     /*****************************************************************
935     * @return          next object read or null if no more
936     *****************************************************************/
937     public static ReleaseRec
938     readNext( BufferedReader reader )
939         throws IOException
940       {
941       ReleaseRec  rec = null;
942       int  index = -1;
943       IsotopeRec[]  base_acts = null, deps = null, non_deps = null;
944
945                 // Outer Read Loop
946                 //
947       for ( boolean finished = false; ! finished; )
948         {
949         String  line;
950         Matcher  m;
951
952                         // Check for Finished Sentinel
953                         //
954         if (
955             (line = reader.readLine()) == null ||
956             (line = line.trim()).contains( "+release: " ) ||
957             (line = line.trim()).contains( "+activityTableInputs: " )
958             )
959           finished = true;
960
961         else if (
962             line.startsWith( "++" ) &&
963             (m = PATTERN_releaseBaseActivity.matcher( line )).matches()
964             )
965           {
966           index = DataUtils.parseInt( m.group( 1 ), -1 );
967           base_acts = IsotopeRec.readArray( reader );
968           }
969
970         else if ( line.contains( "++depositor list" ) )
971           deps = IsotopeRec.readArray( reader );
972
```

```
973         else if ( line.contains( "++non-depositor list" ) )
974           {
975           non_deps = IsotopeRec.readArray( reader );
976           finished = true;
977           }
978         } // for outer loop
979
980       if ( index >= 0 )
981         rec = new ReleaseRec( index, base_acts, deps, non_deps );
982
983       return  rec;
984       } // readNext
985     } // ReleaseRec
986   } // NfacLog
```

C. RADFILE SOURCE LISTING

```
 1 // $Id$
 2 //--------------------------------------------------------------------
 3 //      NAME:           RadFile.java                                  -
 4 //      HISTORY:                                                      -
 5 //              2015-05-19      leerw@ornl.gov
 6 //        Added materialPIRIndexIncr property, used in
 7 //        readMaterialFactorsItem(), and incrPIRIndex().
 8 //              2015-05-18      leerw@ornl.gov
 9 //        Material index name is the first 10 chars, ci{nd,dp}NNssMM
10 //        Calling TestUtils.equals() on factors in MaterialDoseFactor.equals().
11 //              2014-12-12      leerw@ornl.gov
12 //        Fixed getMaterialIndexName() to always return the first
13 //        8 chars.
14 //              2014-10-03      leerw@ornl.gov
15 //        Re-structured to use SortedMap for easy comparisons.
16 //              2014-10-02      leerw@ornl.gov
17 //              2014-10-01      leerw@ornl.gov
18 //              2014-09-30      leerw@ornl.gov
19 //              2014-09-25      leerw@ornl.gov
20 //        Started readMaterialIsotopeActivityFactors().
21 //              2014-09-15      leerw@ornl.gov
22 //              2014-09-12      leerw@ornl.gov
23 //--------------------------------------------------------------------
24 package mil.dtra.hpac.models.nfac.test;
25
26 import java.io.BufferedReader;
27 import java.io.File;
28 import java.io.FileInputStream;
29 import java.io.IOException;
30 import java.io.InputStream;
31 import java.io.InputStreamReader;
32 import java.io.PrintStream;
33 import java.io.Reader;
34
35 import java.util.ArrayList;
36 import java.util.Arrays;
37 import java.util.Collections;
38 import java.util.HashMap;
39 import java.util.List;
40 import java.util.Map;
41 import java.util.Set;
42 import java.util.SortedMap;
43 import java.util.TreeMap;
44
45 import java.util.zip.GZIPInputStream;
46
47 import mil.dtra.hpac.util.DataUtils;
48
49
50 //--------------------------------------------------------------------
51 //      CLASS:          RadFile                                       -
52 /**********************************************************************
53 * Captures the NFAC log "FINE" level output showing inventories and
54 * activities.
55 **********************************************************************/
56 public class
57 RadFile
58   {
59                 // Constants
60                 //
61
62   public final static
```

```
 63     String              BASE_NAME = "nfac.test.RadFile";
 64                                 //RadFile.class.getPackage().getName();
 65
 66   public final static
 67     double[]            EMPTY_doubles = new double[ 0 ];
 68
 69   public final static
 70     String              HEADER_LINE =
 71         "#================================================" +
 72         System.getProperty( "line.separator" );
 73
 74   public final static
 75     String              PROP_debug = BASE_NAME + ".debug";
 76
 77 /*
 78   public final static
 79     String              REGEX_releaseBaseActivity =
 80 "^\\Q++\\Erelease_base_activity, pir_ndx=([0-9]+)/([0-9]+), decay_time=([\\d\\.]+).*$";
 81 // "^\\Q++\\Erelease_base_activity, pir_ndx=([0-9]+):$";
 82
 83   public final static
 84     Pattern             PATTERN_releaseBaseActivity =
 85         Pattern.compile( REGEX_releaseBaseActivity );
 86
 87   public final static
 88     Pattern             PATTERN_ws = Pattern.compile( "[\\s,]+" );
 89 */
 90
 91
 92                 // Class Attributes
 93                 //
 94
 95 /*
 96   protected final static
 97     Logger              fLogger_ =
 98         Logger.getLogger( SourceTermTableTest.class.getPackage().getName() );
 99 */
100
101
102                 // Object Attributes
103                 //
104
105   private
106     List<Tuple<String,int[]>>  fComponentDoses;
107     //AtomicReference<List<Tuple<String,int[]>>>  fComponentDosesRef;
108
109   private
110     Map<String,Integer> fComponentDosesIndex;
111     //AtomicReference<Map<String,Integer>>  fComponentDosesIndexRef;
112
113   private
114     String[]                    fHeaderLines;
115
116                                 /**
117                                 * by index name
118                                 */
119   private
120     SortedMap<String,MaterialDoseFactor>  fMaterialDoseFactors;
121     //AtomicReference<List<MaterialDoseFactor>>  fMaterialDoseFactorsRef;
122
123                                 /**
124                                 * by index name
125                                 */
126   private
127     SortedMap<String,MaterialDoseFactor>  fMaterialIsotopeActivityFactors;
```

```
128     //List<Tuple<String,int[]>>  fMaterialIsotopeActivityFactors;
129     //AtomicReference<Map<String,int[]>>  fMaterialIsotopeActivityFactorsRef;
130
131                                 /**
132                                 * by index name of full name
133                                 */
134   private
135     SortedMap<String,String>  fMaterialNames;
136     //AtomicReference<List<String>>  fMaterialNamesRef;
137
138   private
139     int                 fMaterialPIRIndexIncr;
140
141   private
142     List<Tuple<String,int[]>>  fPlottedDoses;
143     //AtomicReference<List<Tuple<String,int[]>>>  fPlottedDosesRef;
144
145   private
146     Map<String,Integer> fPlottedDosesIndex;
147     //AtomicReference<Map<String,Integer>>  fPlottedDosesIndexRef;
148
149   private
150     List<String>        fPlottedIsotopes;
151     //AtomicReference<List<String>>  fPlottedIsotopes;
152
153   private
154     Map<String,Integer> fPlottedIsotopesIndex;
155     //AtomicReference<Map<String,Integer>>  fPlottedIsotopesIndexRef;
156
157
158                 // Object Methods
159                 //
160
161
162   //---------------------------------------------------------------------
163   //    METHOD:         RadFile()                                       -
164   /**********************************************************************
165   **********************************************************************/
166   protected
167   RadFile( int mat_pir_index_incr )
168     {
169     setMaterialPIRIndexIncr( mat_pir_index_incr );
170
171     setComponentDoses( new ArrayList<Tuple<String,int[]>>( 0 ) );
172     setHeaderLines( null );
173     setMaterialDoseFactors( new TreeMap<String,MaterialDoseFactor>() );
174     setMaterialIsotopeActivityFactors( new TreeMap<String,MaterialDoseFactor>() );
175     setMaterialNames( new TreeMap<String,String>() );
176     setPlottedDoses( new ArrayList<Tuple<String,int[]>>( 0 ) );
177     setPlottedIsotopes( new ArrayList<String>( 0 ) );
178 /*
179     fComponentsDosesRef = new AtomicReference<List<Tuple<String,int[]>>>();
180     fMaterialDoseFactorsRef = new AtomicReference<List<MaterialDoseFactor>>();
181     fMaterialIsotopeActivityFactorsRef =
182         new AtomicReference<List<Tuple<String,int[]>>>();
183     fMaterialNamesRef = new AtomicReference<List<String>>();
184     fPlottedDosesRef = new AtomicReference<List<Tuple<String,int[]>>>();
185     fPlottedIsotopes = new AtomicReference<List<String>>();
186 */
187     } // RadFile
188
189
190   //---------------------------------------------------------------------
191   //    METHOD:         RadFile()                                       -
192   /**********************************************************************
```

```
193   ************************************************************************/
194   public
195   RadFile( File file )
196       throws IOException
197     {
198     this( file, 0 );
199     } // RadFile
200
201
202   //-----------------------------------------------------------------------
203   //    METHOD:         RadFile()                                         -
204   /************************************************************************
205   ************************************************************************/
206   public
207   RadFile( InputStream input )
208       throws IOException
209     {
210     this( input, 0 );
211     } // RadFile
212
213
214   //-----------------------------------------------------------------------
215   //    METHOD:         RadFile()                                         -
216   /************************************************************************
217   ************************************************************************/
218   public
219   RadFile( File file, int mat_pir_index_incr )
220       throws IOException
221     {
222     this( mat_pir_index_incr );
223     read( file );
224     } // RadFile
225
226
227   //-----------------------------------------------------------------------
228   //    METHOD:         RadFile()                                         -
229   /************************************************************************
230   ************************************************************************/
231   public
232   RadFile( InputStream input, int mat_pir_index_incr )
233       throws IOException
234     {
235     this( mat_pir_index_incr );
236     read( input );
237     } // RadFile
238
239
240   //-----------------------------------------------------------------------
241   //    METHOD:         equals()                                          -
242   /************************************************************************
243   ************************************************************************/
244   @Override
245   public boolean
246   equals( Object that )
247     {
248     return  !(that instanceof RadFile) ? false : equals( (RadFile)that );
249     } // equals
250
251
252   //-----------------------------------------------------------------------
253   //    METHOD:         equals()                                          -
254   /************************************************************************
255   ************************************************************************/
256   public boolean
257   equals( RadFile that )
```

```
258     {
259     boolean  equal = false;
260
261     ArrayList<Tuple<String,int[]>>  this_comp_doses =
262         this.fComponentDoses == null ? null :
263         new ArrayList<>( this.fComponentDoses );
264     Collections.sort( this_comp_doses );
265     ArrayList<Tuple<String,int[]>>  that_comp_doses =
266         that.fComponentDoses == null ? null :
267         new ArrayList<>( that.fComponentDoses );
268     Collections.sort( that_comp_doses );
269
270     ArrayList<Tuple<String,int[]>>  this_plotted_doses =
271         this.fPlottedDoses == null ? null :
272         new ArrayList<>( this.fPlottedDoses );
273     Collections.sort( this_plotted_doses );
274     ArrayList<Tuple<String,int[]>>  that_plotted_doses =
275         that.fPlottedDoses == null ? null :
276         new ArrayList<>( that.fPlottedDoses );
277     Collections.sort( that_plotted_doses );
278
279     ArrayList<String>  this_plotted_isos =
280         this.fPlottedIsotopes == null ? null :
281         new ArrayList<>( this.fPlottedIsotopes );
282     Collections.sort( this_plotted_isos );
283     ArrayList<String>  that_plotted_isos =
284         that.fPlottedIsotopes == null ? null :
285         new ArrayList<>( that.fPlottedIsotopes );
286     Collections.sort( that_plotted_isos );
287
288     Set<String>  this_mat_names =
289         this.fMaterialNames == null ? null : this.fMaterialNames.keySet();
290     Set<String>  that_mat_names =
291         that.fMaterialNames == null ? null : that.fMaterialNames.keySet();
292
293     if ( Boolean.getBoolean( PROP_debug ) )
294       {
295       boolean[]  eq = new boolean[ 7 ];
296
297       eq[ 0 ] = TestUtils.deepEquals( this.fHeaderLines, that.fHeaderLines );
298       System.err.printf( "[RadFile.equals] headerLines=%b%n", eq[ 0 ] );
299
300       eq[ 1 ] = TestUtils.deepEquals( this_comp_doses, that_comp_doses );
301       System.err.printf( "[RadFile.equals] componentDoses=%b%n", eq[ 1 ] );
302
303       eq[ 2 ] = TestUtils.deepEquals( this_plotted_doses, that_plotted_doses );
304       System.err.printf( "[RadFile.equals] plottedDoses=%b%n", eq[ 2 ] );
305
306       eq[ 3 ] = TestUtils.deepEquals( this_plotted_isos, that_plotted_isos );
307       System.err.printf( "[RadFile.equals] plottedIsotopes=%b%n", eq[ 3 ] );
308
309       eq[ 4 ] = TestUtils.deepEquals( this_mat_names, that_mat_names );
310       System.err.printf( "[RadFile.equals] materialNames=%b%n", eq[ 4 ] );
311
312       eq[ 5 ] = TestUtils.
313           deepEquals( this.fMaterialDoseFactors, that.fMaterialDoseFactors );
314       System.err.printf( "[RadFile.equals] materialDoseFactors=%b%n", eq[ 5 ] );
315
316       eq[ 6 ] = TestUtils.deepEquals(
317           this.fMaterialIsotopeActivityFactors,
318           that.fMaterialIsotopeActivityFactors
319           );
320       System.err.printf(
321           "[RadFile.equals] materialIsotopeActivityFactors=%b%n",
322           eq[ 6 ]
```

```
323           );
324
325       equal = true;
326       for ( int i = 0; i < eq.length && equal; i++ )
327         equal &= eq[ i ];
328       }
329
330     else
331       {
332       equal =
333           TestUtils.deepEquals( this.fHeaderLines, that.fHeaderLines ) &&
334           TestUtils.deepEquals( this_comp_doses, that_comp_doses ) &&
335           TestUtils.deepEquals( this_plotted_doses, that_plotted_doses ) &&
336           TestUtils.deepEquals( this_plotted_isos, that_plotted_isos ) &&
337           TestUtils.deepEquals( this_mat_names, that_mat_names ) &&
338           TestUtils.deepEquals(
339               this.fMaterialDoseFactors,
340               that.fMaterialDoseFactors
341               ) &&
342           TestUtils.deepEquals(
343               this.fMaterialIsotopeActivityFactors,
344               that.fMaterialIsotopeActivityFactors
345               );
346       }
347
348     return  equal;
349     } // equals
350
351
352   //---------------------------------------------------------------------
353   //    METHOD:         getComponentDoses()                            -
354   /**********************************************************************
355   * @return             object reference
356   **********************************************************************/
357   public List<Tuple<String,int[]>>
358   getComponentDoses()
359     {
360     return  fComponentDoses;
361     //return  getComponentDosesRef().get();
362     } // getComponentDoses
363
364
365   //---------------------------------------------------------------------
366   //    METHOD:         getComponentDosesIndex()                       -
367   /**********************************************************************
368   * Map by name of 1-based indices.
369   * @return             object reference
370   **********************************************************************/
371   public Map<String,Integer>
372   getComponentDosesIndex()
373     {
374     return  fComponentDosesIndex;
375     } // getComponentDosesIndex
376
377
378   //---------------------------------------------------------------------
379   //    METHOD:         getComponentDosesRef()                         -
380   /**********************************************************************
381   * @return             object reference
382   **********************************************************************/
383 /*
384   protected AtomicReference<List<Tuple<String,int[]>>>
385   getComponentDosesRef()
386     {
387     return  fComponentDosesRef;
```

```
388     } // getComponentDosesRef
389 */
390
391
392   //---------------------------------------------------------------------
393   //    METHOD:         getHeaderLines()                                -
394   /**********************************************************************
395   * @return             array reference, possibly null
396   **********************************************************************/
397   public String[]
398   getHeaderLines()
399     {
400     return  fHeaderLines;
401     } // getHeaderLines
402
403
404   //---------------------------------------------------------------------
405   //    METHOD:         getMaterialDoseFactors()                        -
406   /**********************************************************************
407   * Map by indexname of records.
408   * @return             object reference
409   **********************************************************************/
410   public SortedMap<String,MaterialDoseFactor>
411   getMaterialDoseFactors()
412     {
413     return  fMaterialDoseFactors;
414     //return  getMaterialDoseFactorsRef().get();
415     } // getMaterialDoseFactors
416
417
418   //---------------------------------------------------------------------
419   //    METHOD:         getMaterialIsotopeActivityFactors()             -
420   /**********************************************************************
421   * Map by indexname of records.
422   * @return             object reference
423   **********************************************************************/
424   public SortedMap<String,MaterialDoseFactor>
425   getMaterialIsotopeActivityFactors()
426     {
427     return  fMaterialIsotopeActivityFactors;
428     //return  getMaterialIsotopeActivityFactorsRef().get();
429     } // getMaterialIsotopeActivityFactors
430
431
432   //---------------------------------------------------------------------
433   //    METHOD:         getMaterialNames()                              -
434   /**********************************************************************
435   * Hash by index material name of full material names.
436   * @return             object reference
437   **********************************************************************/
438   public Map<String,String>
439   getMaterialNames()
440     {
441     return  fMaterialNames;
442     //return  getMaterialNamesRef().get();
443     } // getMaterialNames
444
445
446   //---------------------------------------------------------------------
447   //    METHOD:         getMaterialPIRIndexIncr()                       -
448   /**********************************************************************
449   **********************************************************************/
450   public int
451   getMaterialPIRIndexIncr()
452     {
```

```
453     return  fMaterialPIRIndexIncr;
454     } // getMaterialPIRIndexIncr
455
456
457   //-------------------------------------------------------------------
458   //    METHOD:         getPlottedDoses()                            -
459   /*******************************************************************
460   * @return            object reference
461   *******************************************************************/
462   public List<Tuple<String,int[]>>
463   getPlottedDoses()
464     {
465     return  fPlottedDoses;
466     //return  getPlottedDosesRef().get();
467     } // getPlottedDoses
468
469
470   //-------------------------------------------------------------------
471   //    METHOD:         getPlottedDosesIndex()                       -
472   /*******************************************************************
473   * Map by name of 1-based indices.
474   * @return            object reference
475   *******************************************************************/
476   public Map<String,Integer>
477   getPlottedDosesIndex()
478     {
479     return  fPlottedDosesIndex;
480     } // getPlottedDosesIndex
481
482
483   //-------------------------------------------------------------------
484   //    METHOD:         getPlottedIsotopes()                         -
485   /*******************************************************************
486   * @return            object reference
487   *******************************************************************/
488   public List<String>
489   getPlottedIsotopes()
490     {
491     return  fPlottedIsotopes;
492     //return  getPlottedIsotopesRef().get();
493     } // getPlottedIsotopes
494
495
496   //-------------------------------------------------------------------
497   //    METHOD:         getPlottedIsotopesIndex()                    -
498   /*******************************************************************
499   * Map by name of 1-based indices.
500   * @return            object reference
501   *******************************************************************/
502   public Map<String,Integer>
503   getPlottedIsotopesIndex()
504     {
505     return  fPlottedIsotopesIndex;
506     } // getPlottedIsotopesIndex
507
508
509   //-------------------------------------------------------------------
510   //    METHOD:         read()                                       -
511   /*******************************************************************
512   * Calls {@link #read( InputStream )}.
513   *******************************************************************/
514   public void
515   read( File file )
516       throws IOException
517     {
```

```
518     //FileInputStream  file_in = new FileInputStream( file );
519     InputStream  input = new FileInputStream( file );
520
521     if ( file.getName().endsWith( ".gz" ) )
522       input = new GZIPInputStream( input );
523
524     try
525       { read( input ); }
526     finally
527       { input.close(); }
528     } // read
529
530
531   //-----------------------------------------------------------------------
532   //    METHOD:         read()                                           -
533   /************************************************************************
534   ************************************************************************/
535   public void
536   read( InputStream input )
537       throws IOException
538     {
539     read( new InputStreamReader( input ) );
540     } // read
541
542
543   //-----------------------------------------------------------------------
544   //    METHOD:         read()                                           -
545   /************************************************************************
546   * Implementation method.
547   ************************************************************************/
548   public void
549   read( Reader reader_in )
550       throws IOException
551     {
552     BufferedReader  reader =
553        (reader_in instanceof BufferedReader) ?
554        (BufferedReader)reader_in : new BufferedReader( reader_in );
555
556     setHeaderLines( new String[]{ reader.readLine(), reader.readLine() } );
557
558     ArrayList<Tuple<String,int[]>>  component_doses = new ArrayList<>( 128 );
559     readComponentDoses( reader, component_doses );
560     setComponentDoses( component_doses );
561
562     ArrayList<String>  component_dose_names =
563         new ArrayList<>( component_doses.size() );
564     for ( Tuple<String,int[]> item : component_doses )
565       component_dose_names.add( item.x );
566
567     ArrayList<Tuple<String,int[]>>  plotted_doses = new ArrayList<>( 128 );
568     readPlottedDoses( reader, plotted_doses );
569     setPlottedDoses( plotted_doses );
570
571     TreeMap<String,MaterialDoseFactor>  mat_dose_factors = new TreeMap<>();
572     TreeMap<String,String>  mat_names = new TreeMap<>();
573     readMaterialDoseFactors(
574         reader, component_dose_names, mat_dose_factors, mat_names
575         );
576     setMaterialDoseFactors( mat_dose_factors );
577     setMaterialNames( mat_names );
578
579     ArrayList<String>  plotted_isotopes = new ArrayList<>( 128 );
580     readPlottedIsotopes( reader, plotted_isotopes );
581     setPlottedIsotopes( plotted_isotopes );
582
```

```
583     TreeMap<String,MaterialDoseFactor>  mat_iso_activity_factors =
584         new TreeMap<>();
585     readMaterialIsotopeActivityFactors(
586         reader, getPlottedIsotopes(), mat_iso_activity_factors
587         );
588     setMaterialIsotopeActivityFactors( mat_iso_activity_factors );
589     } // read
590
591
592   //------------------------------------------------------------------------
593   //    METHOD:         readComponentDoses()                              -
594   /*************************************************************************
595   *************************************************************************/
596   protected void
597   readComponentDoses( BufferedReader reader, List<Tuple<String,int[]>> list )
598       throws IOException
599     {
600     String  line, line2;
601
602     while ( !(
603         (line = reader.readLine()) == null || line.contains( "Component Doses" )
604         ) )
605       ;
606
607     reader.readLine();
608
609     if ( (line = reader.readLine()) != null )
610       {
611       int  count = DataUtils.parseInt( line.trim() );
612
613       for ( int i = 0; i < count; i++ )
614         {
615         if (
616             (line = reader.readLine()) != null &&
617             (line2 = reader.readLine()) != null
618             )
619           {
620           String  name = line.trim();
621           int  value = DataUtils.parseInt( line2.trim() );
622           list.add( new Tuple<String,int[]>( name, new int[]{ value } ) );
623           } // if both lines
624         } // for each dose
625       } // if we have a count line
626     } // readComponentDoses
627
628
629   //------------------------------------------------------------------------
630   //    METHOD:         readMaterialDoseFactors()                         -
631   /*************************************************************************
632   *************************************************************************/
633   protected void
634   readMaterialDoseFactors(
635       BufferedReader    reader,
636       List<String>      ref_names,
637       SortedMap<String,MaterialDoseFactor>  factors_map,
638       Map<String,String>  names
639       )
640       throws IOException
641     {
642     String  line, line2;
643
644     while ( !(
645         (line = reader.readLine()) == null ||
646         line.contains( "Material Dose Factors" )
647         ) )
```

```
      ;

    reader.readLine();

    if ( (line = reader.readLine()) != null )
      {
      int  count = DataUtils.parseInt( line.trim() );

      for ( int i = 0; i < count; i++ )
        readMaterialFactorsItem( reader, ref_names, factors_map, names, 3 );
      } // if count line
    } // readMaterialDoseFactors


  //-------------------------------------------------------------------------
  //    METHOD:         readMaterialFactorsItem()                          -
  /*************************************************************************
  *************************************************************************/
  protected void
  readMaterialFactorsItem(
      BufferedReader    reader,
      List<String>      ref_names,
      SortedMap<String,MaterialDoseFactor>  factors_map,
      Map<String,String>  names,
      int               numbers_count
      )
      throws IOException
    {
    String  line;

    if ( (line = reader.readLine()) != null )
      {
      String  full_name = line.trim();
      String  index_name = getMaterialIndexName( full_name );

      if ( getMaterialPIRIndexIncr() != 0 )
        {
        full_name = incrPIRIndex( full_name, getMaterialPIRIndexIncr() );
        index_name = incrPIRIndex( index_name, getMaterialPIRIndexIncr() );
        }

                        // Update Names Map
                        //
      if ( names != null )
        names.put( index_name, full_name );

                        // Times
                        //
      double[]  times = EMPTY_doubles;
      int  times_count = 0;

      if ( (line = reader.readLine()) != null )
        {
        times_count = DataUtils.parseInt( line.trim() );
        times = readValuesBlock( reader, times_count );
        }

                        // Mystery Numbers
                        //
      double[]  numbers = null;
      if ( numbers_count > 0 )
        {
        numbers = new double[ numbers_count ];
        for ( int k = 0; k < numbers.length; k++ )
          {
```

```
713           numbers[ k ] =
714               (line = reader.readLine()) == null ? 0.0 :
715               DataUtils.parseDouble( line.trim(), 0.0 );
716           }
717         }
718
719                         // Factors
720                         //
721       //double[][]  factors = null;
722       //int[]  factor_indexes = null;
723       TreeMap<String,double[]>  factors = null;
724
725       if ( (line = reader.readLine()) != null )
726         {
727         int  factors_count = DataUtils.parseInt( line.trim() );
728         //factors = new double[ factors_count ][];
729         //factor_indexes = new int[ factors_count ];
730         //Arrays.fill( factors, EMPTY_doubles );
731         factors = new TreeMap<String,double[]>();
732
733         for ( int i = 0; i < factors_count; i++ )
734           {
735           int  mat_index;
736
737           if (
738               (line = reader.readLine()) != null &&
739               (mat_index = DataUtils.parseInt( line.trim(), 0 )) > 0 &&
740               mat_index <= ref_names.size()
741               )
742             {
743             factors.put(
744                 ref_names.get( mat_index - 1 ),
745                 readValuesBlock( reader, times_count )
746                 );
747 /*
748             factors.put(
749                 new Integer( mat_index ),
750                 readValuesBlock( reader, times_count )
751                 );
752 */
753             }
754           }
755         } // if factors count line
756
757                         // Update List
758                         //
759       MaterialDoseFactor  rec = new MaterialDoseFactor(
760           index_name, full_name,
761           times, factors, numbers
762           //times, factor_indexes, factors, numbers
763           );
764       factors_map.put( index_name, rec );
765       } // if name line read
766     } // readMaterialFactorsItem
767
768
769   //------------------------------------------------------------------------
770   //    METHOD:         readMaterialIsotopeActivityFactors()            -
771   /*************************************************************************
772   *************************************************************************/
773   protected void
774   readMaterialIsotopeActivityFactors(
775       BufferedReader    reader,
776       List<String>      ref_names,
777       SortedMap<String,MaterialDoseFactor>  factors_map
```

```
778       )
779       throws IOException
780     {
781     String  line, line2;
782
783     while ( !(
784         (line = reader.readLine()) == null ||
785         line.contains( "Material Isotope Activity Factors" )
786         ) )
787       ;
788
789     reader.readLine();
790
791     if ( (line = reader.readLine()) != null )
792       {
793       int  count = DataUtils.parseInt( line.trim() );
794
795       for ( int i = 0; i < count; i++ )
796         readMaterialFactorsItem( reader, ref_names, factors_map, null, 0 );
797       } // if count line
798     } // readMaterialIsotopeActivityFactors
799
800
801   //-------------------------------------------------------------------
802   //    METHOD:         readPlottedDoses()                           -
803   /*******************************************************************
804   *******************************************************************/
805   protected void
806   readPlottedDoses( BufferedReader reader, List<Tuple<String,int[]>> list )
807       throws IOException
808     {
809     String  line, line2, line3;
810
811     while ( !(
812         (line = reader.readLine()) == null || line.contains( "Plotted Doses" )
813         ) )
814       ;
815
816     reader.readLine();
817
818     if ( (line = reader.readLine()) != null )
819       {
820       int  count = DataUtils.parseInt( line.trim() );
821
822       for ( int i = 0; i < count; i++ )
823         {
824         if (
825             (line = reader.readLine()) != null &&
826             (line2 = reader.readLine()) != null &&
827             (line3 = reader.readLine()) != null
828             )
829           {
830           String  name = line.trim();
831
832           String[]  tokens = line3.trim().split( "[\\s,]+" );
833           int[]  value_arr = new int[ tokens.length + 1 ];
834           value_arr[ 0 ] = DataUtils.parseInt( line2.trim() );
835
836           for ( int k = 0; k < tokens.length; k++ )
837             value_arr[ k + 1 ] = DataUtils.parseInt( tokens[ k ] );
838
839           list.add( new Tuple<String,int[]>( name, value_arr ) );
840           } // if both lines
841         } // for each dose
842       } // if we have a count line
```

```
843     } // readPlottedDoses
844
845
846   //------------------------------------------------------------------------
847   //    METHOD:         readPlottedIsotopes()                             -
848   /*************************************************************************
849   *************************************************************************/
850   protected void
851   readPlottedIsotopes( BufferedReader reader, List<String> list )
852       throws IOException
853     {
854     String  line;
855
856     while ( !(
857         (line = reader.readLine()) == null || line.contains( "Plotted Isotopes" )
858         ) )
859       ;
860
861     reader.readLine();
862
863     if ( (line = reader.readLine()) != null )
864       {
865       int  count = DataUtils.parseInt( line.trim() );
866
867       for ( int i = 0; i < count; i++ )
868         {
869         if ( (line = reader.readLine()) != null )
870           list.add( line.trim() );
871         } // for each dose
872       } // if we have a count line
873     } // readPlottedIsotopes
874
875
876   //------------------------------------------------------------------------
877   //    METHOD:         setComponentDoses()                               -
878   /*************************************************************************
879   * @param  value       object reference to store
880   *************************************************************************/
881   public void
882   setComponentDoses( List<Tuple<String,int[]>> value )
883     {
884     setComponentDosesRef( value );
885
886     Map<String,Integer>  index = new HashMap<>( value.size() << 1 );
887
888     for ( int i = 0; i < value.size(); i++ )
889       index.put( value.get( i ).x, new Integer( i + 1 ) );
890
891     setComponentDosesIndex( index );
892     } // setComponentDoses
893
894
895   //------------------------------------------------------------------------
896   //    METHOD:         setComponentDosesRef()                            -
897   /*************************************************************************
898   * @param  value       object reference to store
899   *************************************************************************/
900   protected void
901   setComponentDosesRef( List<Tuple<String,int[]>> value )
902     {
903     fComponentDoses = Collections.unmodifiableList( value );
904     } // setComponentDosesRef
905
906
907   //------------------------------------------------------------------------
```

```
908   //    METHOD:         setComponentDosesIndex()                        -
909   /*************************************************************************
910   * Map by name of 1-based indices.
911   * @param  value       object reference to store
912   *************************************************************************/
913   protected void
914   setComponentDosesIndex( Map<String,Integer> value )
915     {
916     fComponentDosesIndex = Collections.unmodifiableMap( value );
917     } // setComponentDosesIndex
918
919
920   //-----------------------------------------------------------------------
921   //    METHOD:         setHeaderLines()                                -
922   /*************************************************************************
923   * @param  value       array reference to store
924   *************************************************************************/
925   public void
926   setHeaderLines( String[] value )
927     {
928     fHeaderLines = value;
929     } // setHeaderLines
930
931
932   //-----------------------------------------------------------------------
933   //    METHOD:         setMaterialDoseFactors()                        -
934   /*************************************************************************
935   * Records by 1-based index - 1.
936   * @param  value       object reference to store
937   *************************************************************************/
938   public void
939   setMaterialDoseFactors( SortedMap<String,MaterialDoseFactor> value )
940     {
941     fMaterialDoseFactors = Collections.unmodifiableSortedMap( value );
942     //getMaterialDoseFactorsRef().set( Collections.unmodifiableList( value ) );
943     } // setMaterialDoseFactors
944
945
946   //-----------------------------------------------------------------------
947   //    METHOD:         setMaterialIsotopeActivityFactors()             -
948   /*************************************************************************
949   * Rec by plotted isotope 1-based index - 1.
950   * @param  value       object reference to store
951   *************************************************************************/
952   public void
953   setMaterialIsotopeActivityFactors( SortedMap<String,MaterialDoseFactor> value )
954     {
955     fMaterialIsotopeActivityFactors = Collections.unmodifiableSortedMap( value );
956 /*
957     getMaterialIsotopeActivityFactorsRef().
958         set( Collections.unmodifiableMap( value ) );
959 */
960     } // setMaterialIsotopeActivityFactors
961
962
963   //-----------------------------------------------------------------------
964   //    METHOD:         setMaterialNames()                              -
965   /*************************************************************************
966   * Hash by index material name of full material names.
967   * @param  value       object reference to store
968   *************************************************************************/
969   public void
970   setMaterialNames( SortedMap<String,String> value )
971     {
972     fMaterialNames = Collections.unmodifiableSortedMap( value );
```

```
 973     //getMaterialNamesRef().set( Collections.unmodifiableList( value ) );
 974     } // setMaterialNames
 975
 976
 977   //------------------------------------------------------------------------
 978   //    METHOD:         setMaterialPIRIndexIncr()                         -
 979   /**************************************************************************
 980   **************************************************************************/
 981   public void
 982   setMaterialPIRIndexIncr( int value )
 983     {
 984     fMaterialPIRIndexIncr = value;
 985     } // setMaterialPIRIndexIncr
 986
 987
 988   //------------------------------------------------------------------------
 989   //    METHOD:         setPlottedDoses()                                 -
 990   /**************************************************************************
 991   * @param  value       object reference to store
 992   **************************************************************************/
 993   public void
 994   setPlottedDoses( List<Tuple<String,int[]>> value )
 995     {
 996     //getPlottedDosesRef().set( Collections.unmodifiableMap( value ) );
 997     setPlottedDosesRef( value );
 998
 999     Map<String,Integer>  index = new HashMap<>( value.size() << 1 );
1000
1001     for ( int i = 0; i < value.size(); i++ )
1002       index.put( value.get( i ).x, new Integer( i + 1 ) );
1003
1004     setPlottedDosesIndex( index );
1005     } // setPlottedDoses
1006
1007
1008   //------------------------------------------------------------------------
1009   //    METHOD:         setPlottedDosesRef()                              -
1010   /**************************************************************************
1011   * @param  value       object reference to store
1012   **************************************************************************/
1013   protected void
1014   setPlottedDosesRef( List<Tuple<String,int[]>> value )
1015     {
1016     fPlottedDoses = Collections.unmodifiableList( value );
1017     } // setPlottedDosesRef
1018
1019
1020   //------------------------------------------------------------------------
1021   //    METHOD:         setPlottedDosesIndex()                            -
1022   /**************************************************************************
1023   * Map by name of 1-based indices.
1024   * @param  value       object reference to store
1025   **************************************************************************/
1026   protected void
1027   setPlottedDosesIndex( Map<String,Integer> value )
1028     {
1029     fPlottedDosesIndex = Collections.unmodifiableMap( value );
1030     } // setPlottedDosesIndex
1031
1032
1033   //------------------------------------------------------------------------
1034   //    METHOD:         setPlottedIsotopes()                              -
1035   /**************************************************************************
1036   * @param  value       object reference to store
1037   **************************************************************************/
```

```
1038   public void
1039   setPlottedIsotopes( List<String> value )
1040     {
1041     setPlottedIsotopesRef( value );
1042
1043     Map<String,Integer>  index = new HashMap<>( value.size() << 1 );
1044
1045     for ( int i = 0; i < value.size(); i++ )
1046       index.put( value.get( i ), new Integer( i + 1 ) );
1047
1048     setPlottedIsotopesIndex( index );
1049     } // setPlottedIsotopes
1050
1051
1052   //------------------------------------------------------------------------
1053   //    METHOD:         setPlottedIsotopesRef()                           -
1054   /*************************************************************************
1055   * @param  value       object reference to store
1056   *************************************************************************/
1057   protected void
1058   setPlottedIsotopesRef( List<String> value )
1059     {
1060     fPlottedIsotopes = Collections.unmodifiableList( value );
1061     } // setPlottedIsotopesRef
1062
1063
1064   //------------------------------------------------------------------------
1065   //    METHOD:         setPlottedIsotopesIndex()                         -
1066   /*************************************************************************
1067   * Map by name of 1-based indices.
1068   * @param  value       object reference to store
1069   *************************************************************************/
1070   protected void
1071   setPlottedIsotopesIndex( Map<String,Integer> value )
1072     {
1073     fPlottedIsotopesIndex = Collections.unmodifiableMap( value );
1074     } // setPlottedIsotopesIndex
1075
1076
1077   //------------------------------------------------------------------------
1078   //    METHOD:         write()                                           -
1079   /*************************************************************************
1080   *************************************************************************/
1081   public void
1082   write( PrintStream print )
1083       throws IOException
1084     {
1085     if ( getHeaderLines() != null )
1086       {
1087       for ( String line : getHeaderLines() )
1088         print.println( line );
1089       }
1090
1091     print.printf( HEADER_LINE + "Component Doses%n" + HEADER_LINE );
1092     List<Tuple<String,int[]>>  comp_doses = getComponentDoses();
1093     writeTuples( print, comp_doses, false );
1094
1095     print.printf( HEADER_LINE + "Plotted Doses%n" + HEADER_LINE );
1096     List<Tuple<String,int[]>>  plotted_doses = getPlottedDoses();
1097     writeTuples( print, plotted_doses, true );
1098
1099     print.printf( HEADER_LINE + "Material Dose Factors%n" + HEADER_LINE );
1100     SortedMap<String,MaterialDoseFactor>  mat_dose_factors =
1101         getMaterialDoseFactors();
1102     print.printf( " %d%n", mat_dose_factors.size() );
```

```
1103    for ( MaterialDoseFactor mat_dose_factor : mat_dose_factors.values() )
1104      mat_dose_factor.write( print, getComponentDosesIndex() );
1105
1106    print.printf( HEADER_LINE + "Plotted Isotopes%n" + HEADER_LINE );
1107    List<String>  plotted_isos = getPlottedIsotopes();
1108    print.printf( " %d%n", plotted_isos.size() );
1109    for ( String iso : plotted_isos )
1110      print.println( iso );
1111
1112    print.printf( HEADER_LINE + "Material Isotope Activity Factors%n" + HEADER_LINE );
1113    SortedMap<String,MaterialDoseFactor>  mat_iso_factors = getMaterialIsotopeActivityFactors();
1114    print.printf( " %d%n", mat_iso_factors.size() );
1115    for ( MaterialDoseFactor mat_iso_factor : mat_iso_factors.values() )
1116      mat_iso_factor.write( print, getPlottedIsotopesIndex() );
1117    } // write
1118
1119
1120  //---------------------------------------------------------------------
1121  //    METHOD:         writeTuples()                                   -
1122  /*********************************************************************
1123  *********************************************************************/
1124  public void
1125  writeTuples(
1126      PrintStream       print,
1127      List<Tuple<String,int[]>>  list,
1128      boolean           multi_line
1129      )
1130      throws IOException
1131    {
1132    if ( list != null )
1133      {
1134      print.printf( " %d%n", list.size() );
1135      for ( Tuple<String,int[]> tuple : list )
1136        {
1137        int  st_ndx = 0;
1138
1139        print.println( tuple.x );
1140        if ( tuple.y.length > 1 && multi_line )
1141          {
1142          print.printf( " %d%n", tuple.y[ 0 ] );
1143          st_ndx = 1;
1144          }
1145
1146        while ( st_ndx < tuple.y.length )
1147          print.printf( " %d", tuple.y[ st_ndx++ ] );
1148        print.println();
1149        } // for
1150      } // if
1151    } // writeTuples
1152
1153
1154                // Class Methods
1155                //
1156
1157
1158  //---------------------------------------------------------------------
1159  //    METHOD:         getMaterialIndexName()                          -
1160  /*********************************************************************
1161  *********************************************************************/
1162  public static String
1163  getMaterialIndexName( String full_name )
1164    {
1165    return
1166        full_name == null ? null :
1167        full_name.length() >= 10 ? full_name.substring( 0, 10 ) :
```

```
1168        full_name;
1169    } // getMaterialIndexName
1170
1171
1172  //-------------------------------------------------------------------
1173  //    METHOD:         incrPIRIndex()                                -
1174  /*******************************************************************
1175  *******************************************************************/
1176  public static String
1177  incrPIRIndex( String mat_name, int incr )
1178    {
1179    String  result = mat_name;
1180
1181    if ( mat_name.length() >= 6 )
1182      {
1183      int  cur_index = DataUtils.parseInt( mat_name.substring( 4, 6 ), 0 );
1184      result = String.format(
1185          "%s%02d%s",
1186          mat_name.substring( 0, 4 ),
1187          Math.max( 0, Math.min( 99, cur_index + incr ) ),
1188          mat_name.substring( 6 )
1189          );
1190      } // if minimum required length
1191
1192    return  result;
1193    } // incrPIRIndex
1194
1195
1196  //-------------------------------------------------------------------
1197  //    METHOD:         main()                                        -
1198  /*******************************************************************
1199  *******************************************************************/
1200  public static void
1201  main( String[] argv )
1202    {
1203    try
1204      {
1205      File  one = null, two = null;
1206      int  pir_incr = 0;
1207      boolean  usage_flag = false;
1208
1209      for ( int i = 0; i < argv.length; i++ )
1210        {
1211        if ( "-h".equals( argv[ i ] ) )
1212          usage_flag = true;
1213
1214        else if ( "-i".equals( argv[ i ] ) && i < argv.length -1 )
1215          pir_incr = DataUtils.parseInt( argv[ ++i ], 0 );
1216
1217        else if ( one == null )
1218          one = new File( argv[ i ] );
1219          //one = new RadFile( new File( argv[ i ] ) );
1220
1221        else
1222          two = new File( argv[ i ] );
1223          //two = new RadFile( new File( argv[ i ] ) );
1224        } // for
1225
1226      if ( usage_flag || (one == null && two == null) )
1227        System.out.println( "Usage: RadFile [ -i pir_incr ] file1 [ file2 ]" );
1228
1229      else if ( two == null )
1230        new RadFile( one, pir_incr ).write( System.out );
1231
1232      else
```

```
1233        {
1234        RadFile
1235            rone = new RadFile( one, pir_incr ),
1236            rtwo = new RadFile( two );
1237        System.out.println( rone.equals( rtwo ) ? "EQUAL" : "not equal" );
1238        }
1239      } // try
1240
1241    catch ( Exception ex )
1242      {
1243      ex.printStackTrace( System.err );
1244      }
1245    } // main
1246
1247
1248  //-------------------------------------------------------------------
1249  //    METHOD:         toDecimal()                                    -
1250  /*******************************************************************
1251  *******************************************************************/
1252  public static String
1253  toDecimal( int width, int dec_index, double value )
1254    {
1255    StringBuilder  buffer = new StringBuilder( width + 16 );
1256    String  value_str = Double.toString( value );
1257    int  dot_index = value_str.indexOf( "." );
1258
1259    String  left, right;
1260    if ( dot_index < 0 )
1261      {
1262      left = value_str;
1263      right = "";
1264      }
1265    else
1266      {
1267      left = value_str.substring( 0, dot_index );
1268      right = value_str.substring( dot_index );
1269      }
1270
1271    int  padding = dec_index - left.length() - 1;
1272    if ( padding > 0 )
1273      buffer.append( String.format( String.format( "%%%ds", padding ), " " ) );
1274
1275    buffer.append( left ).append( right );
1276
1277    padding = width - buffer.length();
1278    if ( padding > 0 )
1279      buffer.append( String.format( String.format( "%%%ds", padding ), " " ) );
1280
1281    value_str = buffer.toString();
1282    if ( value_str.length() > width )
1283      {
1284      if ( value_str.contains( "E" ) || value_str.contains( "e" ) )
1285        {
1286        int  count = 0, len = value_str.length();
1287        while ( value_str.charAt( count ) == ' ' && len > width )
1288          {
1289          count++;
1290          len--;
1291          }
1292        value_str = value_str.substring( count );
1293        }
1294      else
1295        value_str = value_str.substring( 0, width );
1296      } // if must shift
1297
```

```
1298    return  value_str;
1299    } // toDecimal
1300
1301
1302  //-------------------------------------------------------------------
1303  //    METHOD:         readValuesBlock()                            -
1304  /*******************************************************************
1305  *******************************************************************/
1306  public static double[]
1307  readValuesBlock( BufferedReader reader, int count )
1308      throws IOException
1309    {
1310    double[]  values = new double[ count ];
1311    String  line;
1312
1313    for ( int ndx = 0; ndx < count && (line = reader.readLine()) != null; )
1314      {
1315      for ( String token : line.trim().split( "[\\s,]+" ) )
1316        {
1317        values[ ndx++ ] = DataUtils.parseDouble( token );
1318        if ( ndx >= count )  break;
1319        }
1320      } // for lines
1321
1322    return  values;
1323    } // readValuesBlock
1324
1325
1326  //-------------------------------------------------------------------
1327  //    METHOD:         writeValuesBlock()                           -
1328  /*******************************************************************
1329  *******************************************************************/
1330  public static void
1331  writeValuesBlock( PrintStream print, double[] values )
1332      throws IOException
1333    {
1334    if ( values != null && values.length > 0 )
1335      {
1336      int  count = 0;
1337      for ( double value : values )
1338        {
1339        print.printf( "%20s", toDecimal( 20, 10, value ) );
1340        if ( ++count >= 6 )
1341          {
1342          print.println();
1343          count = 0;
1344          }
1345        }
1346
1347      if ( count != 0 )
1348        print.println();
1349      } // if values
1350    } // writeValuesBlock
1351
1352
1353                // Inner Classes
1354                //
1355
1356
1357  //-------------------------------------------------------------------
1358  //    CLASS:          MaterialDoseFactor                           -
1359  /*******************************************************************
1360  *******************************************************************/
1361  public static class
1362  MaterialDoseFactor
```

```
1363      implements Comparable<MaterialDoseFactor>
1364    {
1365    //public final SortedMap<Integer,double[]>  factors;
1366    public final SortedMap<String,double[]>  factors;
1367
1368    public final String  fullname;
1369
1370    public final String  indexname;
1371
1372    public final double[]  numbers;
1373
1374    public final double[]  times;
1375
1376
1377    //-----------------------------------------------------------------------
1378    //  METHOD:         MaterialDoseFactor.MaterialDoseFactor()         -
1379    /*************************************************************************
1380    *************************************************************************/
1381    public
1382    MaterialDoseFactor(
1383        String          index_name,
1384        String          full_name,
1385        double[]        times,
1386        //SortedMap<Integer,double[]>  factors,
1387        SortedMap<String,double[]>  factors,
1388        double[]        numbers
1389        )
1390      {
1391      this.factors = factors;
1392      this.fullname = full_name;
1393      this.indexname = index_name;
1394      this.numbers = numbers;
1395      this.times = times;
1396      } // MaterialDoseFactor
1397
1398
1399    //-----------------------------------------------------------------------
1400    //  METHOD:         compareTo()                                     -
1401    /*************************************************************************
1402    * Compares based on indexname.
1403    *************************************************************************/
1404    public int
1405    compareTo( MaterialDoseFactor that )
1406      {
1407      return
1408          this.indexname == null ? (that.indexname == null ? 0 : -1) :
1409          that.indexname == null ? 1 :
1410          this.indexname.compareTo( that.indexname );
1411      } // compareTo
1412
1413
1414    //-----------------------------------------------------------------------
1415    //  METHOD:         MaterialDoseFactor.equals()                     -
1416    /*************************************************************************
1417    *************************************************************************/
1418    public boolean
1419    equals( Object that )
1420      {
1421      boolean  equal = false;
1422
1423      if ( that instanceof MaterialDoseFactor )
1424        {
1425        MaterialDoseFactor  mf = (MaterialDoseFactor)that;
1426
1427        equal =
```

```
1428            //x no DataUtils.equals( this.fullname, mf.fullname ) &&
1429            DataUtils.equals( this.indexname, mf.indexname ) &&
1430            Arrays.equals( this.times, mf.times ) &&
1431            Arrays.equals( this.numbers, mf.numbers ) &&
1432            TestUtils.getInstance().equals( this.factors, mf.factors );
1433            //TestUtils.deepEquals( this.factors, mf.factors );
1434        } // if
1435
1436      return  equal;
1437      } // equals
1438
1439
1440    //-----------------------------------------------------------------------
1441    //  METHOD:         MaterialDoseFactor.write()                          -
1442    /*************************************************************************
1443    *************************************************************************/
1444    public void
1445    write( PrintStream print, Map<String,Integer> ref_index_map )
1446        throws IOException
1447      {
1448      print.println( this.fullname );
1449
1450      if ( this.times == null )
1451        print.println( " 0" );
1452      else
1453        {
1454        print.printf( " %d%n", this.times.length );
1455        writeValuesBlock( print, this.times );
1456        } // if we have times
1457
1458      if ( this.numbers == null )
1459        ;
1460
1461      else if ( this.numbers.length >= 3 )
1462        {
1463        print.printf(
1464            " %d%n%s%n %d%n",
1465            (int)this.numbers[ 0 ],
1466            toDecimal( 20, 10, this.numbers[ 1 ] ),
1467            (int)this.numbers[ 2 ]
1468            );
1469        }
1470
1471      else
1472        {
1473        for ( double cur_number : numbers )
1474          print.printf( " %d%n", (int)cur_number );
1475        }
1476
1477      if ( this.factors == null || this.factors.size() == 0 )
1478        print.println( " 0" );
1479      else
1480        {
1481        print.printf( " %d%n", this.factors.size() );
1482        for ( Map.Entry<String,double[]> item : this.factors.entrySet() )
1483          {
1484          if ( item.getValue() != null )
1485            {
1486            Integer  ref_index = ref_index_map.get( item.getKey() );
1487
1488            print.printf(
1489                " %d%n",
1490                ref_index == null ? 0 : ref_index.intValue()
1491                );
1492            writeValuesBlock( print, item.getValue() );
```

```
1493            }
1494          }
1495 /*
1496        for ( Map.Entry<Integer,double[]> item : this.factors.entrySet() )
1497          {
1498          if ( item.getValue() != null )
1499            {
1500            print.printf( " %d%n", item.getKey() );
1501            writeValuesBlock( print, item.getValue() );
1502            }
1503          }
1504 */
1505        } // else factors exists
1506      } // MaterialDoseFactor
1507    } // MaterialDoseFactor
1508
1509
1510  //---------------------------------------------------------------------
1511  //    CLASS:          Tuple                                           -
1512  /*********************************************************************
1513  *********************************************************************/
1514  public static class
1515  Tuple<X extends Comparable<X>, Y>
1516      implements Comparable<Tuple<X, Y>>
1517    {
1518    public final X      x;
1519    public final Y      y;
1520
1521
1522    //---------------------------------------------------------------------
1523    //  METHOD:         Tuple.Tuple()                                     -
1524    /*********************************************************************
1525    *********************************************************************/
1526    public
1527    Tuple( X x, Y y )
1528      {
1529      this.x = x;
1530      this.y = y;
1531      } // Tuple
1532
1533
1534    //---------------------------------------------------------------------
1535    //  METHOD:         compareTo()                                       -
1536    /*********************************************************************
1537    *********************************************************************/
1538    public int
1539    compareTo( Tuple<X, Y> that )
1540      {
1541      return
1542          this.x == null ? (that.x == null ? 0 : -1) :
1543          that.x == null ? 1 :
1544          this.x.compareTo( that.x );
1545      } // compareTo
1546
1547
1548    //---------------------------------------------------------------------
1549    //  METHOD:         Tuple.equals()                                    -
1550    /*********************************************************************
1551    *********************************************************************/
1552    public boolean
1553    equals( Object that )
1554      {
1555      boolean  equal = false;
1556
1557      if ( that instanceof Tuple )
```

```
1558         {
1559         Tuple  t = (Tuple)that;
1560
1561         equal =
1562             TestUtils.deepEquals( this.x, t.x ) &&
1563             TestUtils.deepEquals( this.y, t.y );
1564         }
1565
1566       return  equal;
1567       } // equals
1568     } // Tuple
1569   } // RadFile
```

ORNL/TM-2015/726

INTERNAL DISTRIBUTION

1. R. W. Lee
2. R. H. Morris
3. C. D. Sulfredge
4. ORNL Office of Technical Information and Classification

EXTERNAL DISTRIBUTION

5. Todd Hann, Chief, Hazards & Effects Modeling Division, Information Sciences & Applications Department, Defense Threat Reduction Agency, 8725 John J. Kingman Rd., Stop 6201, Ft Belvoir, VA 22060-6201