

OAK RIDGE NATIONAL LABORATORY

managed by
UT-BATTELLE, LLC
for the
U.S. DEPARTMENT OF ENERGY

RSICC COMPUTER CODE COLLECTION

ADVANTG 3.0.5

AutomateD VAriaNce reducTion Generator

Contributed by:

Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA



RADIATION SAFETY INFORMATION COMPUTATIONAL CENTER

Legal Notice: This material was prepared as an account of Government sponsored work and describes a code system or data library which is one of a series collected by the Radiation Safety Information Computational Center (RSICC). These codes/data were developed by various Government and private organizations who contributed them to RSICC for distribution; they did not normally originate at RSICC. RSICC is informed that each code system has been tested by the contributor, and, if practical, sample problems have been run by RSICC. Neither the United States Government, nor the Department of Energy, nor UT-BATTELLE, LLC, nor any person acting on behalf of the Department of Energy or UT-BATTELLE, LLC, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, usefulness or functioning of any information code/data and related material, 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, the Department of Energy, UT-BATTELLE, LLC, nor any person acting on behalf of the Department of Energy or UT-BATTELLE, LLC.

Distribution Notice: This code/data package is a part of the collections of the Radiation Safety Information Computational Center (RSICC) developed by various government and private organizations and contributed to RSICC for distribution. Any further distribution by any holder, unless otherwise specifically provided for is prohibited by the U.S. Department of Energy without the approval of RSICC, P.O. Box 2008, Oak Ridge, TN 37831-6002.

Documentation for CCC-831 Code Package

	<u>PAGE</u>
RSICC Computer Code Abstract.....	iii
“ADVANTG—An Automated Variance Reduction Parameter Generator,” ORNL/TM-2013/416 Rev. 1, Oak Ridge National Laboratory (2015)	1
“Exnihilo Documentation Release 5.4 (Dev)” (July 14, 2015).	110

RSICC CODE PACKAGE CCC-831

1. NAME AND TITLE

ADVANTG 3.0.5: AutomateD VAriance reducTion Generator

AUXILIARY PROGRAMS INCLUDED

Denovo: 3-D, parallel, multigroup discrete ordinates transport solver

DATA LIBRARIES

BUGLE-96 (DLC-185)

DABL69 (DLC-130)

SCALE-6.1 27n19g and 200n47g ENDF/B-VII libraries (CCC-785)

2. CONTRIBUTOR

Oak Ridge National Laboratory, Oak Ridge, Tennessee

3. CODING LANGUAGE AND COMPUTERS

C, C++, Fortran 90, and Python source code and executables for Linux, Mac OS X, and Windows Operating Systems.

4. NATURE OF PROBLEM SOLVED

ADVANTG is an automated tool for generating variance reduction parameters for fixed-source continuous-energy Monte Carlo simulations with MCNP5-v1.6 (CCC-810, not included in this distribution) based on approximate 3-D multigroup discrete ordinates adjoint transport solutions generated by Denovo (included in this distribution). The variance reduction parameters generated by ADVANTG consist of space and energy-dependent weight-window bounds and biased source distributions, which are output in formats that can be directly used with unmodified versions of MCNP5. ADVANTG has been applied to neutron, photon, and coupled neutron-photon simulations of real-world radiation detection and shielding scenarios. ADVANTG is compatible with all MCNP5 geometry features and can be used to accelerate cell tallies (F4, F6, F8), surface tallies (F1 and F2), point-detector tallies (F5), and Cartesian mesh tallies (FMESH).

5. METHOD OF SOLUTION

ADVANTG implements the Consistent Adjoint Driven Importance Sampling (CADIS) method and the Forward-Weighted CADIS (FW-CADIS) method for generating variance reduction parameters. The CADIS and FW-CADIS methods provide a prescription for generating space- and energy-dependent weight-window targets and a consistent biased source distribution. The CADIS method was developed for accelerating individual tallies, whereas FW-CADIS can be applied to multiple tallies and mesh tallies. The CADIS method has been demonstrated to provide speed-ups in the tally FOM of $O(10^1-10^4)$ across a broad range of radiation detection and shielding problems. The FW-CADIS method has been shown to produce relatively uniform statistical uncertainties across multiple cell tallies and large space- and energy-dependent mesh tallies in real-world applications.

Denovo implements a structured, Cartesian-grid discrete ordinates solver based on the Koch-Baker-Alcouffe algorithm for parallel sweeps across x - y domain blocks. Multiple discretization schemes are available: step characteristics, linear-discontinuous, tri-linear discontinuous and diamond difference (optionally theta-weighted or with negative-flux fixup). Multiple quadrature sets are available: QR product, QR triangular, Gauss-Legendre product, linear-discontinuous finite element, level-symmetric, as well as user-defined quadratures. Denovo contains two embedded first-collision source treatments: an analytic kernel for point sources and a Monte Carlo treatment for distributed sources. The Trilinos parallel solvers package is used to apply GMRES to accelerate the within-group iterations, resulting in a computationally efficient and robust transport solver.

The references provide a detailed description of the CADIS and FW-CADIS methods, as well as the methods and algorithms implemented in the Denovo discrete ordinates package.

6. RESTRICTIONS OR LIMITATIONS

The implementations of the CADIS and FW-CADIS methods in ADVANTG are based on the use of scalar flux estimates from Denovo calculations. As a result, no directional biasing, in either the weight-window parameters or the biased source distributions, is currently implemented.

7. TYPICAL RUNNING TIME

The run time consumed by ADVANTG is problem-dependent. The majority of computational time is consumed by the discrete ordinates solver, so the primary factors that affect the run time are: the size of the deterministic spatial grid, the physics of the problem (e.g., photon-only versus coupled neutron-photon, presence of upscatter, etc.), the number of quadrature directions, and the number of energy groups.

ADVANTG can drive either serial (i.e., single-processor) or parallel Denovo calculations. For serial calculations on modern desktops, typical run times vary from several minutes to several hours. The run time can be significantly reduced by executing the Denovo calculations in parallel.

8. COMPUTER HARDWARE REQUIREMENTS

ADVANTG and Denovo are operable on modern x86-64 platforms.

9. COMPUTER SOFTWARE REQUIREMENTS

The included ADVANTG and Denovo executables run on 64-bit Linux, Mac OS X, and Windows operating systems.

On all platforms, ADVANTG requires an MCNP5-1.60 executable from package CCC-740 and the associated data libraries (neither are included in this distribution). Any of the pre-built i386 or x86-64 MCNP5-1.60 executables from CCC-740 will work. Executables from other versions of MCNP (e.g., MCNP6-1.0) and all MCNPX executables will not work.

On Mac OS X machines, the included ADVANTG binary distribution requires a system installation of Python, which can be obtained by downloading XCode for free from the App Store, then running `xcode-select --install` from the terminal.

For Windows machines, a 64-bit installation of Python-2.7.x is required (Python-2.7.10 is recommended). Microsoft MPI v5 or later is recommended to enable running Denovo calculations in parallel.

The included executables were created using the following compilers and open-source packages:

Linux:	GCC-4.8.4 with HDF5-1.8.15-patch1, LAPACK-3.5.0, OpenMPI-1.8.6, Python-2.7.10, Silo-4.10.2-BSD, SWIG-2.0.12, and Trilinos-12.0.1
Mac OS X:	GCC-4.9.1 with HDF5-1.8.15, OpenMPI-1.8.4, Silo-4.10.2-BSD, SWIG-3.0.2, and Trilinos-casl-dev
Windows:	GCC-4.9.2 with HDF5-1.8.15-patch1, LAPACK-3.5.0, Microsoft MPI-5.0.12435.6, Python 2.7.10, Silo-4.10.2-BSD, SWIG-2.0.12, and Trilinos-12.0.1

ADVANTG and Denovo binaries can be built from source code on Linux, Mac OS X and Windows operating systems using the open-source GNU Compiler Collection (GCC). Denovo additionally supports the Intel and PGI C++ and Fortran compilers.

The open-source packages listed below are required to build ADVANTG and Denovo source code. Note that the version number specifies the *minimum* version required. An exception is that only Python-2.x (with $x \geq 7$) is supported. Python-3.x is not currently supported.

CMake 2.8.11	http://www.cmake.org/
SWIG 2.0.12	http://www.swig.org/
Python 2.7	https://www.python.org/
LAPACK 3.0	http://www.netlib.org/lapack/
Trilinos 12.0	https://trilinos.org/
Silo 4.10	https://wci.llnl.gov/simulation/computer-codes/silo/

Building a parallel version of Denovo requires an MPI (Message Passing Interface) library. For Linux and Mac OS X, OpenMPI (<http://www.open-mpi.org/>) is recommended. For Windows, Microsoft MPI (<https://msdn.microsoft.com/en-us/library/bb524831.aspx>) is recommended.

Both ADVANTG and Denovo output Silo-format files that can be read by the open-source VisIt 3-D, parallel visualization tool (<https://wci.llnl.gov/simulation/computer-codes/visit/>). Use of VisIt to inspect the quality of the deterministic solutions before starting Monte Carlo runs is highly recommended.

10. REFERENCES

- S.W. Mosher et al., “ADVANTG—An Automated Variance Reduction Parameter Generator,” ORNL/TM-2013/416 Rev. 1, Oak Ridge National Laboratory (2015).
- J. C. Wagner and A. Haghight, “Automated Variance Reduction of Monte Carlo Shielding Calculations Using the Discrete Ordinates Adjoint Function,” *Nuclear Science and Engineering*, **128**, 186 (1998).

- T. M. Evans, A. S. Stafford, R. N. Slaybaugh, and K. T. Clarno, “Denovo: A New Three-Dimensional Parallel Discrete Ordinates Code in SCALE,” *Nuclear Technology*, **171**, 171–200 (2010).
- J. C. Wagner, D. E. Peplow, and S. W. Mosher, “FW-CADIS Method for Global and Semi-Global Variance Reduction of Monte Carlo Radiation Transport Calculations,” *Nuclear Science and Engineering*, **176**, 37–57 (2014).

11. CONTENTS OF CODE PACKAGE

The DVD contains ADVANTG and Denovo executables and source code for Linux, Mac OS X, and Windows systems, documentation, and BUGLE-96, DABL69, SCALE 27n19g, and SCALE 200n47g multigroup cross section libraries in ANISN format.

12. DATE OF ABSTRACT

August 2015

KEYWORDS: VARIANCE REDUCTION; DISCRETE ORDINATES; HYBRID TRANSPORT; CADIS; FW-CADIS; MONTE CARLO; MCNP

ADVANTG—An Automated Variance Reduction Parameter Generator



Approved for public release.
Distribution is unlimited.

S. W. Mosher
S. R. Johnson
A. M. Bevill
A. M. Ibrahim
C. R. Daily
T. M. Evans
J. C. Wagner
J. O. Johnson
R. E. Grove

August 2015

DOCUMENT AVAILABILITY

Reports produced after January 1, 1996, are generally available free via 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
5285 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.gov
Website <http://www.ntis.gov/help/ordermethods.aspx>

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

Office of Scientific and Technical Information
PO 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.

Reactor and Nuclear Systems Division

**ADVANTG—AN AUTOMATED VARIANCE REDUCTION
PARAMETER GENERATOR**

Scott W. Mosher
Seth R. Johnson
Aaron M. Bevill
Ahmad M. Ibrahim
Charles R. Daily
Thomas M. Evans
John C. Wagner
Jeffrey O. Johnson
Robert E. Grove

Date Published: August 2015

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

CONTENTS

LIST OF FIGURES	v
LIST OF TABLES.....	vii
DEFINITIONS, ACRONYMS, AND ABBREVIATIONS	ix
1. INTRODUCTION	1
2. METHODS	3
2.1 CADIS METHODOLOGY	3
2.2 MULTIPLE TALLIES.....	4
2.2.1 CADIS Method.....	5
2.2.2 Cooper and Larsen Method.....	5
2.3 FW-CADIS METHOD	6
2.3.1 Path-Length Weighting.....	6
2.3.2 Global Weighting.....	7
2.3.3 Response Weighting	8
2.3.4 Spectral Tallies	8
3. IMPLEMENTATION.....	9
3.1 COMPUTATIONAL SEQUENCES	9
3.2 MULTIGROUP CROSS SECTION LIBRARIES	9
3.3 MATERIAL COMPOSITION MAPPING.....	11
3.4 MATERIAL REGION MAPPING.....	11
3.5 SOURCE MAPPING.....	15
3.6 TALLY REGION MAPPING	17
3.7 DETERMINISTIC CALCULATIONS	18
3.8 VARIANCE REDUCTION PARAMETER GENERATION	19
4. RUNNING ADVANTG	21
5. INPUT.....	23
5.1 INPUT FILE FORMAT.....	23
5.2 DRIVER OPTIONS.....	24
5.3 MODEL OPTIONS	25
5.3.1 MCNP-Specific Options	25
5.3.2 SWORD-Specific Options	27
5.4 METHOD OPTIONS.....	28
5.4.1 FW-CADIS-Specific Options	29
5.4.2 DX-Specific Options.....	30
5.5 SPATIAL MESH.....	30
5.6 MULTIGROUP LIBRARY OPTIONS	32
5.7 SOLVER OPTIONS	32
5.8 OUTPUT OPTIONS.....	40
5.8.1 MCNP-Specific Options	40
5.8.2 Silo-Specific Options	42
6. OUTPUT.....	43

7.	EXAMPLES	45
7.1	UEKI SHIELDING EXPERIMENTS	45
7.1.1	Background.....	45
7.1.2	Objective.....	45
7.1.3	MCNP Model and Results	47
7.1.4	ADVANTG Calculations.....	49
7.1.5	Results.....	54
7.2	SIMPLIFIED PORTAL MONITOR	58
7.2.1	Background.....	58
7.2.2	Objective.....	58
7.2.3	ADVANTG Calculations.....	59
7.2.4	Results.....	63
7.3	JAPAN POWER DEMONSTRATION REACTOR	66
7.3.1	Background.....	66
7.3.2	Objective.....	66
7.3.3	MCNP Model and Results	67
7.3.4	ADVANTG Calculations.....	70
7.3.5	Results.....	76
8.	TIPS AND SUGGESTIONS	79
8.1	SCOPING SIMULATIONS	79
8.2	MCNP INPUT PREPARATION	80
8.3	DISCRETE ORDINATES CALCULATIONS	81
8.4	ADVANTG CALCULATIONS	81
8.5	MCNP SIMULATIONS	82
8.6	TALLY CONVERGENCE.....	83
9.	LIMITATIONS.....	85
9.1	SUPPORTED MCNP SDEF DISTRIBUTION TYPES	85
9.2	SUPPORTED MCNP TALLIES	85
9.3	LIMITATIONS IN DENOVO.....	86
9.4	LIMITATIONS IN MCNP5	86
9.4.1	Biasing Continuous Distributions	86
9.4.2	Pulse-Height Tallies.....	87
10.	REFERENCES	89
	APPENDIX A. MULTIGROUP ENERGY BOUNDS	A-1
	APPENDIX B. INDEX OF FREQUENTLY USED KEYWORDS.....	B-1

LIST OF FIGURES

Fig. 3-1. Geometry of problem INP12 with modified source	12
Fig. 3-2. Ray tracing in the $+x$ direction through problem INP12	13
Fig. 3-3. Mixed material distribution of problem INP12	14
Fig. 3-4. ITER 40° sector model (left) and unfolded model (right)	14
Fig. 3-5. Source regions in problem INP12	16
Fig. 3-6. Response regions in problem INP12	17
Fig. 3-7. Adjoint scalar flux in problem INP12	18
Fig. 3-8. Weight-window targets in problem INP12	20
Fig. 4-1. Sample PBS script	22
Fig. 7-1. Schematic arrangement of source, shields, and detector (Fig. 2 from Ueki et al.)	46
Fig. 7-2. Measured dose attenuation factors (Fig. 3 from Ueki et al.)	46
Fig. 7-3. MCNP model of Ueki experiment for the $T = 35$ cm case	47
Fig. 7-4. ADVANTG input for the Ueki experiment, $T = 35$ cm case	49
Fig. 7-5. Discretized material map for the Ueki problem	50
Fig. 7-6. User changes to SDEF cards for the Ueki problem	51
Fig. 7-7. Total adjoint flux for the Ueki problem, $T = 5$ cm case	52
Fig. 7-8. Total adjoint flux for the Ueki problem, $T = 35$ cm case	52
Fig. 7-9. ADVANTG changes to MCNP input for Ueki problem, $T = 35$ cm case	53
Fig. 7-10. Ratio of biased to unbiased source bin probabilities, $T = 35$ cm case	54
Fig. 7-11. Simplified portal problem	58
Fig. 7-12. Source location in portal problem	59
Fig. 7-13. ADVANTG input for portal problem	61
Fig. 7-14. Denovo forward (left) and adjoint (right) total fluxes for the portal problem	62
Fig. 7-15. ADVANTG changes to SDEF cards for the portal problem	62
Fig. 7-16. Pulse height spectra for the portal problem	64
Fig. 7-17. Pulse-height relative uncertainties for the portal problem	64
Fig. 7-18. Fraction of tally bins with less than a given uncertainty for the portal problem	65
Fig. 7-19. JPDR reactor enclosure and specifications (Fig. 1 and Table A.1 from Sukegawa et al.)	66
Fig. 7-20. JPDR 2-D (r, z) benchmark geometry (Figs. A.5[1] - [3] from Sukegawa et al.)	68
Fig. 7-21. JPDR MCNP model	69
Fig. 7-22. MCNP total flux (left) and relative uncertainty (right) for the JPDR problem	69
Fig. 7-23. ADVANTG input for the JPDR problem	70
Fig. 7-24. Discretized material map for the JPDR problem	71
Fig. 7-25. User changes to SDEF cards for the JPDR problem	72
Fig. 7-26. Denovo forward (left) and adjoint (right) total flux for JPDR problem	73
Fig. 7-27. ADVANTG changes to SDEF cards for the JPDR problem	74
Fig. 7-28. ADVANTG-based total flux (left) and relative uncertainty (right) for the JPDR problem	76
Fig. 7-29. Fraction of tally bins with less than a given uncertainty for the JPDR problem	77

LIST OF TABLES

Table 3-1. Multigroup libraries.....	9
Table 3-2. Computational steps	10
Table 3-3. Computational sequences	10
Table 4-1. Command line options.....	21
Table 6-1. Problem sub-directories.....	44
Table 7-1. MCNP5 dose rate results for the Ueki problem.....	48
Table 7-2. MCNP5 attenuation factor results for the Ueki problem.....	48
Table 7-3. ADVANTG-based dose rate results for the Ueki problem.....	56
Table 7-4. ADVANTG-based attenuation factor results for the Ueki problem	56
Table 7-5. Comparison of ADVANTG and MCNP5 F4 tally results.....	57
Table 7-6. Comparison of ADVANTG and MCNP5 F5 tally results.....	57
Table 7-7. ADVANTG source energy biasing.....	63
Table 7-8. Distribution of differences for the simplified portal problem.....	65
Table 7-9. ADVANTG radial biasing for the JPDR problem.....	75
Table 7-10. ADVANTG axial biasing for the JPDR problem.....	75
Table 9-1. Supported MCNP5 tally types and tally modifiers.....	86
Table A-1. 27n19g library groups.....	A-1
Table A-2. 200n47g library groups.....	A-2
Table A-3. BUGLE-96 and BPLUS library groups	A-4
Table A-4. DABL69 and DPLUS library groups	A-5
Table A-5. FENDL67 library groups.....	A-6

DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

ADVANTG	AutomateD VAriaNce reducTion Generator
CADIS	Consistent Adjoint Driven Importance Sampling, a method for generating variance reduction parameters to accelerate the estimation of an individual tally.
Denovo	3-D, block-parallel discrete ordinates transport code developed at Oak Ridge National Laboratory.
deterministic	Methods (e.g., the discrete ordinates method) and codes that discretize the independent variables of the transport equation and solve the resulting linear algebraic systems of equations via iterative methods.
FW-CADIS	Forward-Weighted CADIS, a method for generating variance reduction parameters to obtain relatively uniform statistical uncertainties across multiple tally regions or energy bins.
FOM	Tally figure of merit, calculated as $1/(R^2 T)$, where R is the tally relative error and T is the simulation run time in minutes.
GL	Gauss-Legendre, a type of product quadrature.
GMRES	Generalized Minimum RESidual, a Krylov subspace method for iteratively solving linear algebraic systems of equations.
LD	Linear discontinuous, a discretization scheme that expands the angular flux within a voxel in terms of a volume-average value and a slope in the x , y , and z dimensions.
LDFE	Linear-discontinuous finite element, a type of triangular quadrature.
MCNP	Monte Carlo N-Particle. A continuous-energy Monte Carlo transport code developed at Los Alamos National Laboratory.
Monte Carlo	Methods and codes that simulate particle transport by stochastically sampling individual particle events (e.g., emission from source, free-streaming between collisions, collision kinematics) and tallying the average behavior.
PBS	Portable Batch System, a job scheduling system used in Linux cluster environments
Python	Open-source scripting language (https://www.python.org/).
QR	Quadruple range, a type of product quadrature.
SDEF	MCNP fixed-source specification
Silo	Open-source mesh and field library and scientific database originally developed at Lawrence Livermore National Laboratory (https://wci.llnl.gov/simulation/computer-codes/silo).
S_N	discrete ordinates
SWORD	SoftWare for Optimization of Radiation Detectors, a graphical user interface and framework for constructing and evaluating radiation detection systems developed by the U.S. Naval Research Laboratory. The software is distributed through RSICC as package CCC-767.

TLD	Trilinear discontinuous, a discretization scheme that is similar to LD, but is based on an angular flux expansion that also includes the xy , yz , xz , and xyz cross terms.
Trilinos	Collection of open-source software packages developed at Sandia National Laboratory (https://trilinos.org/). Includes packages for solving linear systems of equations using modern iterative methods.
VisIt	Open-source 3-D, parallel visualization tool originally developed at Lawrence Livermore National Laboratory (https://wci.llnl.gov/simulation/computer-codes/visit/).
WWINP	MCNP weight-window input file
ZAID	MCNP nuclide identification number. ZAIDs are written in the form ZZZAAA, where ZZZ is a one- to three-digit atomic number and AAA is a three-digit mass number.

1. INTRODUCTION

The Automated Variance Reduction Generator (ADVANTG) software automates the process of generating variance reduction parameters for continuous-energy Monte Carlo simulations of fixed-source neutron, photon, and coupled neutron-photon transport problems using MCNP5 (X-5 Monte Carlo Team 2003). ADVANTG generates space- and energy-dependent mesh-based weight-window bounds and biased source distributions using three-dimensional (3-D) discrete ordinates (S_N) solutions of the adjoint transport equation that are calculated by the Denovo package (Evans et al. 2010). ADVANTG outputs weight-window lower bounds as an MCNP-compatible weight-window input (WWINP) file. Weight-window control parameters and biased source distributions are output as **WWP** and **SB** cards, respectively, in an extended version of the user's original MCNP input file.

ADVANTG also provides the capability to execute discrete ordinates calculations without generating variance reduction parameters. ADVANTG extracts problem geometry, material composition, source, and tally information from either an MCNP5 model or from a model created using the SWORD software (Novikova et al. 2006). From this information, ADVANTG constructs a discretized representation of the transport problem on a user-provided spatial grid and given multigroup cross section library. ADVANTG can automatically launch Denovo to execute the discrete ordinates calculation, possibly using multiple cores and/or processors (e.g., on multi-core desktop systems and clusters). The discretized models and discrete ordinates solutions can be visualized using the open-source VisIt 3-D visualization software (Childs et al. 2005).

The primary objective of ADVANTG is to reduce both the user effort and the computational time required to obtain accurate and precise tally estimates across a broad range of challenging transport applications. ADVANTG has been applied to simulations of real-world radiation shielding, detection, and neutron activation problems. Examples of shielding applications include material damage and dose rate analyses of the Oak Ridge National Laboratory (ORNL) Spallation Neutron Source and High Flux Isotope Reactor (Risner and Blakeman 2013) and the ITER Tokamak (Ibrahim et al. 2011). ADVANTG has been applied to a suite of radiation detection, safeguards, and special nuclear material movement detection test problems (Shaver et al. 2011). ADVANTG has also been used in the prediction of activation rates within light water reactor facilities (Pantelias and Mosher 2013). In these projects, ADVANTG was demonstrated to significantly increase the tally figure of merit (FOM) relative to an analog MCNP simulation. The ADVANTG-generated parameters were also shown to be more effective than manually generated geometry splitting parameters.

ADVANTG provides a powerful, efficient, and fully automated alternative to traditional methods for generating variance reduction parameters. Because ADVANTG employs a deterministic transport solver, no extra effort is required on the part of the user to generate weight-window parameters that span the entire problem domain. For challenging problems, multiple iterations may be required to refine the deterministic spatial mesh, quadrature set, or other computational options to obtain high-quality variance reduction parameters. This process can generally be accomplished in much less time overall and with less effort than using a stochastic weight-window generator or manually developing cell importances.

The variance reduction generator methods implemented in ADVANTG are described in Section 2. Implementation of the methods is summarized in Section 3. Running ADVANTG from the command line is the subject of Section 4. Input and output are described in Sections 5 and 6, respectively. Several example problems are described in detail in Section 7. Tips and suggestions for using ADVANTG to obtain accurate and precise MCNP tally estimates are discussed in Section 8. Finally, known limitations are listed in Section 9.

2. METHODS

ADVANTG implements the Consistent Adjoint Driven Importance Sampling (CADIS) method (Wagner and Haghishat 1998) and the Forward-Weighted CADIS (FW-CADIS) method (Wagner et al. 2014) for generating variance reduction parameters. The CADIS and FW-CADIS methods provide a prescription for generating space- and energy-dependent weight-window targets and a consistent biased source distribution. The CADIS method was developed for accelerating individual tallies, whereas FW-CADIS was developed for multiple tallies and mesh tallies. The CADIS method has been demonstrated to provide speed-ups in the tally FOM of $O(10^1\text{-}10^4)$ across a broad range of radiation detection and shielding problems. The FW-CADIS method has been shown to produce relatively uniform statistical uncertainties across multiple cell tallies and large space- and energy-dependent mesh tallies in real-world applications (Wagner et al. 2010).

2.1 CADIS METHODOLOGY

The CADIS method was developed for transport problems in which a single scalar quantity is to be estimated. Consider the fixed-source transport equation

$$\mathbf{H}\psi(\vec{\mathbf{r}}, \hat{\boldsymbol{\Omega}}, E) = q, \quad (2-1)$$

where \mathbf{H} is the transport operator, q is the known source distribution, ψ is the unknown angular flux density, and boundary conditions are given. It is assumed that the quantity of interest can be written as the integral

$$R = \langle \sigma_d, \psi \rangle, \quad (2-2)$$

where the angle brackets denote integration over all phase-space variables. In Eq. (2-2), σ_d denotes an arbitrary response function, for example, a detector cross section or flux-to-dose-rate conversion factor.

Associated with Eqs. (2-1) and (2-2) is the adjoint transport equation

$$\mathbf{H}^+\psi^+ = \sigma_d, \quad (2-3)$$

where \mathbf{H}^+ is the adjoint transport operator and ψ^+ is the adjoint flux density. The adjoint transport operator is related to the forward operator by

$$\langle \psi, \mathbf{H}^+\psi^+ \rangle = \langle \psi^+, \mathbf{H}\psi \rangle, \quad (2-4)$$

and thus the response can also be written in terms of the adjoint flux

$$R = \langle q, \psi^+ \rangle. \quad (2-5)$$

The boundary conditions of Eq. (2-3) are chosen to be identical to the forward conditions, though they apply to the opposite directional half-space (i.e., to outgoing as opposed to incoming directions).

The solution of Eq. (2-3) can be interpreted as an importance function (Bell and Glasstone 1970). This can be understood by setting $q = \delta(\mathbf{P} - \mathbf{P}_0)$, where δ is the Dirac delta function and $\mathbf{P}_0 = (\vec{\mathbf{r}}_0, \hat{\boldsymbol{\Omega}}_0, E_0)$ denotes an arbitrary point in the problem phase-space. In this case, Eq. (2-4) reduces to

$$\psi^+(\mathbf{P}_0) = \int G(\mathbf{P}_0 \rightarrow \mathbf{P})\sigma_d(\mathbf{P})d\mathbf{P}, \quad (2-6)$$

where the forward solution is the Green's function G . This equation states that $\psi^+(\mathbf{P}_0)$ is the expected contribution to the response R from a unit-weight particle emitted at \mathbf{P}_0 . This property makes the adjoint function particularly useful in Monte Carlo simulations; it can be used to determine whether a particle's trajectory will carry it toward a region where it is relatively more or less likely to contribute to the tally of interest. For this reason, the solution of the adjoint transport equation is often referred to as an *importance function* or *importance map*.

The CADIS method is a recipe for calculating space- and energy-dependent weight-window targets and a consistent biased source distribution using an estimate of the adjoint function. (For a description of the weight-window variance reduction technique, see, for example, the MCNP Manual, Vol. I, Sec. 2.VII.B.6.) First, an importance map is generated by solving Eq. (2-3) and appropriate boundary conditions using a relatively inexpensive deterministic transport calculation. Weight targets are then computed in proportion to the inverse of the adjoint scalar flux

$$w(\mathbf{P}) = \frac{R}{\phi^+(\mathbf{P})}. \quad (2-7)$$

For the MCNP code, weight-window lower bounds are generated as

$$w_\ell(\mathbf{P}) = \frac{2}{(1+r)} \frac{R}{\phi^+(\mathbf{P})}, \quad (2-8)$$

where r is the ratio of the upper and lower weight bounds. A unique feature of the CADIS method is the use of a biased source distribution

$$\hat{q}(\mathbf{P}) = \frac{q(\mathbf{P})\phi^+(\mathbf{P})}{R}, \quad (2-9)$$

which ensures that source particles are preferentially sampled in regions of high importance. In addition, each source particle will start with a weight consistent with Eq. (2-7).

The variance reduction parameters in Eqs. (2-7) and (2-9) depend on the response value, R , which is the quantity to be estimated. If highly accurate response and adjoint flux estimates were required to produce useful variance reduction parameters, then this approach would be of little value. Fortunately, experience has shown that this is not the case. For some problems, even crude estimates can be used to generate effective variance reduction parameters.

2.2 MULTIPLE TALLIES

The CADIS method is an effective technique for estimating a single quantity of interest. In many problems, though, one desires to estimate multiple quantities. Consider a simulation in which estimates are sought for multiple responses with uniform statistical precision:

$$R_i = \langle \sigma_{d,i} \phi \rangle, \text{ for } i = 1, 2, \dots, N, \quad (2-10)$$

where both the $\sigma_{d,i}$ and N are arbitrary. Several approaches for generating variance reduction parameters that seek to satisfy this objective are described in the sections that follow.

2.2.1 CADIS Method

In some cases, the CADIS method can be effectively applied to estimate multiple tallies. A straightforward application of CADIS to a simulation with N different tallies would be to calculate N different adjoint solutions, generate N different sets of variance reduction parameters, and execute N different Monte Carlo simulations. This approach can be reasonable when N is small. For tallies with many cell or energy bins or for mesh tallies, though, this technique is generally inefficient.

A second approach would be to simply treat the sum of the responses as the response of interest in Eq. (2-2):

$$R = R_1 + R_2 + \cdots + R_N, \quad (2-11)$$

so that

$$q^+ = \sigma_{d,1} + \sigma_{d,2} + \cdots + \sigma_{d,N} = \sigma_d. \quad (2-12)$$

This technique can be very effective, for example, in problems where the tallies reside within the same vicinity of the problem domain. However, when it is applied to tallies located at significantly different distances from the source, the tally FOMs will generally differ greatly in magnitude. In many cases, the tally farthest from the source will have an FOM on par with an analog simulation. This is a consequence of the CADIS method's definition of importance as the expected contribution to the total response R . Relatively fewer contributions are made to tallies with relatively lower expected values.

A third technique can be effective when estimates are sought for tallies over concentric regions surrounding the source. In this case, defining the response of interest to be the tally in the outermost region will generally reduce the variance of all of the tallies. Of course, this occurs simply because particles must pass through the inner tally volumes to reach the outermost region.

In most cases, application of the CADIS method to simultaneously estimate multiple tallies will produce an undesirable amount of variation in tally FOMs. For these problems, a different approach to constructing the adjoint source is needed.

2.2.2 Cooper and Larsen Method

Cooper and Larsen (2001) developed a method for constructing weight windows for global transport problems, in which flux estimates are sought across the entire spatial domain. The authors showed that the flux density of Monte Carlo particles is related to the physical flux by

$$m(\mathbf{P}) = \frac{\phi(\mathbf{P})}{\bar{w}(\mathbf{P})}, \quad (2-13)$$

where \bar{w} is the mean particle weight at phase-space location \mathbf{P} . Thus, if the centers of the weight windows are chosen to be proportional to the forward flux, then an approximately uniform Monte Carlo particle flux (i.e., computational particle flux) will be obtained throughout the problem. While obtaining uniform Monte Carlo flux density is not exactly equivalent to obtaining uniform statistical uncertainties, the authors demonstrated that this choice for the weight window produced nearly uniform relative variances in numerical tests.

2.3 FW-CADIS METHOD

The FW-CADIS method was developed with an objective similar to that of the Cooper and Larsen method—that is, to generate variance reduction parameters for simultaneous estimation of multiple tallies with approximately uniform statistical precision. However, whereas Cooper and Larsen’s method was designed for global problems, the FW-CADIS method is intended to span the range from a few localized tallies to space- and energy-dependent mesh tallies that encompass the entire domain. This is accomplished by constructing an adjoint source that consists of appropriately weighted contributions from all tallies of interest. The weights are the inverses of the individual responses:

$$q^+ = \frac{1}{R_1} \sigma_{d,1} + \frac{1}{R_2} \sigma_{d,2} + \cdots + \frac{1}{R_N} \sigma_{d,N}. \quad (2-14)$$

Then the total response is a sum of equal-weight terms

$$R = \langle q^+, \phi \rangle = 1 + 1 + \cdots + 1. \quad (2-15)$$

Because the CADIS method defines importance as the expected contribution to the total response R , approximately the same number of contributions will be made to all tallies regardless of their expected values. Implicit in this argument is the assumption that every particle contributes to just a single tally. Though this assumption is often not strictly valid, relatively uniform uncertainties are obtained in most problems.

To construct the weighted adjoint source in Eq. (2-14), estimates of individual responses ($R_i, i = 1, 2, \dots, N$) are required. For this reason, applying the FW-CADIS method requires two deterministic calculations: an initial forward calculation to estimate the responses and an adjoint calculation to estimate the importance function resulting from the weighted adjoint source. The importance function is then used to construct weight-window bounds and a biased source distribution according to Eqs. (2-7) and (2-9), respectively. In essence, FW-CADIS is a recipe for constructing an adjoint source that can be used within the CADIS framework.

In the subsections that follow, Eq. (2-14) is used to derive adjoint sources for the common cases of interest. Spatial weighting options will be developed for problems with multiple cell-averaged tallies and mesh tallies. Two energy weighting options are also developed for estimating energy-integrated tallies or detailed energy spectra. These options can be used in different combinations to tailor the biasing parameters for a particular calculation.

2.3.1 Path-Length Weighting

Consider a problem in which estimates are sought for an arbitrary number of cell-averaged responses with approximately uniform statistical precision. Let the volume of the i^{th} cell be denoted by V_i . The i^{th} response to be estimated is then

$$R_i = \frac{1}{V_i} \int \sigma_i(E) \int_{V_i} \phi(\vec{r}, E) dV dE, \quad (2-16)$$

where σ_i is a tally multiplier—for example, an energy-dependent cross section, a flux-to-dose-rate conversion function, or just a constant. Using Eq. (2-10), we find that

$$\sigma_{d,i}(\vec{r}, E) = \frac{1}{V_i} f_i(\vec{r}) \sigma_i(E), \quad (2-17)$$

where f_i is the indicator function:

$$f_i(\vec{r}) = \begin{cases} 1, & \text{for } \vec{r} \in V_i, \\ 0, & \text{otherwise.} \end{cases} \quad (2-18)$$

Now using Eq. (2-14) we find that the adjoint source for the tally in the i^{th} cell is

$$q_i^+(\vec{r}, E) = \frac{f_i(\vec{r}) \sigma_i(E)}{\int \sigma_i(E') \int_{V_i} \phi(\vec{r}, E') dV dE'}. \quad (2-19)$$

As expected, the adjoint source density increases as the flux in the cell or the cell volume decreases. Moreover, the magnitude of σ_i has no impact on the importance function. This is appropriate, because all cell tallies in MCNP are track-length based, so tally multipliers (e.g., from an FM card) do not contribute to variance.

The spatial weighting in Eq. (2-19) is referred to as *path-length weighting* and is the default treatment in ADVANTG. It can be used with cell and surface tallies. (Note that surface tallies have an associated volume after being mapped onto the deterministic mesh.) It can also be used with mesh tallies, however the global weighting technique (described in the next subsection) is generally preferred. When path-length weighting is applied to mesh tallies, statistical uncertainties will generally be lowest/highest in mesh tally voxels that contribute the most/least to the volume-averaged response.

2.3.2 Global Weighting

Consider a problem in which the desire is to obtain approximately uniform statistical precision across a large tally volume (e.g., a mesh tally). Let the tally volume, V , be subdivided into N regions, each having volume $\Delta V = V/N$. If path-length weighting is applied to each of the regions using Eq. (2-19), then

$$q^+(\vec{r}, E) = \sum_{i=1}^N \frac{f_i(\vec{r}) \sigma_i(E)}{\int \sigma_i(E') \int_{\Delta V} \phi(\vec{r}, E') dV dE'}. \quad (2-20)$$

Because CADIS parameters are derived from the ratio $\phi^+/\langle \phi, q^+ \rangle$, multiplying the adjoint source by a constant has no effect on the final variance reduction parameters. If we divide the denominator of Eq. (2-20) by ΔV and then consider the limiting case for large N , we arrive at

$$q^+(\vec{r}, E) = \frac{f(\vec{r}) \sigma(E)}{\int \sigma(E') \phi(\vec{r}, E') dE'}, \quad (2-21)$$

where $f(\vec{r})$ is the indicator function for the mesh tally volume.

The spatial treatment in Eq. (2-21) is referred to as *global weighting*, though it can be applied to a mesh tally of any size. It must be explicitly turned on (see the description of the `fwcadis_spatial_treatment` card in Section 5.4.1). Because the adjoint source was developed based on equal-volume subdivisions of the mesh tally region, only the outer boundary of the tally region is considered when constructing the adjoint source (i.e., the locations of the interior mesh planes are ignored). Smaller than average voxels will tend to have larger than average statistical uncertainties, and vice versa.

2.3.3 Response Weighting

In the previous two subsections, energy-integrated tallies are considered. As a result, the adjoint sources shown in Eqs. (2-19) and (2-21) are normalized by the energy-integrated response. For historical reasons, this type of energy weighting is referred to as *response weighting*. This normalization is appropriate, for example, when estimating total fluxes, dose rates, and reaction rates. It can be used regardless of whether the tally region is a point, surface, or volume.

With response weighting, tally statistical uncertainties will generally be lowest at energies that contribute most strongly to the total response. Estimating energy-dependent tallies with approximately uniform precision across all energy bins is possible (as described in the next subsection), but is generally more computationally expensive than response weighting and is needed less often. For this reason, response weighting is the default energy treatment in ADVANTG.

2.3.4 Spectral Tallies

In problems where detailed spectral information is desired, response weighting can be turned off (see the `fwcadis_response_weighting` input keyword in Section 5.4.1). In the case of path-length weighting, the adjoint source becomes:

$$q_i^+(\vec{r}, E) = \frac{f_i(\vec{r})}{\int_{V_i} \phi(\vec{r}, E) dV}. \quad (2-22)$$

For global weighting, the adjoint source is

$$q^+(\vec{r}, E) = \frac{f(\vec{r})}{\phi(\vec{r}, E)}. \quad (2-23)$$

At a given point in space, the magnitudes of the adjoint sources shown above are relatively higher at energies where the flux is relatively lower in magnitude. In this way, energy-dependent tallies can be estimated with approximately uniform precision across all energy bins. This treatment generally results in a significant increase in the average number of splitting events per history, and thus an increase in computational time, relative to response weighting.

3. IMPLEMENTATION

This section discusses implementation of the methods described in Section 2. The objective here is not to provide a thorough and complete description of all algorithms implemented in ADVANTG. Instead, the basic operation of ADVANTG is discussed, along with the details of algorithms that affect the accuracy of the model discretization process and, by extension, the quality of the deterministic results and variance reduction parameters.

3.1 COMPUTATIONAL SEQUENCES

ADVANTG performs a series of computational steps to implement the CADIS and FW-CADIS methods. The steps are listed and briefly described in Table 3-2. Only certain steps are included in the execution of each method, as shown in Table 3-3. For future reference, the second column of Table 3-3 lists the input option that selects each sequence (see the description of the `method` input keyword in Section 5.2). ADVANTG provides a third sequence, shown in the last row of Table 3-3, which discretizes the problem geometry, source, and tallies, and outputs the discretized model for visualization and inspection purposes. This `dx` sequence can also be used to run a forward or adjoint Denovo discrete ordinates calculation without generating variance reduction parameters (see Section 5.4.2).

3.2 MULTIGROUP CROSS SECTION LIBRARIES

The discrete ordinates calculations performed by Denovo require multigroup cross sections. The ADVANTG distribution includes several ANISN-format coupled neutron-gamma cross section libraries, listed in Table 3-1. For future reference, the second column of the table lists the `anisn_library` input option used to select each library (see Section 5.6). The library energy group structures are given in Appendix A. No auxiliary codes are needed to use these libraries. ADVANTG can read ANISN-format libraries, mix cross sections, and output a working library for Denovo.

Table 3-1. Multigroup libraries

Library	anisn_library option	# of groups (N / G)	# of isotopes or elements	Evaluation	Reference
27n19g	27n19g	27 / 19	393	ENDF/B-VII.0	Wiarda et al. 2008
200n47g	200n47g	200 / 47	393	ENDF/B-VII.0	Wiarda et al. 2008
BUGLE-96	bugle96	47 / 20	120	ENDF/B-VI.3	White et al. 1995
BPLUS	bplus	47 / 20	393	ENDF/B-VII.0	N/A
DABL69	dabl69	46 / 23	80	ENDF/B-V	Ingersoll et al. 1989
DPLUS	dplus	46 / 23	393	ENDF/B-VII.0	N/A
FENDL67	fendl67	46 / 21	71	FENDL-2.1	López Aldama and Trkov, 2004

Table 3-2. Computational steps

Step	Tasks
A	<ul style="list-style-type: none"> • Read and check user input • Generate and read an MCNP runtpe file • Mix multigroup cross sections • Map material regions onto the deterministic spatial mesh
B	<ul style="list-style-type: none"> • Map tally regions onto mesh
C	<ul style="list-style-type: none"> • Map MCNP SDEF source onto mesh and energy groups
D	<ul style="list-style-type: none"> • Setup and execute forward Denovo calculation • Read forward Denovo flux solution
E	<ul style="list-style-type: none"> • Generate FW-CADIS adjoint source
F	<ul style="list-style-type: none"> • Generate CADIS adjoint sources from tallies
G	<ul style="list-style-type: none"> • Setup and execute adjoint Denovo calculation • Read adjoint Denovo flux solution
H	<ul style="list-style-type: none"> • Generate and write weight-window bounds • Estimate biased source probabilities • Write new MCNP input file with WWP and SB cards
I	<ul style="list-style-type: none"> • Write Silo output for visualization

Table 3-3. Computational sequences

Method or sequence	method option	Steps included								
		x = always included					• = optional			
		A	B	C	D	E	F	G	H	I
CADIS	<code>cadis</code>	x	x				x	x	x	•
FW-CADIS	<code>fwcadis</code>	x	x	x	x	x		x	x	•
Discretize-only or Denovo-only calculation	<code>dx</code>	x	•	•	•		•	•		•

The 27n19g and 200n47g libraries are general-purpose shielding libraries based on a weighting function that consists of a fission spectrum, a $1/E$ slowing down spectrum, and a Maxwellian distribution. The BUGLE-96 library was developed for light water reactor shielding and pressure vessel dosimetry applications. The broad-group cross sections were generated by collapsing the VITAMIN-B6 library using five different weighting spectra calculated from a 1-D model of a reactor cavity and bioshield. The DABL69 library was developed for use in defense-related radiation shielding applications. It was created by collapsing the VITAMIN-E library using a weighting function similar to the 200n47g library, but with an added 14 MeV fusion peak. The FENDL67 library was developed for fusion applications by collapsing the FENDL/MG-2.1 library (175 neutron and 42 gamma groups) to a broad-group structure.

The BPLUS and DPLUS libraries, developed by the ADVANTG team, are updated versions of the BUGLE-96 and DABL69 libraries, respectively. These libraries were generated using the same group structures and weighting spectra as their older counterparts, but they include all 393 isotopes in the ENDF-B/VII.0 evaluation data files. The BPLUS and DPLUS libraries have not yet been thoroughly validated, but they have been used to generate effective variance reduction parameters for many problems.

3.3 MATERIAL COMPOSITION MAPPING

For discrete ordinates calculations, a multigroup cross section working library must be generated based on the material compositions defined on the `m` cards in the MCNP input file. This task requires mapping MCNP nuclide identification numbers (ZAIDs) (e.g., 26056 for ^{56}Fe) to ANISN cross section table IDs. For this purpose, each of the multigroup libraries distributed with ADVANTG has an associated ZAID-to-index mapping file (`.zaid` file) that defines the default mapping. Users can override and/or add mappings using the `anisn_zaid_map` keyword (see Section 5.6).

The ZAID-to-index mapping process can require one or two steps. The ZAID is first located in the mapping database and, if an associated ANISN table exists, it is used immediately. If the ZAID was not found in the database, the next step will depend on the type of multigroup library being used:

- If the library is a coupled neutron-photon or neutron-only library and the ZAID refers to an element (e.g., 26000), then the element is expanded into its naturally occurring isotopes based on the abundances listed in Rosman and Taylor (1998). A search is then performed to find mappings for all of the isotopes. If the total abundance of isotopes that are not found in the database is less than 0.5%, then the abundances for the remaining isotopes are renormalized and used. If the fraction of missing isotopes is greater than 0.5%, the expansion process is aborted, and an error is generated.
- If the library is a photon-only library, then a search is made to find a ZAID with the same Z number (proton number). If one is found, its mapping is used. Otherwise, an error is generated.

In all other cases, an error message is generated. For convenience, ADVANTG attempts to find multigroup cross section tables for all materials before issuing an error message. If a mapping error does occur, all of the missing ZAIDs are listed in the message.

3.4 MATERIAL REGION MAPPING

The Denovo discrete ordinates code solves a discretized form of the transport equation on a structured rectangular mesh. The MCNP5 code uses a combinatorial geometry in which material cells are described as volumes bounded by several possible types of surfaces (planes, cylinders, spheres, cones, etc.). A

fundamental task in creating an approximate representation of the Monte Carlo model for Denovo is to map the combinatorial geometry onto a user-specified structured grid.

To illustrate the material mapping process implemented in ADVANTG, consider test problem INP12 from McKinney and Iverson (1996), shown in Fig. 3-1. This problem models an oil-well logging scenario in which an iron sonde containing two ^3He detectors and a neutron source is inserted into a water-filled borehole within a limestone formation. (The geometry of the problem is divided into mesh-like cells to utilize cell-based weight-window parameters from a diffusion code. The input file for this problem was created before mesh-based weight windows were available in MCNP.) For demonstration purposes, the original point source defined in the problem was changed to a volume source that fills the source cell.

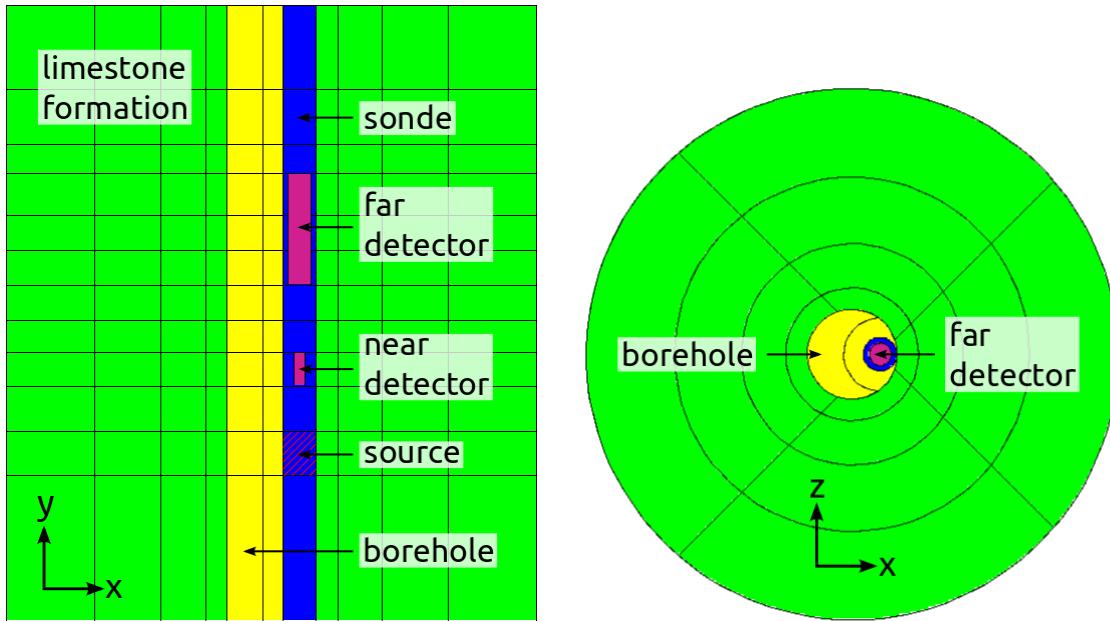


Fig. 3-1. Geometry of problem INP12 with modified source.

ADVANTG performs material region mapping by tracing rays through the MCNP geometry model. The starting points of rays are randomly sampled on the exterior $-x$, $-y$, and $-z$ faces of the mesh. Rays are then traced in the $+x$, $+y$, and $+z$ directions to the opposite side of the mesh from the starting location. As each ray is traced, tallies record the track length each ray traverses through each material within each voxel of the structured grid. This process is illustrated in Fig. 3-2, which depicts two rays that started from the same external voxel face and were traced in the $+x$ direction through the borehole region of problem INP12. In the figure, the gray dashed lines represent mesh boundaries, and the blue crosses denote locations where the rays cross a mesh plane, and the red crosses indicate where the rays cross an MCNP cell surface. In the center voxel, for example, the ray with greater z -coordinate contributes three track-length scores for three different materials, whereas the other ray contributes only a single score.

Once the ray tracing is completed, the track-length tallies are used to estimate the volume fractions of materials within each voxel. The material fractions are then used to generate a mixed-material specification for the Denovo calculation. Materials with volume fractions that differ by less than a tolerance value are combined to minimize the total number of mixed materials generated. The tolerance

can be decreased to obtain more accurate material maps in problems where the response is sensitive to the volume of one or more materials.

The material maps generated by ADVANTG can be displayed using the VisIt visualization software. The material map generated for problem INP12 on an $81 \times 57 \times 81$ voxel nonuniform mesh is shown in Fig. 3-3. In the figure, two images are displayed, both of which are clipped at $z = 0$ to show the geometry at the center of the borehole region. In the left image, distinct material interfaces were reconstructed by VisIt before displaying the plot. (VisIt applies this treatment by default.) The right image uses a *clean zones only* option, which displays voxels with mixed materials as white. No interface reconstruction is performed in either ADVANTG or Denovo, so it is important to consider the impact that material mixing will have on the discrete ordinates calculation. For this reason, examining both types of material plots is recommended.

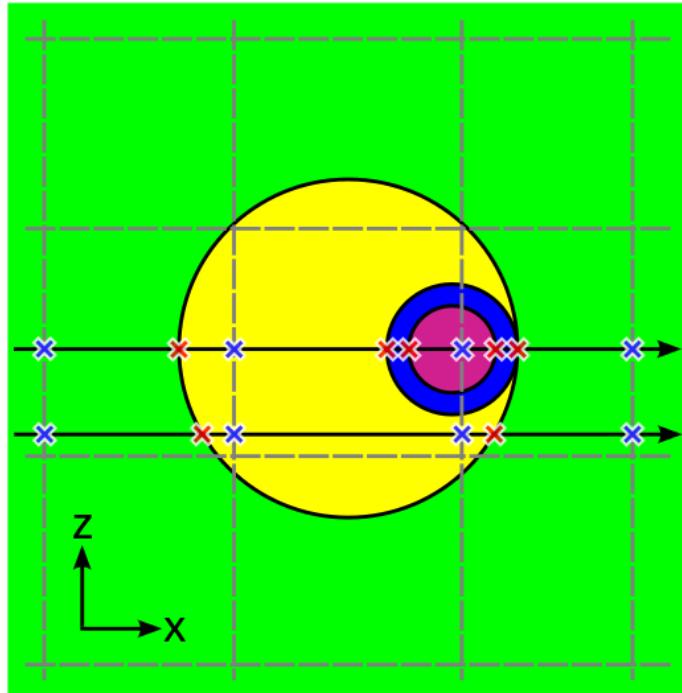


Fig. 3-2. Ray tracing in the $+x$ direction through problem INP12.

By default, ADVANTG will trace a minimum of ten rays from each external voxel face on the $-x$, $-y$, and $-z$ edges of the mesh. Increasing the number of rays will produce more accurate material volume fractions and thus a more accurate deterministic calculation. (See Section 5.3.1 for a description of the ray tracer settings.) The computational time consumed by the ray tracing is proportional to the number of rays. The average cost of tracing a single ray can vary greatly between MCNP models. Ray tracing is generally more expensive in models that contain many complicated cells (i.e., cells that are defined using the union or complement operators).

Denovo supports specularly reflective boundary conditions, but only on boundaries that are perpendicular to one of the coordinate axes. ADVANTG provides the capability to unfold reflected geometries that cannot be modeled directly by Denovo (see Section 5.3.1 for the associated input options). For example, many MCNP models of the ITER Tokamak consist of a sector of the reactor with reflective boundary

conditions on the external azimuthal surfaces. Fig. 3-4 shows the discretized geometries generated for a 40° sector ITER model with and without the unfolding option (Ibrahim et al. 2011). Note that for certain geometries, it may be necessary to extend the boundaries of the model in order for the unfolding to work properly. In all cases, careful inspection of the unfolded geometry in VisIt is highly recommended.

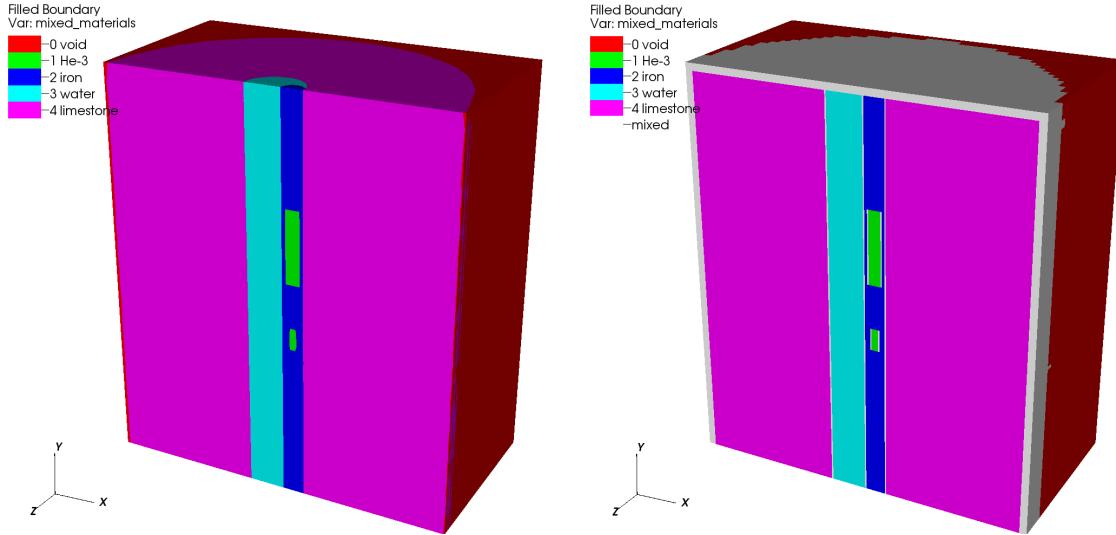


Fig. 3-3. Mixed material distribution of problem INP12. The left image was generated using the default material interface reconstruction technique, and the right image was generated using *clean zones only* option.

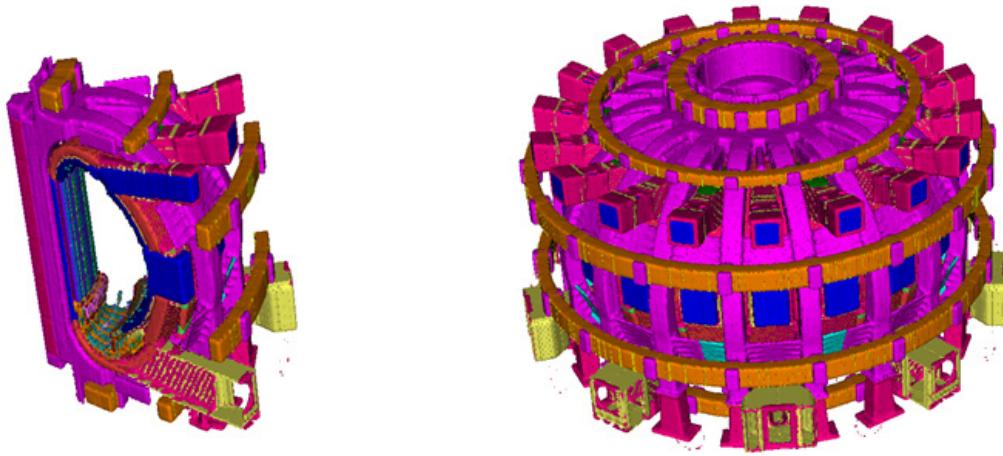


Fig. 3-4. ITER 40° sector model (left) and unfolded model (right).

3.5 SOURCE MAPPING

When setting up a forward discrete ordinates calculation, ADVANTG maps the fixed source defined in an MCNP input (via the SDEF card) into a form acceptable to Denovo. The mapped source consists of the following data:

- List of multigroup energy spectra
- List of any point sources, including the location, strength, and spectrum index
- Array of volume source intensities for each mesh voxel
- Array of spectrum indices for each mesh voxel

This source specification may contain point sources, volume sources, or both. Also, the same spectrum can be used for multiple points or voxels.

Point sources are treated differently than volume sources in Denovo. This is because localized sources tend to cause ray effects in discrete ordinates solutions, especially when the source is located within an optically thin or strongly absorbing medium. Ray effects due to localized sources can be mitigated, and in many cases nearly eliminated, by replacing point and small volume sources with the corresponding first-collided scattering source, which is distributed throughout the problem domain. For point sources, Denovo automatically generates an energy-dependent first-collided source based on an uncollided flux distribution that it calculates using an analytic point-kernel approach. The total flux is then calculated as the sum of the uncollided flux and the converged S_N solution for the first-collided source. By default, Denovo does not apply a first-collision treatment to volume sources; however, a Monte Carlo-based approach to estimating the first-collision source for a volume source can be activated if desired. (See Section 5.7 for a description of Denovo options.)

ADVANTG provides an option to treat a volume source as a point source in the discrete ordinates calculations, and it is generally recommended to do so whenever the source is very small relative to the size of the problem domain. The location of the source point is determined by sampling the source and averaging the sampled positions. If the source is uniform in volume, the point source will be located at the centroid of the source region with some offset due to sampling error. (See Section 5.3.1 for a description of the MCNP source settings.)

In MCNP, SDEF sources can be specified so as to contain a mix of volume and point sources. For simplicity, ADVANTG treats an SDEF source as either a volume source or a collection of one or more point sources. Though there are several ways of constructing a distribution of point sources using the SDEF cards, ADVANTG will detect only the following cases:

- If none of the POS, X, Y, Z, RAD, or EXT variables appear on the SDEF card, the source is treated as a point source located at the origin, as in MCNP.
- If neither the RAD nor EXT variables appears on the SDEF card and the X, Y, and Z variables are given as explicit or default values, the source is treated as a point source located at (X, Y, Z).
- If POS is an explicit value or an L, S, DS L, or DS S distribution and none of the X, Y, Z, RAD, or EXT variables is listed on the SDEF card, the source is treated as the collection of points defined in the POS distribution or in its child distributions.

In all other cases, ADVANTG will treat the source as a volume source.

ADVANTG passes the locations of point sources directly to Denovo. Volume sources, on the other hand, must first be mapped onto the voxels of the spatial mesh. This is done by sampling source particles and then tallying the particle weight in the voxel in which the source point is located. The spatial distribution of the source density is then estimated as the average weight in each voxel. In this mapping process, the number of source samples to be simulated is determined based on three user input parameters: the minimum and maximum number of samples and a target minimum average number of samples per mesh voxel containing source. The intent of the target is to provide a relatively problem-independent means for obtaining consistently sampled volume sources. Because the spatial extent of the source is not known *a priori*, the number of samples is determined during sampling. Initially, the minimum number of samples is drawn. The number of samples is then divided by the number of voxels that have been found to contain source. If this number is greater than or equal to the target density, no additional samples are taken. Otherwise, another sample is drawn, and the stopping criterion is re-evaluated. Sampling proceeds until either the target is met or the maximum number of samples is reached, in which case a warning message is printed. The spatial distribution of volume sources generated by ADVANTG can be displayed in VisIt. The source region generated for problem INP12 is shown in Fig. 3-5.

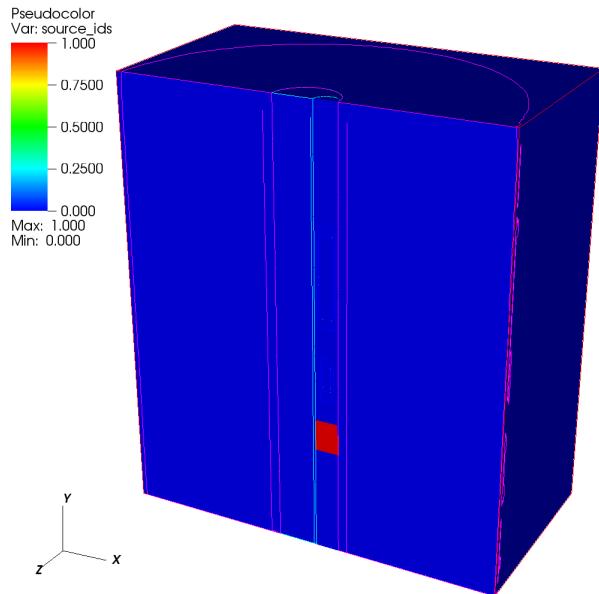


Fig. 3-5. Source regions in problem INP12.

ADVANTG maps SDEF energy distributions onto the group structure of the multigroup cross section library using analytic integration. All distribution types are supported: discrete lines (**SI L**), histograms (**SI H**), piecewise-linear densities (**SI A**), built-in functions (**SP -f**), as well as distributions of arbitrary sets of distributions (**SI S**). If more than one energy distribution is defined and the source is separable in space and energy, ADVANTG collapses the distributions into a single spectrum. If the source is not separable, a multigroup spectrum is generated for each distribution and then mapped to spatial elements later. Though ADVANTG is capable of mapping all types of distributions, the use of the **DS H**, **DS L**, and **DS T** cards for the **ERG** variable is not currently supported because this would require partitioning the energy distribution bins in a spatially dependent manner. For the same reason, if **ERG** is an independent variable that is depended upon by another variable, it must be specified as an **S** distribution.

If the source is separable in space and energy, each source point or voxel will have the same spectrum. If the source is not separable, the association between points or voxels and spectra must be determined. For point sources, this is done by looking at the dependencies between distributions. For volume sources, the number of samples that have appeared from each energy distribution is tracked for each voxel. If a point or voxel is found to contain a mix of spectra, the already-mapped spectra are combined based on their relative contributions, as determined from the probabilities defined in the SDEF distributions (for point sources) or based on the relative number of observed samples (for volume sources).

3.6 TALLY REGION MAPPING

To construct adjoint source distributions for the Denovo S_N calculations, ADVANTG maps surface, cell, and mesh tallies onto the user-specified spatial grid. Tally region mapping is done simultaneously with material region mapping to avoid ray-tracing through the geometry a second time. The discretized tally regions generated by ADVANTG can be displayed in VisIt, as shown in Fig. 3-6 for problem INP12. In the figure, the tally IDs correspond to the order in which they are listed in the ADVANTG input file. In this example, the near detector was listed before the far detector. ADVANTG supports cell tallies with multiple cell bins and tallies on repeated structures and lattices. Tally regions may be missed by the ray tracer if they are small relative to the mesh voxels in which they reside. If this occurs, the mesh should be refined and/or the number of rays should be increased.

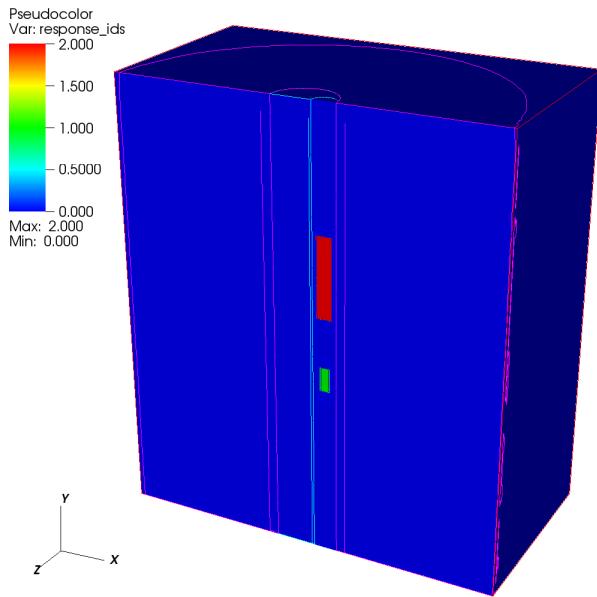


Fig. 3-6. Response regions in problem INP12.

From the perspective of the CADIS and FW-CADIS methods, all tallies have an associated energy spectrum. For example, the DE/DF cards define the response spectrum for tallies that have associated dose functions. As with forward source spectra, ADVANTG maps the response spectrum onto the multigroup structure using analytic integration. ADVANTG supports DE/DF, FM, and EM multipliers. Though MCNP allows a tally to have more than one type of multiplier, ADVANTG will select and use only a single type of multiplier. The order of precedence is DE/DF, FM, then EM. For example, if a tally has DE/DF cards, any FM or EM cards will be ignored. If no multiplier is defined, a uniform spectrum is used. For pulse-height

(F8) tallies, ADVANTG uses the total cross section of the material in the tally cell as the response spectrum. This treatment has been found to be effective for most problems involving pulse-height tallies.

3.7 DETERMINISTIC CALCULATIONS

ADVANTG obtains deterministic transport solutions by preparing inputs for and executing the Denovo 3-D, block-parallel discrete ordinates package. Denovo was selected for this purpose because it provides a powerful, robust, and efficient general deterministic transport capability. It is built around modern linear algebra solvers provided by the Trilinos library (Heroux et al. 2003) and uses the Generalized Minimum RESidual (GMRES) solver to converge within-group (scattering) iterations. This Krylov subspace solver has been shown to significantly outperform source iteration (Richardson iteration) in problems with thick scattering media. Denovo also implements several spatial discretization schemes, quadrature sets, and both analytic point-kernel and Monte Carlo-based first collision sources. For large-scale problems, the transport sweeps can be performed in parallel on multiple processors. Denovo has been used to solve several problems with more than one billion spatial cells.

Denovo provides multiple front ends for input. The primary and full-featured front end is implemented as a Python (scripting language) module. ADVANTG drives Denovo through this Python interface, which provides access to multiple spatial discretization schemes (step characteristics, linear discontinuous, trilinear discontinuous, and weighted diamond difference), multiple quadrature sets (quadruple range, LDDE, Gauss-Legendre product, and level symmetric), and the ability to perform parallel calculations.

Both group-wise and energy-integrated scalar flux solutions generated by Denovo can be displayed in VisIt. Fig. 3-7 shows the discretized material map (left) and the energy-integrated adjoint scalar flux (right) from a CADIS calculation where the far detector was the response of interest. The clip operator was used to arrange the material and flux plots next to each other. Contours were also plotted and overlaid on the pseudocolor flux plot. The near detector does not show up in the material map because the CADIS calculation was performed on a relatively coarse mesh.

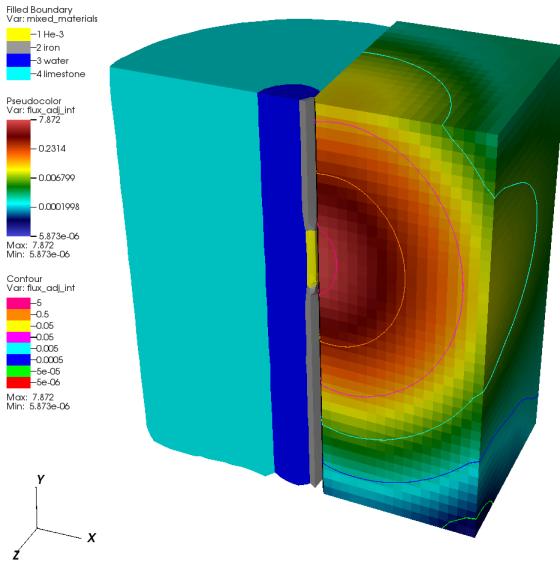


Fig. 3-7. Adjoint scalar flux in problem INP12.

It is strongly recommended to carefully examine the group-wise scalar flux solution(s) to determine if there are any significant nonphysical features (e.g., ray effects or negative fluxes). If necessary, the deterministic calculation(s) should be refined and the variance reduction parameters regenerated before proceeding to the Monte Carlo simulation.

3.8 VARIANCE REDUCTION PARAMETER GENERATION

ADVANTG provides computational sequences that generate variance reduction parameters using the CADIS and FW-CADIS methods. The parameters consist of space- and energy-dependent weight-window targets and a biased source that is consistent with the weight map. The total response, R , from Eq. (2-5) and the biased source probabilities, \hat{q} , from Eq. (2-9) are estimated by sampling the original (unbiased) source distribution and scoring the importance evaluated at the source particle's location and energy. The total response is the average score per source particle. The biased source probabilities are calculated as the average scores within the bins defined by the SI cards in the MCNP input file. Once the total response is known, weight-window targets are computed according to Eq. (2-8).

By default, ADVANTG automatically generates biased probabilities and outputs SB cards for all SDEF distributions that can be effectively biased with a space- and energy-dependent importance function. In most cases, probabilities will be generated only for spatial and energy distributions. However, if a variable (e.g., DIR) is depended upon by a spatial variable or the energy variable (ERG), biased probabilities will be generated for the independent variable as well.

In most problems, it is advantageous to partition the SDEF distribution bins in order to capture the spatial and spectral variations of the importance function in the biased source. Section 7 provides examples of how to generate effective source biasing parameters for several sample problems. Note that it is possible to recalculate the biased source probabilities after changing the SDEF distribution bins without re-running the discrete ordinates calculation(s). To do that, use the -f option when starting ADVANTG. (See Section 4 for a description of the ADVANTG command line options.)

When spatial source biasing is used with cell or cookie-cutter rejection, the mean weight of a source particle is in general no longer equal to one and the starting particle weight (WGT parameter of the SDEF card) must be corrected to preserve tally mean values. Because rejection is commonly used, an approach for estimating the corrected starting weight and its uncertainty has been developed and implemented in ADVANTG. The implementation is currently limited to the case in which rejection is applied using a single cell; rejection over multiple cells is not yet supported. For the single-cell case, ADVANTG calculates and outputs the corrected WGT value on a replacement SDEF card and its relative error in an adjacent comment card. Because the corrected starting weight directly multiplies all tally results, a warning message is printed if the relative uncertainty in the WGT estimate is greater than 1%. The uncertainty can be reduced by increasing the number of samples on the mcnp_num_wgt_samples card, which is described in Section 5.8.1. As with source biasing parameters, the WGT parameter can be recalculated without re-running the discrete ordinates calculation(s).

ADVANTG assumes that rejection is possible whenever the CEL or CCC keyword appears on the SDEF card. If it is truly impossible for a sampled source location to ever be rejected, then no correction to the starting particle weight is needed. However, ADVANTG will still estimate a corrected WGT parameter because it cannot generally know when rejection is impossible. In this case, ADVANTG will estimate a corrected WGT that is identical to the original starting weight with zero estimated relative error.

When cell or cookie-cutter rejection is possible, the corrected WGT parameter depends on the biased probabilities of the spatial variables as well as the biased probabilities of any variables on which the

spatial variables depend. For this reason, changing or removing the biased probabilities may invalidate the WGT parameter estimated by ADVANTG and lead to incorrectly normalized MCNP tally results.

Note that it is possible to disable the generation of biasing parameters for spatial variables, for the energy variable, or altogether. (See Section 5.8.1 for a description of the source biasing settings.) If rejection is possible but SB card generation is disabled for spatial variables, no weight correction is needed. In this case, ADVANTG will not estimate a corrected SDEF WGT parameter.

The WWINP file created by ADVANTG contains weight-window lower bounds calculated using Eq. (2-8) with R set to one. The value of R normalized to a unit source strength is set as the WNORM value on the corresponding WWP card(s). (WNORM is a lower-bound multiplier; it is the seventh parameter on the WWP card.) If the SDEF card in the MCNP input file specifies an initial particle weight other than one, then the WNORM value is multiplied by this modified starting weight.

Group-wise and energy-integrated weight-window targets can be displayed in VisIt, as shown in Fig. 3-8. The figure is analogous to Fig. 3-7, but it shows weight-window targets instead of importances. When the weight-window map is used in an MCNP simulation, particles moving toward the detector will undergo splitting, whereas particles moving away from the detector will be subjected to roulette.

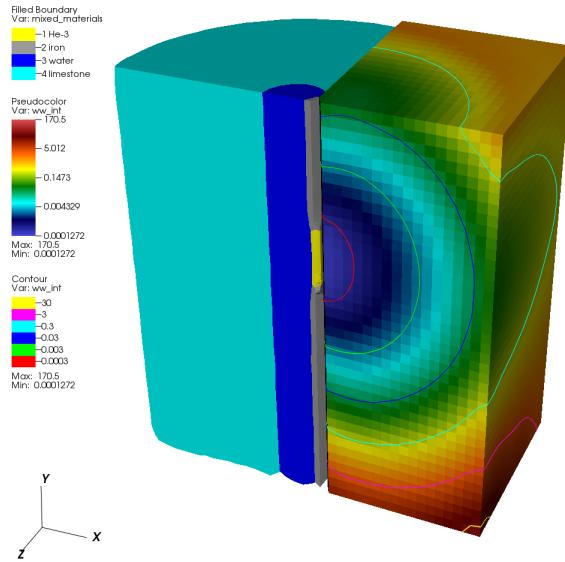


Fig. 3-8. Weight-window targets in problem INP12.

4. RUNNING ADVANTG

ADVANTG is executed from the command line using:

```
$ advantg [options] input_file
```

Command line options are listed in Table 4-1 below. The format and content of the input file is the subject of Section 5.

Table 4-1. Command line options

Option	Description
-h, --help	Print command line help and exit
--version	Print the code version and exit
--template [args]	Print a listing of the ADVANTG input keywords and associated descriptions. Arguments filter the keywords that are displayed.
-c, --clean	Ignore existing working subdirectories. The default is to reuse existing data unless input has changed.
-f, --force-resume	Force existing working data to be reused, even if it appears to be outdated
-g, --debug	Write extra information to disk for debugging purposes
-v, --verbose	Print all debug messages
-q, --quiet	Print only informational and warning messages
--very-quiet	Print only warning messages
--silent	Print messages only on failure
--log=LEVEL	Create a log file with the given verbosity level: DEBUG, STATUS, INFO, WARNING, ERROR, or CRITICAL

The current version of ADVANTG executes only in serial mode—that is, using a single thread/core. However, ADVANTG will invoke `mpiexec` to launch a parallel Denovo job when the product of `denovo_x_blocks` and `denovo_y_blocks` (the number of parallel domains) is greater than one. ADVANTG passes the number of domains as the argument to the `mpiexec -np` option. In addition, if a file named `machinefile` is found in the problem directory (the directory in which ADVANTG was launched), then ADVANTG will add a `-machinefile` option to the `mpiexec` command with the path of the file.

ADVANTG can drive parallel Denovo calculations on clusters with job submission systems. Consider a problem in which a Denovo will be executed across 36 domains. An example Portable Batch System (PBS) script is shown in Fig. 4-1 for a cluster with two hex-core processors per node. A PBS directive requests 3 nodes and 12 processes per node. Other directives set the amount of memory per process (1 GB), a 1-hour wall-clock limit, and a job name. The script sets the working directory and dumps the machinefile created by the batch system for this particular job before launching ADVANTG. When the job is started, ADVANTG begins running on a single core. When ADVANTG is ready to perform a

deterministic calculation, it invokes `mpieexec` to execute a parallel Denovo calculation using 36 cores across the three nodes that were reserved for the job.

```
#!/bin/bash
#PBS -N advantg
#PBS -l nodes=3:ppn=12
#PBS -l pmem=1gb
#PBS -l walltime=1:00:00
cd $HOME/calcs/case1/advantg
cat $PBS_NODEFILE > machinefile
advantg case1.adv
```

Fig. 4-1. Sample PBS script.

5. INPUT

ADVANTG reads user input from a free-format plaintext input file. The format of the input file is described in Section 5.1 below. The sections that follow describe the various input options that ADVANTG accepts. A typical input file will contain only a small fraction of the entries described in this section. Default values are defined for most options and only a few entries are required. For MCNP models, a minimal input consists of:

- The name of an MCNP input file,
- A list of one or more tally numbers,
- The name of a generator method (e.g., `cadis` or `fwcadis`),
- Lists of spatial mesh planes in the x , y , and z dimensions, and
- The name of a multigroup cross-section library.

ADVANTG extracts all model-related information (geometry, materials, sources, and tallies) automatically. As a result, input files tend to consist primarily of parameters for the deterministic calculation(s).

For new users, a recommended first step is to look at the example problems in Section 7. ADVANTG inputs are shown in Figs. 7-4, 7-13, and 7-23. This section can be referred to later for more detailed information. An index of frequently used keywords is given in Appendix B.

5.1 INPUT FILE FORMAT

ADVANTG input files are plaintext files organized into a series of entries or cards. Each entry begins with a case-insensitive keyword that must start within the first four columns of a line. The keyword must be followed by at least one value. Multiple values are separated by whitespace. Values can be one of the following types:

- Integer literals (e.g., 0, 123, -5)
- Floating point literals (e.g., 0.5, -1.2345e+6)
- Boolean literals (e.g., `true`, `False`, 0, 1, Y, n, T, f)
- Strings (e.g., `gmres`, "file name")

Strings containing spaces must be enclosed in double quotes ("").

Entries can span multiple lines. Continuation lines are denoted by an indent of at least four columns. The length of input lines is not limited. Empty lines are ignored. Tab characters are expanded to eight spaces. With the exception of denoting a continuation line, multiple whitespace characters are treated the same as a single space.

Comments are denoted by the hash (#) character. All characters from the # character to the end of the line are ignored. Comments can start anywhere on a line and can be placed anywhere within an entry after the keyword.

5.2 DRIVER OPTIONS

The following high-level options determine which components are used to carry out the computational sequence:

model name (default: mcnp)

Type of transport problem model. One of:

mcnp	MCNP5 model
sword	SWORD model

method name (required)

Variance reduction method or type of computational sequence. One of:

cadis	CADIS method (single-tally variance reduction)
fwcadis	FW-CADIS method (multiple-tally variance reduction)
dx	Discretize the transport model and optionally execute deterministic calculation(s)

The `dx` option can be used to generate visualization output for the discretized model (see the `outputs` keyword below). This feature provides a means to quickly inspect the spatial mesh as well as the mapped materials, sources, and tallies, for example, as a preliminary step before proceeding to perform the transport calculations.

The `dx` option can also be used to execute a forward or adjoint deterministic calculation without generating variance reduction parameters. This feature is useful, for example, for performing scoping or feasibility studies or other calculations where continuous-energy Monte Carlo is not needed.

outputs name ... (default: mcnp silo)

Type(s) of output. Any of:

mcnp	MCNP weight-window file (WWINP) and biased source cards for the <code>cadis</code> or <code>fwcadis</code> methods
silo	Silo-format visualization files
response	Calculate, tabulate, and/or plot deterministic response estimates
none	No specialized output

The data within Silo-format files can be visualized using the VisIt open-source, 3-D, parallel visualization package. VisIt is an extremely powerful tool for exploring mesh-based data sets, and its use is highly recommended.

If the `response` option is selected, energy-dependent and energy-integrated responses will be calculated based on the deterministic transport solution. This option is most useful with the `dx` method.

5.3 MODEL OPTIONS

5.3.1 MCNP-Specific Options

The options and settings described in this section apply when `model` is `mcnp`. If a different model type is selected, these settings are ignored.

mcnp_input **filename** (required)

Filename of the MCNP5 input file. Enclose the filename in double quotes if it contains spaces.

mcnp_tallies int ...

MCNP tally number(s) for which to generate variance reduction parameters and/or to use in forming adjoint sources. This keyword is required if a deterministic adjoint calculation will be performed.

mcnp_material_names int name ...

Names of MCNP materials for output purposes, listed as pairs of MCNP `m` card numbers and name strings. If the `silo` output option is selected, the material names will be written to the Silo file and will appear in VisIt. By default, material 0 is named `void`.

Example:

```
mcnp_material_names 1 Air  
10 He-3  
11 "Stainless steel"
```

The following parameters control the number of samples drawn from the MCNP SDEF source when mapping it onto the spatial mesh. Note that these parameters do not affect the calculation of biased source distributions. (Instead, see the keywords described in Section 5.8.1).

`mcnp_min_source_samples` `int >= 0` (default: `1e6`)
`mcnp_max_source_samples` “ (default: `1e8`)

Minimum and maximum number of times to sample the SDEF source.

mcnp_target_source_density int >= 0 (default: 100)

Target minimum average number of source samples per mesh voxel containing source. If this target is not reached after `mcnp_min_source_samples` have been drawn, up to `mcnp_max_source_samples` will be generated in an attempt to reach this target.

A warning message will be printed if the target is not reached.

mcnp_max_point_sources `int >= 0` (**default:** 20)

Maximum number of point sources. If the SDEF source consists of more than this number of point sources, it will be mapped as a volume source for the deterministic calculation.

Each point source increases the run time and memory consumption of the Denovo calculation. This option exists to avoid executing a deterministic calculation that may have an unexpectedly large memory requirement or a long run time due to the number of first-collision sources.

mcnp_force_point_source `bool` (**default:** False)

Treat the source as a single point source in the deterministic calculation?

It is generally recommended to treat a small volume source (relative to the size of the problem domain) as a point source in the discrete ordinates calculation in order to activate a first-collision source treatment. This option provides a means to do this without modifying the original SDEF cards.

The following parameters control the number and direction of rays traced through the MCNP geometry to map materials and tally regions onto the spatial mesh.

mcnp_min_rays_per_face `int > 0` (**default:** 10)

Minimum number of rays to trace through each voxel face in each trace direction.

mcnp_ray_directions `axis ...` (**default:** x y z)

Directions in which to trace rays through the MCNP geometry model. Rays are traced parallel to the spatial axes and in the positive direction. Any combination of x, y, and/or z can be selected.

mcnp_num_rays `int > 0` (**default:** 1)

Nominal total number of rays.

The nominal number of rays to be traced through each voxel face is calculated in proportion to the area of the face. The per-face minimum is then applied so that the actual number of rays is the greater of the nominal number and the **mcnp_min_rays_per_face** setting (10 rays, by default).

By default, the **mcnp_min_rays_per_face** setting determines the total number of rays that are actually sampled. The **mcnp_num_rays** setting is provided because it can be useful in problems where the spatial mesh voxels vary greatly in size.

mcnp_lost_rays `int` (**default:** 10)

Maximum number of lost rays allowed while mapping material and tally regions onto the spatial mesh. An error message is printed if the limit is reached.

mcnp_mix_tolerance `real >= 0.0` **(default: 0.01)**

Tolerance for combining mixed materials. Materials with component volume fractions that differ by less than or equal to this amount will be combined to limit the total number of mixed materials generated. Higher numbers reduce the amount of memory needed to store material mixtures. Lower numbers provide more accurate per-voxel discretized material compositions.

For example, if voxels A, B, and C contain 70, 90, and 91% steel by volume, respectively, and 30, 10, and 9% air, then by default, voxels B and C will contain the same mixed material. Voxel A will contain a different mixture.

The following parameters control the unfolding options that can be used if the MCNP5 model has reflecting boundaries.

mcnp_unfolding_origin `real(3)`

Specifies the x , y , and z coordinates of a reference point inside a reflected MCNP geometry that is to be unfolded for the Denovo deterministic calculations. Avoid placing the reference point near external surfaces.

Note that the rays used to perform the unfolding are terminated when they reach a cell of zero importance. For certain geometries, it may be necessary to extend the model with one or more voided cells that have nonzero importance in order for the unfolding to work properly. In all cases, careful inspection of the unfolded geometry in VisIt is highly recommended.

mcnp_unfolding_safe `bool` **(default: False)**

If `True`, activates a “safe” ray tracing mode for the unfolding process. This safe mode is slower than the default mode, but the probability of generating lost rays is lower. Use the safe mode only if there are issues with lost rays when using the default mode.

5.3.2 SWORD-Specific Options

The options and settings described in this section apply when `model` is set to `sword`. If a different model type is selected, these settings are ignored.

sword_input `filename` **(required)**

Filename of the `.sword` file. Enclose the filename in double quotes if it contains spaces.

sword_mix_tolerance `real` **(default: 0.01)**

Tolerance for combining mixed materials (analogous to `mcnp_mix_tolerance`). Materials with component volume fractions that differ by less than or equal to this amount will be combined to limit the total number of mixed materials generated. Higher numbers reduce the amount of memory needed to store material mixtures. Lower numbers provide more accurate per-cell discretized material compositions.

sword_small_sources `bool` (**default:** True)

If `True`, single-voxel sources are approximated as a point source of equivalent strength to activate the first-collision source treatment in Denovo. Generally, this substitution yields more accurate deterministic results.

sword_sampling `name` (**default:** subcell)

Sampling algorithm to be used for discretizing the SWORD geometry. One of:

- `subcell` Specify the number of samples per cell
 - `res` Specify the sampling resolution
-

sword_subcell `int` (**default:** 2)
sword_subcell_x "
sword_subcell_y "
sword_subcell_z "

When `sword_sampling` is `subcell`, the `sword_subcell` keyword sets the number of point samples per interval in each spatial dimension. The dimension-specific keywords can be used to override this setting for a particular dimension.

For example, when `sword_subcell` is 1, material compositions are based on the material found at the center of each voxel. When the value is 2, eight samples per voxel are used to generate a mixed-material description of the contents of the voxel.

sword_resolution `float`

When `sword_sampling` is `res`, sets the length, in cm, of the desired point sampling interval for each spatial dimension. For example, if `sword_resolution` is set to 2.0, then in a voxel with dimensions $10 \times 6 \times 4$ cm, 5 point samples will be taken along the x dimension, 3 along y , and 2 along z , for a total of 30 samples within the voxel.

5.4 METHOD OPTIONS

No CADIS-specific input parameters are currently defined.

5.4.1 FW-CADIS-Specific Options

fwcadis_spatial_treatment `name` (default: `pathlength`)

Spatial treatment used when constructing the FW-CADIS adjoint source. One of:

- `pathlength` Use path-length weighting
- `global` Use global weighting

The `pathlength` treatment is recommended for problems with multiple cell, surface, and/or point detector tallies. The `global` treatment is recommended for mesh tallies. See Sections 2.3.1 and 2.3.2 for more details.

fwcadis_response_weighting `bool` (default: `True`)

If `True`, construct the adjoint source to accelerate an energy-integrated response (e.g., dose or total flux). If `False`, generate parameters with the objective of obtaining relatively uniform uncertainties across all energy groups. See Sections 2.3.3 and 2.3.4 for more details.

Note that turning off response weighting generally increases the time per history in the Monte Carlo simulation, especially if the group-wise flux in the region of interest varies over many orders of magnitude.

fwcadis_min_response `real >= 0.` (default: `0`)

fwcadis_max_response “ (default: `infinite`)

Lower and upper energy-integrated response bounds for including voxels when constructing the FW-CADIS adjoint source with the `global` spatial treatment. By default, no bounds are used.

This feature can be useful, for example, when estimating dose rate maps. In practical applications, there is generally little interest in dose rates that fall below a certain level. By excluding the adjoint source from the region outside of a particular dose rate contour, a more efficient Monte Carlo simulation is obtained.

Note that the locations of the upper and lower bounds are calculated based on the deterministic solution, which might not be highly accurate. It is therefore recommended to set the lower/upper bounds somewhat lower/higher than the actual contour of interest (e.g., by one or two orders of magnitude). To ensure that the thresholds are properly chosen, the responses calculated using the deterministic and MCNP models should be compared. If the difference between these responses is larger than the margin applied to the bound values, those values should be adjusted and the ADVANTG calculation should be re-run.

5.4.2 DX-Specific Options

The `dx` option provides a means to discretize the model and optionally execute deterministic calculations without generating variance reduction parameters. The options described below control which deterministic calculations are performed, if any.

```
dx_adjoint  bool  (default: False)
dx_forward   "      "
```

If `True`, executes a deterministic calculation in the specified transport mode. By default, no deterministic calculations are executed; the model is discretized, and the quality of the result can be visualized using the appropriate output options (see Section 5.8.2).

5.5 SPATIAL MESH

The following settings are used to construct a 3-D rectangular spatial mesh as the product of 1-D meshes in the x , y , and z dimensions. By default, the mesh of the weight-window map is the same as that used in the deterministic calculations.

```
mesh_refinement  name  (default: mcnp)
```

Technique for refining the mesh. One of:

<code>mcnp</code>	Use the <code>mesh_x_ints</code> , <code>mesh_y_ints</code> , and <code>mesh_z_ints</code> keywords
<code>uniform</code>	Use the <code>mesh_max_width</code> and <code>mesh_min_width</code> keywords

The `mcnp` option uniformly subdivides each mesh interval. The `uniform` option adds and/or removes voxel edges in an attempt to generate nearly uniform widths across each dimension.

```
mesh_x  real ...  (required)
mesh_y    "
mesh_z    "
```

Coordinates of mesh voxel edges in the x , y , and z dimensions, in cm. At least two edges per dimension are required. The edge coordinates must be listed in increasing order.

The mesh can be refined using one of two approaches described with the `mesh_refinement` keyword. By default, the mesh intervals are not subdivided further and the `mesh_x`, `mesh_y`, and `mesh_z` cards define the final mesh.

```
mesh_x_ints  int > 0 ...  (default: 1 subdivision per mesh interval)
mesh_y_ints    "
mesh_z_ints    "
```

The number of uniform subdivisions in each mesh interval when `mesh_refinement` is `mcnp`. The number is ignored if `mesh_refinement` is not `mcnp`. By default, the original mesh intervals are not subdivided further. The number of entries must be one less than for `mesh_x`, `mesh_y`, or `mesh_z`.

Example:

```
mesh_x      0  10  20  
mesh_x_ints 2   4
```

These cards define a spatial grid in the *x*-dimension that has voxel edges at the coordinates: 0, 5, 10, 12.5, 15, 17.5, and 20 cm.

```
mesh_max_width    real > 0.  
mesh_x_max_width  "  
mesh_y_max_width  "  
mesh_z_max_width  "
```

Maximum distance between voxel edges, in cm, when `mesh_refinement` is `uniform`. This is required if `mesh_refinement` is `uniform` and ignored otherwise. The `mesh_max_width` keyword sets the default maximum width for all dimensions. The dimension-specific keywords can be used to override this setting for a particular dimension.

```
mesh_min_width    real >= 0. (default: 0.0)  
mesh_x_min_width  "          "  
mesh_y_min_width  "          "  
mesh_z_min_width  "          "
```

Minimum distance in cm between voxel edges when `mesh_refinement` is `uniform`. This is ignored if `mesh_refinement` is not `uniform`. The `mesh_min_width` keyword sets the default minimum width for all dimensions. The dimension-specific keywords can be used to override this setting for a particular dimension.

By default, the minimum voxel width is zero, and none of the grid points specified on the `mesh_x`, `mesh_y`, or `mesh_z` cards will be removed.

Example:

```
mesh_x      -10  -5.5  0  0.25  5.5  10  
mesh_min_width  0.75  
mesh_max_width  2
```

These input cards will generate an *x*-dimension spatial grid that has voxel edges at: -10, -8.5, -7, -5.5, -3.67, -1.83, 0, 1.83, 3.67, 5.5, 7.0, 8.5, and 10 cm.

Note that the original edge at 0.25 cm has been removed because `mesh_min_width` is 0.75 cm. This can be avoided by setting the minimum width to 0.25 cm or smaller.

5.6 MULTIGROUP LIBRARY OPTIONS

ADVANTG reads multigroup cross section data from ANISN-format libraries. The options described in this section can be used to select a cross section library, set the upscatter treatment, and customize mappings of ZAIDs to cross section tables.

anisn_library `name` (required)

Name of the ANISN-format cross section library (see Section 3.2 for a list of library names). The search path defined at configuration time is searched to locate the library and associated metadata files.

anisn_upscatter `bool` (default: False)

If `False`, then treat upscatter as self-scatter and avoid costly upscatter iterations in the Denovo calculation(s). If the accuracy of thermal neutron fluxes is important, then consider setting `anisn_upscatter` to `True`. This option has no impact on photon-only problems.

anisn_zaid_map `int > 0 ...`

Additional ZAID and ANISN table ID pairs to use when mapping the MCNP material compositions to multigroup materials for the deterministic calculation. An arbitrary number of pairs can be given. Entries on this card add to or override the mappings defined in the `.zaid` file associated with the selected ANISN library.

Example:

```
anisn_library    27n19g
anisn_zaid_map  1001   1
                  1002   7
```

In the 27n19g library, nuclides 1001 (H-1) and 1002 (H-2) are mapped to cross section tables for bound hydrogen in H₂O (table 2395) and D₂O (table 2383), respectively. In this example, these defaults are overridden to map 1001 and 1002 to cross section for unbound H-1 (table 1) and H-2 (table 7). Note that this example is intended for illustrative purposes only. Only in rare cases is it desirable to use unbound hydrogen cross sections.

5.7 SOLVER OPTIONS

The Denovo discrete ordinates solver provides a variety of options and settings. For generating variance reduction parameters, typically only a fraction of these settings are used or modified. The most frequently used parameters are:

- `denovo_quadrature` and associated options, which select and configure the S_N angular approximation,
- `denovo_pn_order`, which determines the degree of scattering anisotropy modeled, and
- `denovo_x_blocks` and `denovo_y_blocks`, which determine the parallel spatial decomposition (if Denovo was built with an MPI library).

The amount of memory used by the Denovo solver state (i.e., by the discrete ordinates solution in memory) is

$$\text{state size} = N_v(N_g + N_K)(L + 1)^2 N_u \cdot 8 \text{ bytes}, \quad (5-1)$$

where N_v is the number of mesh voxels, N_g is the number of solved energy groups, N_K is the number of Krylov vectors (default is 20), and L is the scattering expansion order. N_u is the number of unknowns per voxel and is determined by the spatial discretization scheme (see below). The factor of eight is because the solver state is stored in double precision. First-collision and FW-CADIS adjoint sources may consume significant amounts of additional memory. In addition, a small overhead (for communication buffers, local-to-global maps, etc.) is added for each process when Denovo is executed in parallel. Thus, Eq. (5-1) represents a lower limit, not an upper bound. For cluster jobs, the number of nodes and/or the number of cores per node should be selected so that the memory usage per node does not exceed the available memory (with an allowance for the operating system and any other running jobs).

The following options and settings control the spatial discretization and parallel spatial decomposition:

denovo_discretization [name] (default: sc)

Spatial discretization. One of:

		N_u
ld	Linear discontinuous	4
sc	Step characteristics	1
tld	Trilinear discontinuous	8
twd	Theta-weighted diamond difference	1
wdd	Diamond difference	1
wdd_ff	Diamond difference with negative flux fixup	1

The step-characteristics discretization scheme is robust with regard to mesh size and voxel aspect ratios and always produces positive fluxes given positive sources. It has first-order accuracy (i.e., the discretization error decreases linearly with mesh size in the asymptotic limit) and requires storage of only one unknown per voxel in memory. Because of its robustness, positivity, and low memory requirements, it is the default scheme in ADVANTG. Note that the step-characteristics method will tend to overestimate the fluxes deep within an attenuating material.

The linear and trilinear discontinuous (LD and TLD) schemes have second-order accuracy, but do not ensure positivity. The LD and TLD methods require storage of four and eight unknowns per voxel, respectively, and they also incur a similar increase in run time. These discretization schemes tend to give very accurate results on well-refined meshes but can break down on coarser grids. The LD scheme tends to be rather sensitive to the aspect ratio of the mesh voxels. The TLD scheme is more robust than LD, but it incurs a significant cost relative to step characteristics.

The diamond difference and theta-weighted diamond techniques have a long history. The weighting helps to dampen but does not entirely eliminate the oscillations exhibited by diamond difference solutions in 3-D problems. It also reduces the order of accuracy to somewhere between first and second order. Because the step characteristics scheme provides significantly better solutions in most problems, the use of the diamond difference technique is generally discouraged.

```
denovo_x_blocks  int > 0  (default: 1)
denovo_y_blocks  "        "
denovo_z_blocks  "        "
```

The number of spatial domain partitions in the x and y dimensions and the number of pipelining blocks in the z dimension.

The Koch-Baker-Alcouffe 3-D parallel sweep algorithm is implemented in Denovo on a domain that is decomposed into x - y blocks. Each parallel core is assigned a single x - y block. So for example, with `denovo_x_blocks` and `denovo_y_blocks` both set to 2, a parallel calculation across four cores is performed. The number of blocks in each dimension should be chosen in proportion to the number of mesh intervals.

The x - y blocks can be divided into multiple blocks in the z dimension. The z dimension blocks do not affect the domain decomposition (or the number of cores used), but they determine the frequency at which fluxes are communicated between neighboring domains. This setting can impact performance in problems that have a large number of intervals in the z dimension.

By default, a single-core calculation is performed with no domain decomposition. Note that Denovo must be built with an MPI library in order to execute parallel calculations.

The following options control the partitioning of the energy groups across processors:

```
denovo_energy_sets  int > 0  (default: 1)
```

The number of parallel partitions in energy.

The parallel performance of the solver scales well with the number of spatial domain blocks. At some point, however, the domain blocks are small enough that communication overhead reduces efficiency. Energy partitioning provides a means to maintain solver efficiency across a larger number of cores. The number of executing cores is the product of the number of x - y domain blocks and the number of energy sets. This feature is generally needed only in massively parallel S_N calculations.

```
denovo_partition_upscatter  bool  (default: False)
```

If `True`, partition just the upscatter groups. This option is only valid when more than one energy set is used and when `anisn_upscatter` is `True` (see Section 5.6).

The following options and settings control the S_N angular quadrature:

```
denovo_quadrature  name  (default: qr)
```

Type of quadrature set. One of:

glproduct	Gauss-Legendre product
ldfe	Linear-discontinuous finite element (triangular)
levelsym	Level symmetric (triangular)
qr	Quadruple range (product or triangular)
userdefined	User-defined set

Denovo provides several types of angular quadratures which fall into two basic classes: triangular and product. Triangular quadrature sets are characterized by an integer order (see the `denovo_quad_order` or `denovo_ldfe_order` keywords). Product quadratures are determined by the number of azimuthal and polar angles per octant (see the `denovo_quad_num_azi` and `denovo_quad_num_polar` keywords). Triangular quadrature sets are generally rotationally symmetric, whereas product quadrature sets are generally not. However, product quadratures provide the flexibility to vary the number of azimuthal and polar angles independently, which can be more efficient in some types of transport problems.

Note that the run time of the S_N calculations scales nearly linearly with the number of quadrature angles (all else being equal). For triangular quadratures, the number of angles does not increase linearly with the order (see below). Memory usage is independent from the number of angles, because only the $(L + 1)^2$ angular flux moments are retained in memory.

Gauss-Legendre (GL) product quadratures are formed as the Cartesian product of a set of uniformly distributed azimuthal angles and a 1-D Gauss-Legendre quadrature in the polar angle.

Linear-discontinuous finite element (LDDE) quadratures (Jarrell 2010) are based on an approach that approximates the angular flux as $\psi \approx c_1 + c_2\mu + c_3\eta + c_4\xi$, where μ , η , and ξ are the direction cosines with respect to the x , y , and z coordinate axes. The quadrature is determined by requiring that the integration of the basis functions equals the surface area of the unit sphere. The LDDE sets have positive weights and are rotationally symmetric about all three axes. There are $4^{(N+1)}$ angles per octant, where N is the order. Unlike level-symmetric quadratures, the LDDE order can be even or odd. The maximum LDDE order in Denovo is seven.

Level-symmetric quadratures have a long history with the S_N method. They are rotationally-symmetric quadratures that have positive weights up to S_{20} . In 3-D, there are $N(N + 2)/8$ angles per octant, where N is the order. The maximum order supported by Denovo is 24. The order must be an even number.

Quadruple range (QR) product quadratures (Abu-Shumays 2001) exactly integrate maximal-order products of sines and cosines of the polar and azimuthal angles. The maximum numbers of azimuthal and polar angles per octant for QR quadratures in Denovo are 37 and 16, respectively. The QR product quadratures perform well across a broad range of transport problems and tend to exhibit far less ray effects than level-symmetric quadratures. For these reasons, the QR product set with four azimuthal and four polar angles per octant is the default quadrature used by ADVANTG. This set is often sufficient for generating variance reduction parameters for neutron-only problems. For photon-only problems or Denovo-only calculations, a more refined quadrature set is generally recommended.

Denovo also implements the QR quadrature as a triangular type quadrature that contains the same number of angles as a level-symmetric quadrature set of the same order. However, the triangular QR quadrature is not rotationally symmetric. The maximum order is 32.

For nonsymmetric quadratures (GL and QR), the default polar axis is the z -axis. A different polar axis can be selected using the `denovo_quad_polar_axis` keyword.

Denovo also provides the facility to read in user-provided quadratures. See the `denovo_quad_file` keyword below.

```
denovo_quad_order even int > 1 (default: 10)
denovo_ldfe_order int > 0 (default: 1)
```

Triangular quadrature orders. The `denovo_quad_order` keyword sets the order for level-symmetric and triangular QR sets. The maximum orders implemented in Denovo for the level-symmetric, QR, and LDFE quadratures are 24, 32, and 7, respectively. Note that for orders greater than 20, the level-symmetric quadratures will contain some negative ordinate weights.

```
denovo_quad_num_azi int > 0 (default: 4)
denovo_quad_num_polar " "
```

Number of azimuthal and polar directions per octant for product quadrature sets. The maximum numbers of azimuthal and polar angles per octant for QR quadratures are 37 and 16, respectively. There are no such limits for the GL product quadrature.

```
denovo_quad_num_azi_vec int > 0 ...
```

For product quadrature sets, the number of azimuthal angles per polar angle per octant, ordered from the polar axis toward the equator. If specified, this option takes precedence over `denovo_quad_num_azi`.

Example:

```
denovo_quadrature qr
denovo_quad_polar_axis z
denovo_quad_num_polar 4
denovo_quad_num_azi_vec 3 4 5 6
```

This example specifies a QR product quadrature set with four polar angles per octant, a variable number of azimuthal angles, and a total of 144 (= 18×8) directions. The polar levels with maximum and minimum cosines with respect to the positive z -axis contain three and six azimuthal angles per octant, respectively.

```
denovo_quad_polar_axis axis ... (default: z)
```

Polar axis for nonsymmetric quadrature sets. Must be one of: x , y , or z .

```
denovo_quad_file filename
```

Filename of a user-provided quadrature file. This is ignored unless `denovo_quadrature` is `userdefined`.

The quadrature file is a free-format plaintext file. The direction cosines and weight of each quadrature direction are listed on a single line as: $\mu_n \ \eta_n \ \xi_n \ w_n$. The direction cosines must be normalized so that $\mu^2 + \eta^2 + \xi^2$ is within 1 ± 10^{-6} . The weights must be strictly greater than zero. Lines with a hash character (#) in the first column are ignored.

The following options and settings control the treatment of scattering. If `anisn_upscatter` is `False` (the default), no upscatter iterations are performed. Upscatter iterations are performed if `anisn_upscatter` is `True` and the multigroup library contains upscatter data.

denovo_pn_order `int >= 0` (`default: 3`)

Order of the Legendre scattering-angle expansion. Denovo supports scattering expansions up to 11th order. Most multigroup cross section libraries have scattering data up to 5th or 7th order.

The memory consumed by Denovo scales as $(L + 1)^2$, where L is the scattering order. Thus, a P_3 calculation consumes four times more memory than a P_1 calculation.

Generally, a relatively high expansion order (P_3 and above) is recommended for photon and coupled neutron-photon calculations. A lower order can often lead to a negative scattering source (caused by truncation error); this can produce negative fluxes, even with the step characteristic differencing scheme. Negative fluxes are still possible with a higher order expansion; however, the magnitudes of any negative scattering sources are generally much lower. It is possible to enforce a positive scattering source (see `denovo_transport_correction`) at the expense of accuracy.

denovo_transport_correction `name` (`default: diagonal`)

Transport correction. One of:

cesaro	Cesaro positive-preserving
diagonal	$\hat{\sigma}_g = \sigma_g - \sigma_{sL+1}^{gg}$ $\hat{\sigma}_{sl}^{gg} = \sigma_{sl}^{gg} - \sigma_{sL+1}^{gg}, l = 0, \dots, L$
none	No transport correction

The diagonal transport correction should provide accurate results for most problems. The Cesaro correction ensures a positive scattering source, but at the expense of accuracy. It can be used if `denovo_pn_order` is at least 2.

Denovo contains two embedded first-collision source capabilities: an analytic point-kernel treatment and a Monte Carlo implementation for distributed sources. The Monte Carlo implementation traces rays from randomly sampled starting points in random directions. It can only be used with sources that have a single energy spectrum.

Point sources on the MCNP SDEF card are handled with the analytic first-collision source. The Monte Carlo first-collision source treatment must be explicitly activated for distributed sources using the following options and settings:

denovo_mc_first_collision `bool` (`default: False`)

If `True`, enables the Monte Carlo first collision source for distributed sources. By default, no first-collision source treatment is applied to distributed sources.

denovo_mc_num_particles `int > 0` (`default: 10000`)

Number of particles to simulate with the Monte Carlo first collision treatment.

The following options and settings control the within-group and upscatter solvers:

denovo_solver `name` (**default:** gmres)

Within-group solver. One of:

gmres	Restarted GMRES solver
si	Source (Richardson) iteration

The GMRES solver generally converges more quickly and is able to converge to tighter tolerances than source iteration, especially for problems with thick scattering media.

denovo_multigroup_solver `name` (**default:** gauss_seidel)

Upscatter solver. One of:

gauss_seidel	Gauss-Seidel solver
gmres	Restarted GMRES solver

This option is ignored if `anisn_upscatter` is `False`. The GMRES upscatter solver will generally converge more rapidly than Gauss-Seidel iterations. However, the GMRES option is not currently compatible with either first-collision source implementation.

denovo_preconditioner `name` (**default:** none)

Preconditioner for the within-group solver. One of:

dsa	Diffusion synthetic acceleration
none	No preconditioner

denovo_two_grid `bool` (**default:** False)

If `True` and if `anisn_upscatter` is `True`, enables two-grid upscatter acceleration.

denovo_krylov_space `int > 0` (**default:** 20)

Maximum number of Krylov vectors to store for the GMRES within-group solver.

Reducing the number of vectors reduces the memory consumed by Denovo, particularly in problems with a relatively small number of solved energy groups (e.g., as in photon-only calculations). Setting the number of vectors too low can increase the number of within-group iterations needed to reach the convergence criterion or prevent convergence altogether. Increasing the number of Krylov vectors beyond the default value typically has no benefit.

denovo_max_iterations `int > 0` (default: 100)

Maximum number of within-group iterations.

The GMRES solver will generally converge well before the default upper limit is reached, even in problems with thick scattering materials. Nonetheless, the number of actual iterations per group should be studied to ensure that all within-group solves were converged.

denovo_tolerance `real > 0.` (default: 0.001)

Convergence criterion for the within-group iterations.

The convergence measure used by Denovo is the L_2 norm of the residual vector for the GMRES solver and the L_∞ norm for the source iteration solver. These measures are not equivalent to the pointwise relative flux difference that is used in some other transport codes. The criterion of 10^{-3} has generally been found to be sufficient for the purpose of generating variance reduction parameters. For Denovo-only calculations, it is recommended to set this limit to 10^{-5} or lower.

denovo_upscatter_tolerance `real > 0.` (default: 0.01)

If `anisn_upscatter` is True, the convergence tolerance for the upscatter source.

denovo_upscatter_inner_iterations `int > 0` (default: 10)

If `anisn_upscatter` is True, the maximum number of iterations for each within-group solve in the upscatter iteration.

denovo_upscatter_inner_tolerance `real > 0.` (default: 0.01)

If `anisn_upscatter` is True, the tolerance for each within-group solve in the upscattering iteration.

denovo_first_group `int >= 0` (default: 0)

denovo_last_group `int >= 0`

The first and last energy groups to be solved. Note that in Denovo and ADVANTG, energy groups are indexed from zero rather than one. See Appendix A for lists of the group boundaries of the cross section libraries distributed with ADVANTG.

By default, the last group is the lowest energy neutron group in neutron-only problems and the lowest energy photon group in photon-only and coupled neutron-photon problems.

The following option(s) control the output of Denovo:

denovo_verbose `bool` (**default: True**)

If True, extra solver output (debug information, convergence information, and input configuration) is written to the `stdout` file when Denovo is executed.

5.8 OUTPUT OPTIONS

5.8.1 MCNP-Specific Options

mcnp_input_template `filename`

For CADIS or FW-CADIS calculations, the filename of an alternate MCNP5 input file to use as a basis for generating a new MCNP input with biased source (SB) and weight-window control parameter (WWP) cards. The template file is not modified. By default, `mcnp_input` is used. The new input file, named `inp`, is written to the `output/` directory.

This feature is useful when it is necessary to modify the MCNP input just for the purpose of running ADVANTG.

mcnp_mxspln `int >= 2` (**default: 100**)

Maximum number of weight-window splits per event. This parameter is written as the MXSPLN entry of the WWP card(s) in the generated MCNP input file.

mcnp_ww_ratio `real >= 2.0` (**default: 5.0**)

Ratio of the upper to lower weight-window bounds. This parameter is written as the WUPN entry of the WWP card(s) in the generated MCNP input file.

mcnp_sb_type `name` (**default: space_energy**)

Type of biased source probabilities to generate. One of:

space_energy	Generate SB cards for both spatial and energy distributions
space	Generate SB cards for spatial distributions only
energy	Generate SB cards for energy distributions only
none	Do not generate biased source probabilities

This option can be used, for example, to disable the generation of biased source cards for spatial distributions in problems with cell or cookie-cutter rejection in order to avoid the need to use a corrected starting particle weight.

```
mcnp_min_sb_samples int >= 0 (default: 1e6)
mcnp_max_sb_samples int >= 1 (default: 1e8)
```

Minimum and maximum number of times to sample the SDEF source when estimating biased source probabilities.

```
mcnp_target_sb_density int >= 0 (default: 1e4)
```

Target minimum average number of samples per SDEF distribution bin when estimating biased source probabilities.

```
mcnp_user_sb_sampling bool (default: False)
```

If true, use the SB cards in the original MCNP input file when sampling the SDEF source to estimate biased source probabilities.

By default, any input SB cards are ignored, and the bins of the distributions to be biased are sampled uniformly. In certain cases, for example with rejection or with heavily unbalanced distribution trees, this approach may produce an insufficient number of samples in some bins. This setting can be used to override the uniform bin sampling and provide arbitrary probabilities for any distribution or set of distributions.

```
mcnp_num_wgt_samples int >= 2 (default: 1e7)
```

Number of times to sample the SDEF source when estimating a corrected SDEF WGT parameter. This option is needed only when spatial source biasing is used with cell or cookie-cutter rejection.

```
mcnp_ww_collapse_factor real >= 1 (default: 1)
```

Factor by which to reduce the size of the output weight-window map. This feature is useful in problems that would otherwise generate a weight-window map larger than MCNP can handle.

The collapsing algorithm post-processes the weight-window map to remove mesh planes and/or energy boundaries in phase-space regions that contribute the least to the response of interest, as determined by the product of forward and adjoint scalar fluxes. For CADIS problems, experience indicates that tally FOMs are generally insensitive to reductions in weight-window map size using this approach. However, this is not the case for FW-CADIS problems with mesh tallies that span a significant portion of the problem domain (e.g., as in the problem described in Section 7.3). For this reason, use of this feature is not recommended in that type of problem.

5.8.2 Silo-Specific Options

silo_response_ids **bool** (default: True)
silo_source_ids “ ”

If True, writes maps of individual response or source identifiers to the Silo output file.

silo_source_strength **bool** (default: True)

If True, writes the distribution of the volume-averaged, energy-integrated source strengths to the Silo output file.

silo_ww **bool** (default: False)

If True, writes an expression for the weight-window targets to the Silo output file. The expression has a negligible effect on the size of the Silo file. This option is only valid when an adjoint solution has been calculated.

silo_edit_reactions **int name ...**

Specifies one or more edit fields to be calculated and written to the Silo output file, listed as pairs of edit reaction numbers and name strings. Edit reaction cross sections (or dose functions, fission spectra, etc.) are located in the ANISN cross section table in positions 1 to (IHT – 1), where IHT is the location of the total cross section. The number of edits available and the contents of the edits are library specific.

Note that Silo field names can contain only alphanumeric characters and underscores (_).

Example:

`silo_edit_reactions 1 nu_sigma_f`

6. OUTPUT

The primary output from ADVANTG is a WWINP file containing space- and energy-dependent weight-window lower bounds and the user's MCNP input file extended with biased source distributions and weight-window control parameters. Both files are written in a format compatible with unmodified versions of MCNP. ADVANTG also outputs status, warning, and error messages (if any) to the terminal window. Status messages are intended to indicate the progress of the computational sequence. Warning and error messages indicate potential or actual problems in inputs or computational results.

ADVANTG has the capability to write materials, scalar flux fields, sources, and responses in Silo format. Silo is an open-source library that provides an interface for reading and writing mesh-based data fields to a binary file, or to multiple files in parallel. The data in Silo files can be visualized using the VisIt visualization software. This functionality provides a means for the user to review the discretized model and flux estimates generated by Denovo. The user should carefully study this information to determine if obvious nonphysical features are present in important parts of the solution. If so, then input parameters must be modified to produce a more accurate deterministic calculation.

When ADVANTG is executed, it creates several directories below the current working directory. A list of the directories that can be created, depending on the calculation sequence, is shown in Table 6-1. In these directories, ADVANTG stores nearly all of the data that it generates, as well as input and output files for the codes that it executes. Much of this information, with the exception of large data arrays, is in a human-readable format that can be reviewed and inspected in case any problems arise. By default, ADVANTG will reuse as much of this data as possible if it is executed again. For example, if a tally is added to an FW-CADIS calculation, then the source mapping and forward Denovo calculation can be reused and will not be re-executed.

Table 6-1. Problem sub-directories

Directory	Description
<code>adj_solution/</code>	Denovo working directory for adjoint discrete ordinates calculations with the <code>cadis</code> and <code>dx</code> methods.
<code>fwd_solution/</code>	Denovo working directory for forward discrete ordinates calculations with the <code>fwcadis</code> and <code>dx</code> methods.
<code>fwcadis_adj_solution/</code>	Denovo working directory for adjoint discrete ordinates calculations with the <code>fwcadis</code> method.
<code>model/</code>	Working directory for model discretization tasks. (For MCNP models, MCNP is executed in this directory to create the <code>runtpe</code> file from which ADVANTG extracts model information.)
<code>output/</code>	Output directory containing the <code>WWINP</code> file and modified MCNP input file if variance reduction parameters were calculated, the VisIt-viewable <code>fields.silo</code> file, and the <code>status.log</code> file with all status, warning, and error messages.

7. EXAMPLES

This section presents three example problems that demonstrate the use of ADVANTG to generate variance reduction parameters for continuous-energy MCNP5 simulations. The first example uses the CADIS method to accelerate a point detector tally. The second and third examples use the FW-CADIS method to obtain relatively uniform statistical uncertainties across the energy bins of a pulse-height tally and the voxels of a mesh tally, respectively. MCNP and ADVANTG input files for all three example problems can be found in the `examples/` directory of the ADVANTG distribution.

7.1 UEKI SHIELDING EXPERIMENTS

7.1.1 Background

Ueki et al. (1992) describe a series of experimental measurements that were taken to investigate the neutron shielding properties of several materials. A schematic of the experiment is shown in Fig. 7-1 (reproduced from Ueki et al., Fig. 2). In the figure, all dimensions are listed in cm. A ^{252}Cf neutron source with a strength of 4.05×10^7 n/s was placed at the center of a $50 \times 50 \times 50$ cm block of paraffin with a 45° cone-shaped opening at the front. A neutron dosimeter, providing dose-equivalent rates, was placed behind slabs of shielding material. The shielding material and thickness T were varied over the course of the experiment.

The measured results are shown in Fig. 7-2 (Ueki et al., Fig. 3). The results are reported in terms of an attenuation factor, defined as

$$\text{attenuation factor} = \frac{\text{dose}|_T}{\text{dose}|_{T=0}}. \quad (7-1)$$

No uncertainties in the measured results were reported. For the KRAFTON N2 shielding material, the authors compared measured attenuation factors to results from MCNP2B simulations. Ratios of calculated to experimentally measured results (C/E ratios) ranged from 0.84 to 1.04. Unfortunately, tabulated results were reported only for the KRAFTON material. Results for the other materials were reported in the lin-log plot shown in Fig. 7-2.

7.1.2 Objective

The objective is to (1) use ADVANTG to accelerate continuous-energy MCNP simulations of the graphite shielding experiments performed by Ueki, (2) estimate neutron dose attenuation factors for shield thicknesses of $T = 0, 2, 5, 10, 15, 20, 25, 30$, and 35 cm, (3) compare the results obtained with and without the ADVANTG-generated weight-window and source biasing parameters, and (4) determine if any statistically significant difference exists between the sets of results.

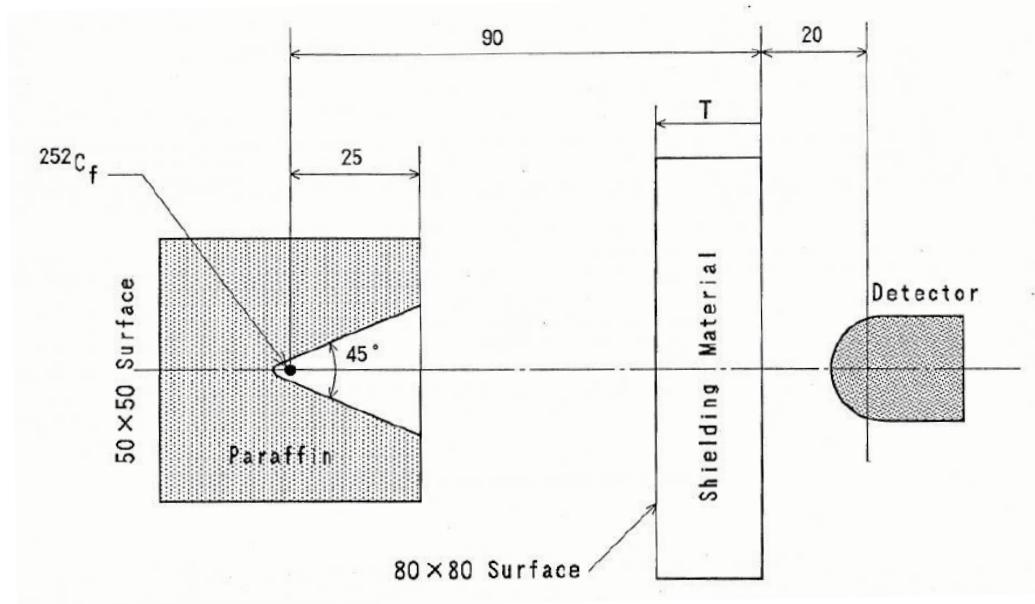


Fig. 7-1. Schematic arrangement of source, shields, and detector (Fig. 2 from Ueki et al.).

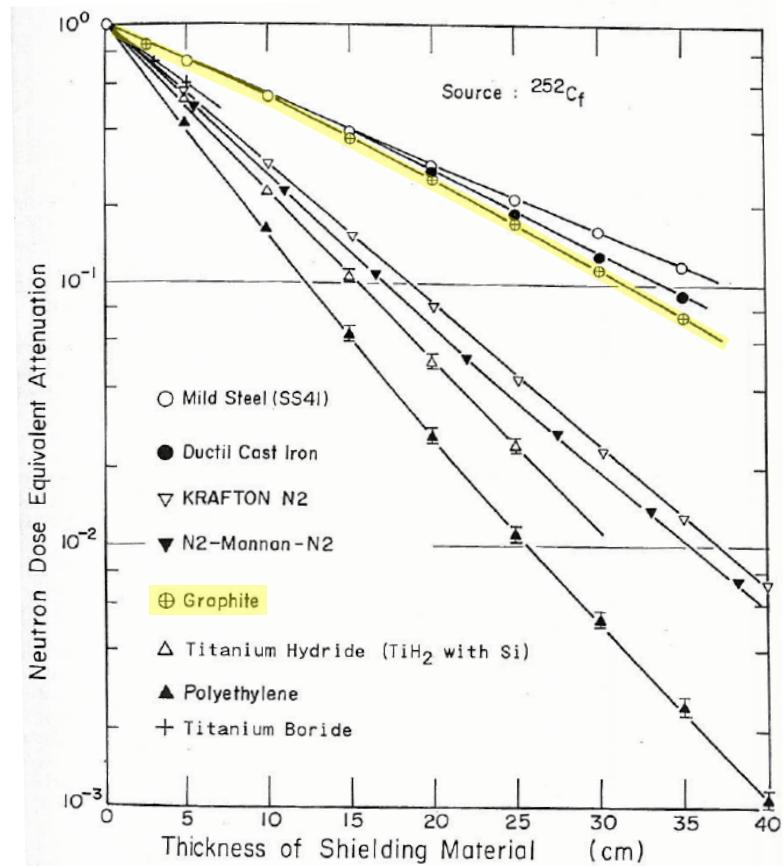


Fig. 7-2. Measured dose attenuation factors (Fig. 3 from Ueki et al.).

7.1.3 MCNP Model and Results

MCNP input files were constructed based on the dimensions shown in Fig. 7-1. Continuous energy ENDF/B-VII.0 cross sections were used for the paraffin and graphite materials, which were modeled at densities of 0.93 and 1.7 g/cm³, respectively. All other regions were modeled as void. The poly.10t and grph.10t S(α,β) tables were used for paraffin and graphite, respectively.

A point neutron source was placed at the center of the paraffin block, as shown in Fig. 7-3. An offset of 0.001 cm was used to avoid placing the source directly on a surface. The source spectrum was modeled as a Watt fission distribution. The neutron dosimeter was idealized as a point detector tally with associated ANSI/ANS-6.1.1 1977 flux-to-dose-rate conversion factors on DE/DF cards. The point detector was placed 20 cm behind the back of the shield. To illustrate that the ADVANTG variance reduction parameters are effective at increasing the number of particle tracks in the vicinity of the detector, an F4 tally was defined over a 5 × 5 × 5 cm cube surrounding the detector point.

MCNP simulations were performed on a hex-core Intel Xeon X5690 processor running at 3.47 GHz. A run-time limit of 6 min per case was used. All simulations were performed using only a single core. Tally results for the nine different shield thicknesses are summarized in Table 7-1. All point detector tallies had relative uncertainties of less than 2% and large figures of merit. However, the F4 tally results exhibit much larger uncertainties. This is expected because the volume of the tally cell is small. Though variance reduction parameters are not particularly needed for these simulations, the well-converged point detector results provide a good basis of comparison for the ADVANTG results that will be generated later.

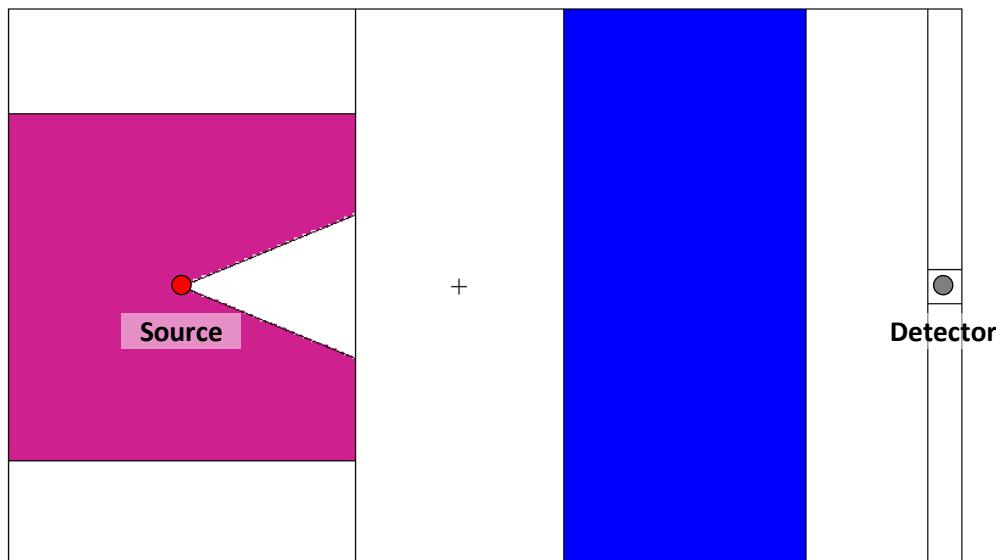


Fig. 7-3. MCNP model of Ueki experiment for the $T = 35$ cm case.

Attenuation factors were calculated based on Eq. (7-1) and are shown in Table 7-2. The table also lists C/E ratios based on experimental values obtained by reading the plot shown in Fig. 7-2. Consequently, the experimental values contain both measurement and reporting uncertainty. For the point detector tally, the C/E ratios range from 1.01 to 1.07, which is roughly comparable to the C/E ratios reported by the authors for the KRAFTON material.

Table 7-1. MCNP5 dose rate results for the Ueki problem

Graphite thickness (cm)	F4 tally		F5 tally	
	Dose rate ($\mu\text{Sv/h}$)	FOM	Dose rate ($\mu\text{Sv/h}$)	FOM
0	454 \pm 8%	25	505 \pm 0.1%	115741
2	449 \pm 8%	26	442 \pm 0.3%	17343
5	359 \pm 9%	20	372 \pm 0.5%	7234
10	245 \pm 10%	16	269 \pm 0.7%	3604
15	182 \pm 12%	11	193 \pm 0.9%	2202
20	133 \pm 14%	8	132 \pm 1.0%	1483
25	89 \pm 17%	6	89 \pm 1.3%	986
30	56 \pm 20%	4	59 \pm 1.6%	668
35	45 \pm 23%	3	40 \pm 1.9%	467

Table 7-2. MCNP5 attenuation factor results for the Ueki problem

Graphite thickness (cm)	F4 tally		F5 tally	
	Attenuation factor	C/E	Attenuation factor	C/E
2	0.989 \pm 11%	1.19	0.876 \pm 0.3%	1.06
5	0.791 \pm 12%	1.10	0.737 \pm 0.5%	1.02
10	0.539 \pm 13%	1.02	0.533 \pm 0.7%	1.01
15	0.401 \pm 15%	1.10	0.382 \pm 0.9%	1.05
20	0.292 \pm 16%	1.15	0.261 \pm 1.1%	1.03
25	0.196 \pm 19%	1.14	0.177 \pm 1.3%	1.03
30	0.123 \pm 21%	1.09	0.116 \pm 1.6%	1.03
35	0.098 \pm 25%	1.32	0.079 \pm 1.9%	1.07

7.1.4 ADVANTG Calculations

ADVANTG input files were created to generate variance reduction parameters using the CADIS method. The input for the $T = 35$ cm case is shown in Fig. 7-4 below. The point detector (tally 5) was defined as the response of interest for the CADIS calculation. ADVANTG defines a point adjoint source at the location of the point detector. This is advantageous, because all point sources are handled with the first-collision treatment in Denovo. This approach greatly reduces ray effects that would otherwise result from small volume sources in nonscattering media. Because the point detector lies at the center of the F4 tally cell, and because the first-collision treatment is desirable in this problem, the F4 tally is not listed on the `mcnp_tallies` card.

A nominally uniform mesh with 2.5-cm-thick voxels was constructed for the Denovo S_N calculation and also for the weight-window parameters. The mesh was defined so as to include the outer boundaries of the paraffin block and the slab shield. Behind the shield, the mesh planes were chosen so as to avoid placing the point adjoint source directly on a mesh boundary (because this can result in nonphysical flux estimates near the point source). The spatial mesh consists of a total of about 61,000 voxels.

```

method          cadis
mcnp_input      ueki35      # MCNP input filename
mcnp_tallies    5           # Tally id(s)
mcnp_material_names 1 paraffin # For visualization output
                           2 graphite
anisn_library   27n19g     # Multigroup library for Denovo
                           # SN calculation
denovo_pn_order 1           # Low-order angular approx is
denovo_quad_num_polar 2       # sufficient for this
denovo_quad_num_azi   2       # scattering-dominated problem

# Mesh is nominally uniform with 2.5cm-thick cells. Planes are selected so
# as to avoid putting a mesh boundary on the detector at (110, 0, 0).

mesh_x          -25 107.5 112.5 # Coarse mesh boundaries
mesh_x_ints     53      3        # Number of fine meshes per coarse mesh
mesh_y          -40  -2.5  2.5  40
mesh_y_ints     15      3      15
mesh_z          -40  -2.5  2.5  40 # Same as y dimension
mesh_z_ints     15      3      15

```

Fig. 7-4. ADVANTG input for the Ueki experiment, $T = 35$ cm case.

For this neutron-only, scattering-dominated problem, a P_1 scattering expansion and a QR quadrature set with two polar and two azimuthal angles per octant (total of 32 angles) were used. These low-order angular treatments are sufficient to capture the attenuation of the neutron flux through the shield and to generate effective variance reduction parameters for this problem. However, this kind of approximation

will generally not produce high-quality deterministic solutions that could be used in place of Monte Carlo results.

To ensure that the spatial mesh was adequate to capture the geometry of the problem, the `method` card was initially changed to use the `dx` option (see Section 5.2). The discretized model is shown in Fig. 7-5. In the figure, a clip plane at $y = 0$ was used to expose the center of the paraffin block. As discussed in Section 3.4, VisIt applies a material interface reconstruction treatment to material geometry plots by default. Though the image displays an unstructured mesh with sharp material boundaries, the actual geometry used in Denovo is defined on a structured grid with mixed materials in-between clean material regions. After inspecting the material map, the `method` card was changed back to the `cadis` option to generate variance reduction parameters.

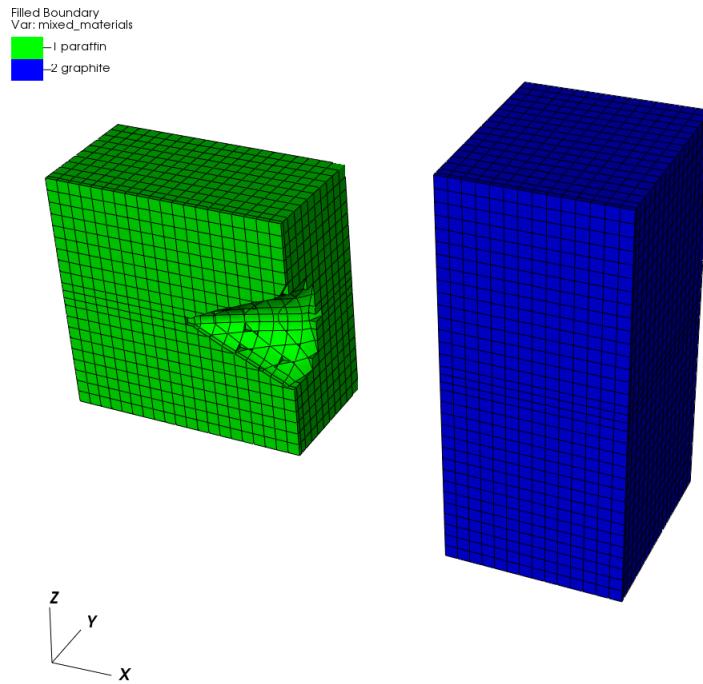


Fig. 7-5. Discretized material map for the Ueki problem.

To obtain effective source biasing parameters, changes were made to the original MCNP input file to partition the SDEF distribution bins. The original SDEF card defined a point source with a Watt fission spectrum, as shown at the top of Fig. 7-6. MCNP provides the capability to bias a continuous distribution with a histogram distribution. ADVANTG provides the capability to calculate importance-weighted biased probabilities for a given set of bins. Before running ADVANTG, an `SI` card was generated using the `watt.py` script (located in the `share/` directory) and was added to the MCNP input file, as shown at the bottom of Fig. 7-6. The script was used to calculate the boundaries of 100 equiprobable bins up to a maximum energy of 6.5 MeV and 0.5 MeV-width bins from 6.5 to 20 MeV.

```
sdef pos=0.001 0 0 erg=d1
sp1 -3 1.025 2.926
```

original SDEF cards

```
sdef pos=0.001 0 0 erg=d1
sp1 -3 1.025 2.926
c Watt spectrum (a = 1.02500, b = 2.92600)
c from 1.000e-11 to 20.0 MeV
c with 100 equiprobable bins below 6.500 MeV
c with 27 uniform bins above 6.500 MeV
si1 1.000000e-11 9.458907e-02 1.518546e-01 2.009263e-01 2.455698e-01
2.873352e-01 3.270501e-01 3.652217e-01 4.021894e-01 4.381943e-01
4.734157e-01 5.079921e-01 5.420329e-01 5.756272e-01 6.088486e-01
6.417593e-01 6.744124e-01 7.068539e-01 7.391242e-01 7.712594e-01
8.032916e-01 8.352499e-01 8.671611e-01 8.990496e-01 9.309385e-01
9.628489e-01 9.948011e-01 1.026814e+00 1.058907e+00 1.091096e+00
1.123399e+00 1.155832e+00 1.188412e+00 1.221155e+00 1.254076e+00
1.287191e+00 1.320515e+00 1.354066e+00 1.387857e+00 1.421905e+00
1.456226e+00 1.490836e+00 1.525753e+00 1.560992e+00 1.596573e+00
1.632512e+00 1.668829e+00 1.705542e+00 1.742671e+00 1.780238e+00
1.818263e+00 1.856769e+00 1.895781e+00 1.935321e+00 1.975417e+00
2.016095e+00 2.057384e+00 2.099315e+00 2.141919e+00 2.185230e+00
2.229285e+00 2.274122e+00 2.319781e+00 2.366308e+00 2.413747e+00
2.462150e+00 2.511572e+00 2.562069e+00 2.613705e+00 2.666548e+00
2.720673e+00 2.776159e+00 2.833095e+00 2.891577e+00 2.951710e+00
3.013613e+00 3.077412e+00 3.143251e+00 3.211291e+00 3.281708e+00
3.354703e+00 3.430504e+00 3.509368e+00 3.591588e+00 3.677502e+00
3.767504e+00 3.862049e+00 3.961676e+00 4.067024e+00 4.178864e+00
4.298129e+00 4.425973e+00 4.563844e+00 4.713589e+00 4.877624e+00
5.059188e+00 5.262777e+00 5.494888e+00 5.765428e+00 6.090633e+00
6.500000e+00 26i 2.000000e+01
```

modified SDEF cards for ADVANTG

Fig. 7-6. User changes to SDEF cards for the Ueki problem.

ADVANTG calculations were performed for the nine different shield thicknesses. The Denovo run times varied from 39 to 54 seconds per case, while the costs of the other operations performed by ADVANTG were negligible. The total (energy integrated) adjoint scalar flux for the $T = 5$ and 35 cm cases are shown in Figs. 7-7 and 7-8, respectively. The weight-window targets are inversely proportional to the adjoint flux. In this way, particles moving toward the detector are split into multiple tracks, whereas particles traveling toward less important regions (e.g., deep into the paraffin block) are rouleotted.

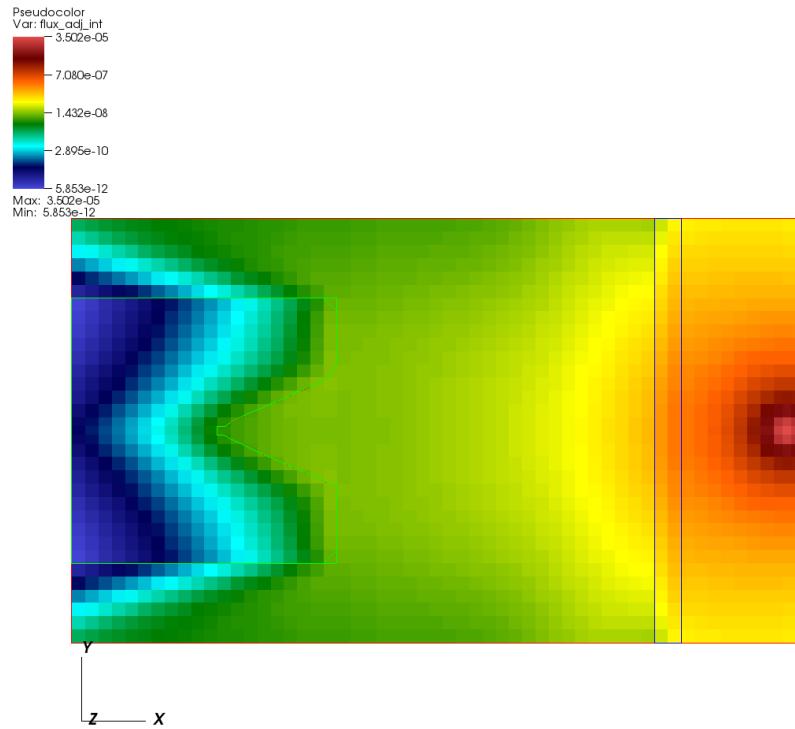


Fig. 7-7. Total adjoint flux for the Ueki problem, $T = 5$ cm case.

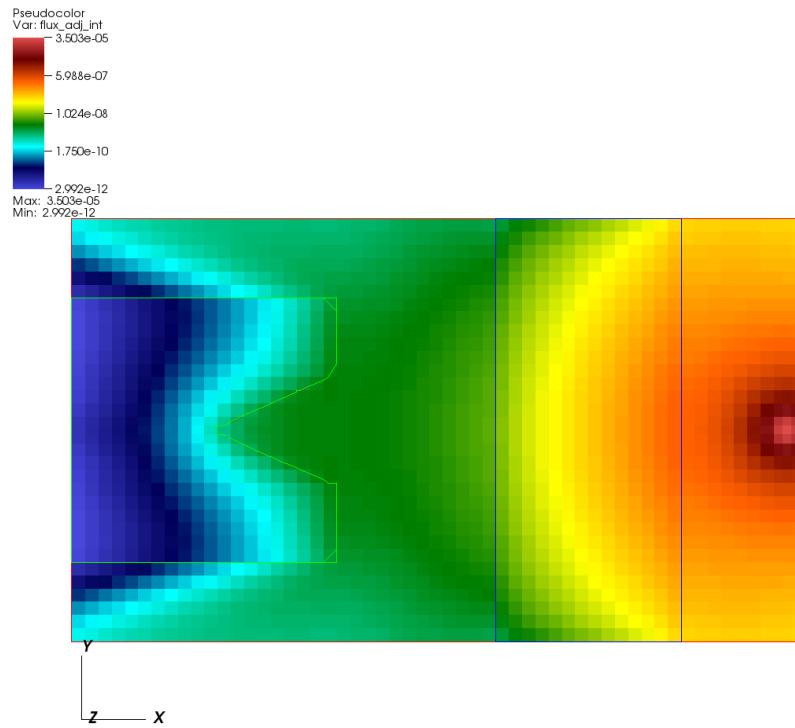


Fig. 7-8. Total adjoint flux for the Ueki problem, $T = 35$ cm case.

Once the CADIS calculation was complete, ADVANTG created a new MCNP input file and a WWINP file containing space- and energy-dependent weight-window bounds in the output/ directory. The changes made to the MCNP input are summarized in Fig. 7-9 below. ADVANTG generated an SB card with importance-weighted biased probabilities for distribution 1 and a WWP card.

```

sdef pos=0.001 0 0 erg=d1
sp1 -3 1.025 2.926
c Watt spectrum (a = 1.02500, b = 2.92600)
si1 1.000000e-11 9.458907e-02 1.518546e-01 2.009263e-01 2.455698e-01
... omitted remainder of si1 card (next 20 lines)
c * added by ADVANTG
sb1 0.00000e+00 5.50735e-04 8.55820e-04 9.66742e-04 9.66741e-04
9.66742e-04 9.66742e-04 9.66740e-04 9.66742e-04 1.75523e-03
1.89621e-03 1.89622e-03 1.89621e-03 1.89622e-03 1.89621e-03
1.89622e-03 1.89622e-03 1.89622e-03 1.89621e-03 1.89622e-03
1.89622e-03 1.89621e-03 1.89622e-03 1.89621e-03 3.18447e-03
3.63250e-03 3.63250e-03 3.63248e-03 3.63258e-03 3.63249e-03
3.63250e-03 3.63246e-03 3.63250e-03 3.63254e-03 3.63250e-03
3.63250e-03 3.63242e-03 3.63261e-03 3.63249e-03 3.63250e-03
6.04662e-03 6.10165e-03 6.10181e-03 6.10160e-03 6.10183e-03
6.10171e-03 6.10182e-03 6.10173e-03 6.10166e-03 6.10180e-03
6.10169e-03 9.41598e-03 1.03452e-02 1.03448e-02 1.03450e-02
1.03449e-02 1.03449e-02 1.03451e-02 1.03450e-02 1.03449e-02
1.03450e-02 1.03450e-02 1.03448e-02 1.03452e-02 1.03448e-02
1.03449e-02 1.03452e-02 1.03449e-02 1.03449e-02 1.03449e-02
1.03451e-02 1.03449e-02 1.03450e-02 1.03450e-02 1.03448e-02
1.05712e-02 1.73959e-02 1.73959e-02 1.73962e-02 1.73960e-02
1.73958e-02 1.73960e-02 1.73961e-02 1.73960e-02 1.73958e-02
1.73961e-02 1.73960e-02 1.73960e-02 1.73959e-02 1.73961e-02
1.73960e-02 1.73959e-02 1.73960e-02 1.73960e-02 1.73961e-02
1.73960e-02 1.73960e-02 1.73960e-02 1.73960e-02 1.73960e-02
2.45877e-02 3.99747e-02 2.88561e-02 2.07155e-02 1.47975e-02
1.05223e-02 7.45134e-03 5.25653e-03 3.69516e-03 2.58909e-03
1.80859e-03 1.25979e-03 8.75187e-04 6.06482e-04 4.19288e-04
2.89229e-04 1.99094e-04 1.36776e-04 9.37867e-05 6.41934e-05
4.38627e-05 2.99220e-05 2.03802e-05 1.38604e-05 9.41284e-06
6.38366e-06 4.32361e-06 2.92464e-06
c * added by ADVANTG
wwp:n 5.0 j 100 j -1 0 5.600880578e-11

```

Fig. 7-9. ADVANTG changes to MCNP input for Ueki problem, $T = 35$ cm case.

As discussed in Section 3.8, the WNORM value (seventh entry) on the WWP:n card is the estimated value of R , normalized to a unit source strength based on the discrete ordinates calculation. For the $T = 35$ cm case, this value is 5.6×10^{-11} rem/h per n/s. Multiplying this value by the source strength of 4.05×10^7 n/s yields an estimate of R of $22.7 \mu\text{Sv}/\text{h}$. This dose rate is about a factor of two lower than the MCNP F5 tally result. The magnitude of this difference is expected given the parameters selected for the deterministic calculation. Nonetheless, the deterministic solution is of sufficient quality to generate effective variance reduction parameters, as will be shown in the next subsection.

The ratio of biased to unbiased source energy bin probabilities is plotted in Fig. 7-10 for the $T = 35$ cm case. The source biasing reduces the probability of sampling energies below about 2 MeV, with a minimum ratio of 0.056. The probability of sampling high-energy source particles is increased significantly, because these particles have a much greater probability of penetrating the shield and reaching the detector. Note that in the MCNP simulation, source particle weights are calculated as the ratio of the unbiased to biased probability (reciprocal of the curve plotted below) to preserve the tally mean value.

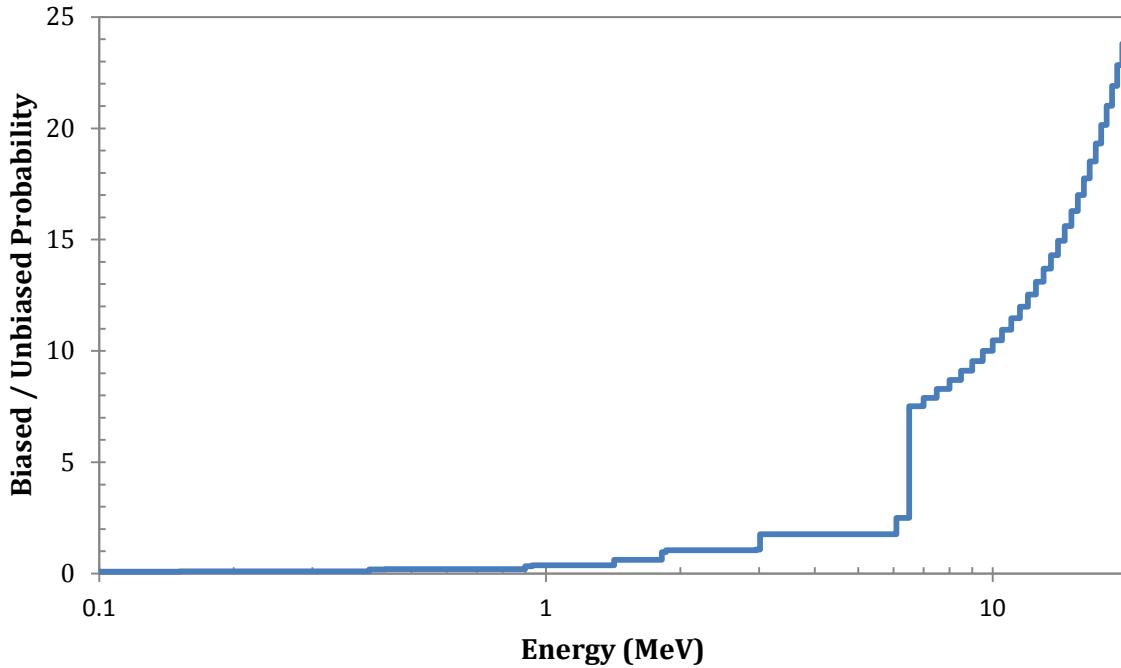


Fig. 7-10. Ratio of biased to unbiased source bin probabilities, $T = 35$ cm case.

7.1.5 Results

The ADVANTG-generated variance reduction parameters were used in MCNP simulations of the nine cases. To perform an equal-time comparison, the run-time of each simulation was limited to five minutes so that the sum of the Denovo and MCNP run times did not exceed the six minutes of the initial MCNP calculations. The ADVANTG-based dose rate and attenuation factor results are shown in Tables 7-3 and 7-4, respectively. The MCNP and ADVANTG dose rates results for the F4 and F5 tallies are compared in Tables 7-5 and 7-6. The FOM shown in the tables for the ADVANTG results was adjusted to account for Denovo run time, according to

$$\text{adjusted FOM} = \frac{1}{R^2(T_{MC} + T_{ADVANTG})}, \quad (7-2)$$

where R is the tally relative error, T_{MC} is the Monte Carlo run time in minutes, and $T_{ADVANTG}$ is the ADVANTG run time. (Generally, the ADVANTG run time is dominated by the S_N calculation.) This adjusted FOM can be used to determine whether the ADVANTG-generated parameters are worth the time

that was required to generate them. This metric can, of course, be abused by making T_{MC} larger than what is actually required in practical applications.

For this problem, the ADVANTG parameters are successful at significantly reducing the variance of both the F4 and F5 tally results. The FOMs are 15 to 600 times higher for the F4 tally and 15 to 60 times higher for the F5 tally, when accounting for the ADVANTG run time. Moreover, no statistically significant differences are observed between the MCNP results with and without the ADVANTG-generated parameters. The C/E ratios for the ADVANTG-based attenuation factor results range from 1.04 to 1.09 and 1.02 to 1.08 for the F4 and F5 tallies, respectively, whereas only the MCNP point detector results showed agreement at this level.

Table 7-3. ADVANTG-based dose rate results for the Ueki problem

Graphite thickness (cm)	F4 tally		F5 tally	
	Dose rate ($\mu\text{Sv/h}$)	FOM	Dose rate ($\mu\text{Sv/h}$)	FOM
0	495 \pm 2.2%	376	504 \pm 0.03%	1949318
2	438 \pm 1.4%	853	441 \pm 0.04%	1105217
5	370 \pm 1.0%	1831	371 \pm 0.07%	358877
10	270 \pm 0.8%	2854	271 \pm 0.11%	143522
15	189 \pm 0.8%	3065	192 \pm 0.16%	67349
20	132 \pm 0.8%	2970	132 \pm 0.19%	47528
25	89 \pm 0.8%	2740	90 \pm 0.22%	35328
30	59 \pm 0.9%	2292	60 \pm 0.26%	25073
35	40 \pm 0.9%	1922	40 \pm 0.33%	15595

Table 7-4. ADVANTG-based attenuation factor results for the Ueki problem

Graphite thickness (cm)	F4 tally		F5 tally	
	Attenuation factor	C/E	Attenuation factor	C/E
2	0.884 \pm 2.6%	1.07	0.876 \pm 0.1%	1.06
5	0.747 \pm 2.4%	1.04	0.735 \pm 0.1%	1.02
10	0.546 \pm 2.3%	1.04	0.538 \pm 0.1%	1.02
15	0.382 \pm 2.3%	1.05	0.381 \pm 0.2%	1.04
20	0.266 \pm 2.3%	1.05	0.262 \pm 0.2%	1.04
25	0.180 \pm 2.3%	1.05	0.178 \pm 0.2%	1.04
30	0.120 \pm 2.3%	1.06	0.120 \pm 0.3%	1.06
35	0.081 \pm 2.4%	1.09	0.080 \pm 0.3%	1.08

Table 7-5. Comparison of ADVANTG and MCNP5 F4 tally results

Graphite thickness (cm)	MCNP		ADVANTG	
	Dose rate ($\mu\text{Sv/h}$)	FOM	Dose rate ($\mu\text{Sv/h}$)	FOM
0	454 \pm 8%	25	495 \pm 2.2%	376
2	449 \pm 8%	26	438 \pm 1.4%	853
5	359 \pm 9%	20	370 \pm 1.0%	1831
10	245 \pm 10%	16	270 \pm 0.8%	2854
15	182 \pm 12%	11	189 \pm 0.8%	3065
20	133 \pm 14%	8	132 \pm 0.8%	2970
25	89 \pm 17%	6	89 \pm 0.8%	2740
30	56 \pm 20%	4	59 \pm 0.9%	2292
35	45 \pm 23%	3	40 \pm 0.9%	1922

Table 7-6. Comparison of ADVANTG and MCNP5 F5 tally results

Graphite thickness (cm)	MCNP		ADVANTG	
	Dose rate ($\mu\text{Sv/h}$)	FOM	Dose rate ($\mu\text{Sv/h}$)	FOM
0	505 \pm 0.1%	115741	504 \pm 0.03%	1949318
2	442 \pm 0.3%	17343	441 \pm 0.04%	1105217
5	372 \pm 0.5%	7234	371 \pm 0.07%	358877
10	269 \pm 0.7%	3604	271 \pm 0.11%	143522
15	193 \pm 0.9%	2202	192 \pm 0.16%	67349
20	132 \pm 1.0%	1483	132 \pm 0.19%	47528
25	89 \pm 1.3%	986	90 \pm 0.22%	35328
30	59 \pm 1.6%	668	60 \pm 0.26%	25073
35	40 \pm 1.9%	467	40 \pm 0.33%	15595

7.2 SIMPLIFIED PORTAL MONITOR

7.2.1 Background

Portal monitors are large detector panels used to screen cargos for illicit radioactive materials at ports of entry and weigh stations. The portal monitoring scenario described in the example problem section of Wagner et al. (2014) models a cargo container holding a ^{133}Ba point source and an array of large blocks of homogenized iron and polyethylene that partially shield the source. The geometry of the problem is shown in Fig. 7-11. The detector panel is greatly simplified and consists only of the four large sodium-iodide (NaI) crystals that are located in front of the container. The point source is located in a streaming pathway at the geometric center of the container, as shown in Fig. 7-12.

7.2.2 Objective

The objective is to: (1) use ADVANTG to accelerate a continuous-energy MCNP photon simulation of the simplified portal monitoring scenario, (2) estimate the pulse-height spectrum integrated over the four large NaI crystals resulting from the ^{133}Ba point source inside the cargo container, (3) use the FW-CADIS method to obtain relatively uniform uncertainties over 1 keV pulse-height (energy) bins, (4) compare the results obtained with and without the ADVANTG-generated weight-window and source biasing parameters, and (5) determine if any statistically significant difference exists between the sets of results.

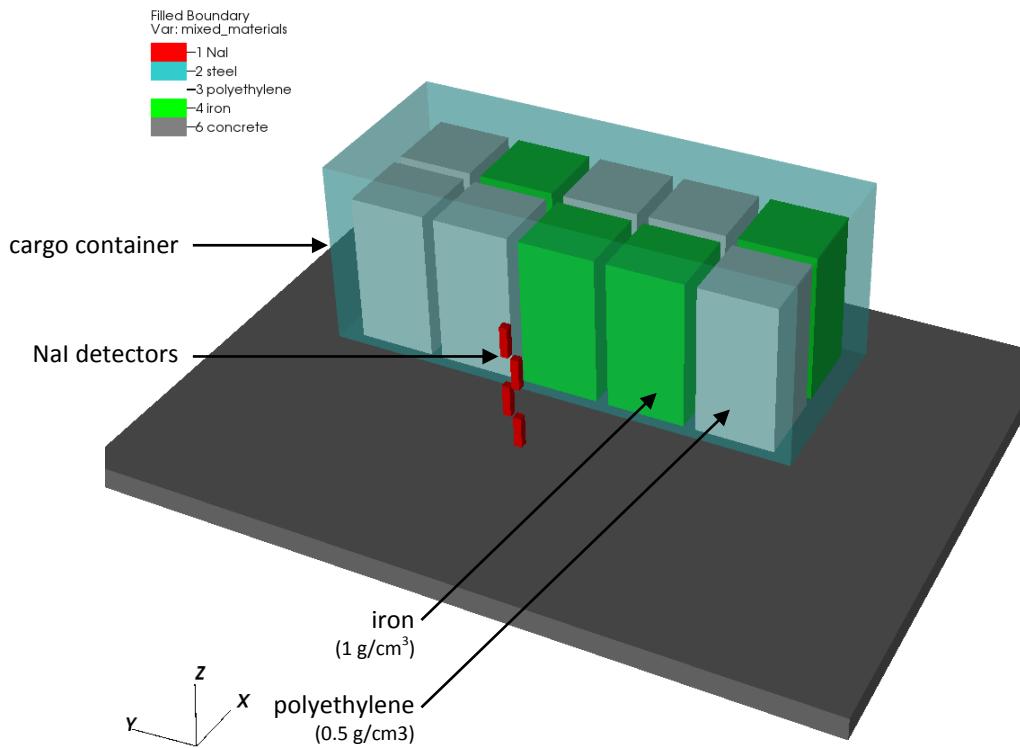


Fig. 7-11. Simplified portal problem.

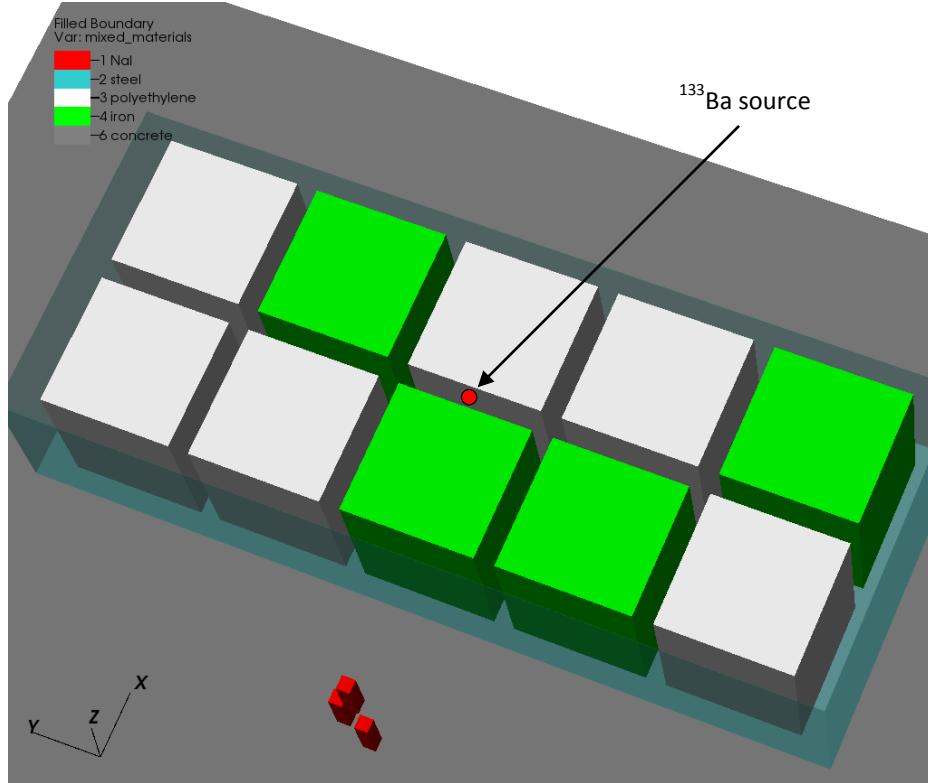


Fig. 7-12. Source location in portal problem.

7.2.3 ADVANTG Calculations

The ADVANTG input file for the simplified portal problem is shown in Fig. 7-13. The response of interest for the FW-CADIS calculation is the F8 pulse height tally. The tally has a single cell bin, which includes the four NaI cells, and multiple pulse-height bins that range from 1 to 400 keV in 1 keV widths. (The zero bin and epsilon bin recommended by the MCNP manual are also included.) The default response weighting treatment was turned off to obtain relatively uniform uncertainties across all pulse-height bins.

A nonuniform mesh that captures all of the material boundaries was constructed for the Denovo forward and adjoint S_N calculations. Within the cargo container, the voxels are nominally 10 cm thick. Additional mesh planes, parallel to the x -axis, were added to the gaps between the homogenized iron and polyethylene blocks. The entire mesh consists of a total of about 600,000 voxels. For this photon-only problem where streaming and scattering effects are important, a P_3 scattering-angle expansion and QR quadrature set with four polar and four azimuthal angles per octant (total of 128 angles) were used. This level of angular approximation is sufficient for generating variance reduction parameters for this problem.

The highest energy emission line of ^{133}Ba is 383.8 keV. Because weight-window bounds above this energy will not be used in the Monte Carlo simulation, the `denovo_first_group` card was used to set the first (highest energy) group of the discrete ordinates calculation to group number 41. (In the 27n19g library, group 41 has an upper energy of 400 keV, as shown in Table A-1 of Appendix A.) This reduces the overall run time of the forward and adjoint deterministic calculations performed in the FW-CADIS method.

```

method          fwcadis
fwcadis_response_weighting false      # Turn off response weighting

mcnp_input      cargo
mcnp_tallies    8
mcnp_material_names
101  NaI
102  steel
103  polyethylene
104  iron
105  air
106  concrete

anisn_library   27n19g

denovo_pn_order 3          # Use higher-order angular approx
denovo_quad_num_polar 4        # for this photon-only problem
denovo_quad_num_azi   4

denovo_first_group 41         # Highest Ba-133 line is 383.8 keV

mesh_x          -182.8800   0.0000   10.1600   12.1600   86.5600
                 90.5403   93.5268   95.7677   97.4490   98.7105
                 99.6571  100.3670  100.9000  101.3000  102.2000
                102.6000  103.1330  103.8430  104.7890  106.0510
                107.7320  109.9730  112.9600  116.9400  216.0000
                231.0400  330.1000  335.1130  340.1270  345.1400
                345.4400  386.0360  406.3330  467.2270  487.5240
                 528.1200

mesh_x_ints     9          5          1          16
                 1          1          1          1
                 1          1          1          3
                 1          1          1          1
                 1          1          1          10
                 5          10         1          1          1
                 1          2          1          3          1
                 2

```

continued on next page

As with the Ueki problem, the dx option was initially used to study the discretized material map shown in Figs. 7-11 and 7-12. In this problem, all material boundaries are orthogonal to one of the spatial axes, so it is possible to avoid material mixing altogether.

The original SDEF cards defined a point source with 13 discrete source energies. In this case, no partitioning of the SDEF distribution bins is possible and hence no changes need to be made to the original MCNP input file before running ADVANTG.

continued from previous page

mesh_y	-487.4800	-446.8840	-426.5870	-365.6930	-345.3960
	-304.8000	-304.5000	-277.7300	-178.6700	-173.6570
	-168.6430	-163.6300	-64.5700	-49.5300	-40.1633
	-30.7967	-21.4300	-1.2700	1.2700	21.4300
	30.7967	40.1633	49.5300	64.5700	163.6300
	168.6430	173.6570	178.6700	277.7300	304.5000
	304.8000	345.3960	365.6930	426.5870	446.8840
	487.4800				
mesh_y_ints		2	1	3	1
	2	1	2	10	1
	1	1	10	5	1
	1	1	10	2	10
	1	1	1	5	10
	1	1	1	10	2
	1	2	1	3	1
	2				
mesh_z	-30.4800	-19.8951	-12.7517	-7.9309	-4.6774
	-2.4818	-1.0000	0.0000	2.4870	4.6983
	7.9866	12.8763	20.1475	30.9600	43.5443
	50.9236	55.2507	57.7880	59.2759	60.1484
	60.6600	61.5600	62.1739	63.4300	66.0004
	71.2600	81.3733	91.4867	101.6000	223.5200
	259.3800	304.5000	304.8000	487.6800	
mesh_z_ints		1	1	1	1
	1	1	1	1	1
	1	1	1	1	1
	1	1	1	1	1
	1	3	1	1	1
	1	1	1	1	12
	4	5	1	9	

Fig. 7-13. ADVANTG input for portal problem.

ADVANTG was executed to generate variance reduction parameters using the FW-CADIS method. This requires both a forward and an adjoint Denovo calculation, which took 6.7 and 4.0 min, respectively. The other tasks performed by ADVANTG required only a few seconds.

The total forward and adjoint scalar fluxes are shown at the height of the point source in Fig. 7-14. In the figure, the left side image contains an inset image of the material map that shows the homogenized iron and polyethylene blocks as green and white, respectively. The fact that the iron material is more effective at attenuating the photon flux can be clearly observed in the forward flux map. Ray effects are visible in the adjoint flux map in the region between the detectors and the front of the cargo container. This region has a high importance because of its proximity to the detectors. However, the weight-window treatment is almost never applied in this area, because it contains only air. (MCNP applies weight windows only at surface crossings, collisions, and every mean free path of travel.) For this reason, and because the importance map within the cargo container appears to be reasonable, no further refinements were made to the deterministic calculations.

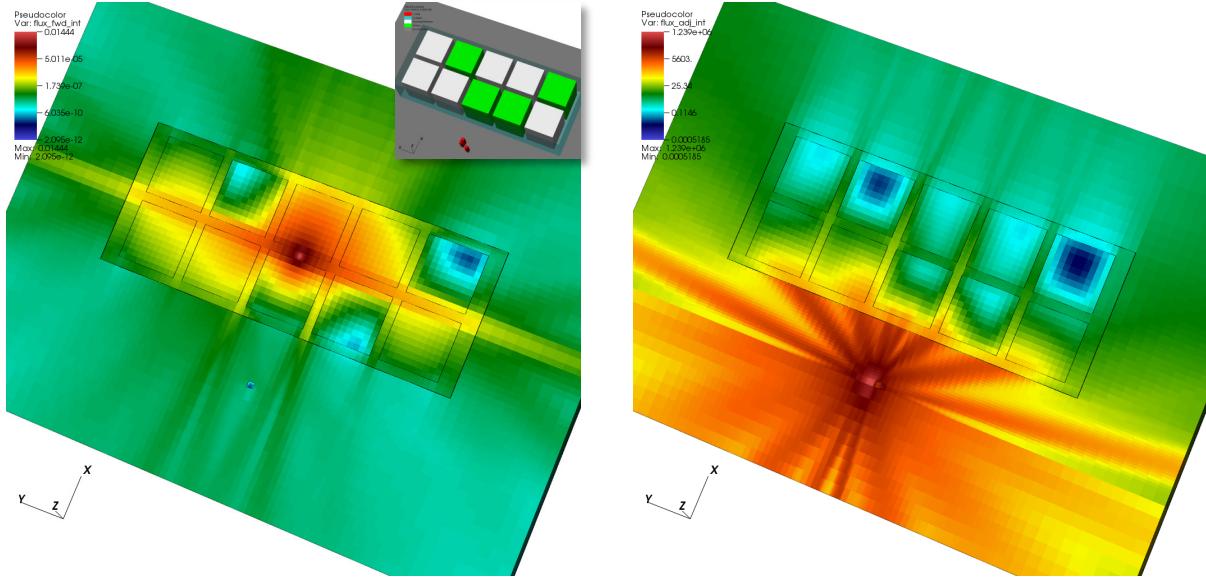


Fig. 7-14. Denovo forward (left) and adjoint (right) total fluxes for the portal problem. The fluxes are shown in the x - y plane at the source height.

ADVANTG created a new MCNP input file and a WWINP file. The changes made to the MCNP input are summarized in Fig. 7-15. ADVANTG generated an SB card with importance-weighted biased probabilities for distribution 1 and a WWP card. For this case, ADVANTG also generated a VAR card to turn off Russian roulette; this is required for MCNP to perform correct variance reduction with pulse-height tallies.

```

sdef pos 223.52 0 182.88 erg=d1
si1 L 4.619900e-03 3.062500e-02 3.097300e-02 3.495300e-02
      5.316220e-02 7.961420e-02 8.099790e-02 1.606120e-01
      2.232368e-01 2.763989e-01 3.028508e-01 3.560129e-01
      3.838485e-01
sp1   6.231324e-02 1.264485e-01 2.335836e-01 8.454194e-02
      7.798265e-03 9.656731e-03 1.200714e-01 2.332192e-03
      1.639822e-03 2.609139e-02 6.683186e-02 2.261133e-01
      3.257780e-02
c * added by ADVANTG
sb1   6.39933e-07 1.29858e-06 2.39881e-06 8.68213e-07
      2.68898e-03 3.32981e-03 4.14027e-02 3.25938e-03
      3.37495e-03 5.36992e-02 1.83182e-01 6.19764e-01
      8.92939e-02
c * added by ADVANTG
wwp:p 5.0 j 100 j -1 0 4.005146570e-01
var rr=off

```

Fig. 7-15. ADVANTG changes to SDEF cards for the portal problem.

The biasing of the source emission lines is shown in Table 7-7 below. Not surprisingly, the ratio of the biased to unbiased probability increases strongly with energy. The probabilities reflect the fact that the highest energy lines have the best chance of contributing to the pulse height tally over the detector cells. In an analog calculation, about 50% of the source particles would be sampled with energies below 50 keV. With the ADVANTG biased source, only about 5 in every 10^6 source particles will be sampled from these lines, because they have a negligible impact on the tally results.

Table 7-7. ADVANTG source energy biasing

Energy (keV)	Unbiased probability	Biased probability	Biased / unbiased
383.8	0.033	0.089	
356.0	0.226	0.620	2.741
302.9	0.067	0.182	
276.4	0.026	0.054	
223.2	0.002	0.003	2.058
160.6	0.002	0.003	1.398
81.0	0.120	0.041	
79.6	0.010	0.003	0.345
53.2	0.008	0.003	
35.0	0.085	8.68E-07	
31.0	0.234	2.40E-06	
30.6	0.126	1.30E-06	1.03E-05
4.6	0.062	6.40E-07	

7.2.4 Results

MCNP simulations of the simplified portal problem with and without the ADVANTG-generated variance reduction parameters were performed. To obtain reasonably well-converged results, the run time limits were set to 95 and 96 CPU hours, respectively. The pulse height tally results and relative uncertainties are displayed in Figs. 7-16 and 7-17. The fraction of pulse-height bins with a relative uncertainty level at or below a given level is plotted in Fig. 7-18.

The FW-CADIS variance reduction parameters have a dramatic impact on the precision of the pulse-height tally results for this problem. Relatively uniform statistical uncertainties are obtained across all pulse-height bins. Without variance reduction, 36, 57, and 67% of the pulse-height bins have relative errors less than or equal to 5, 10, and 15%, respectively. With the FW-CADIS variance reduction parameters, 92, 93, and 100% of the bins have uncertainties at or below these levels.

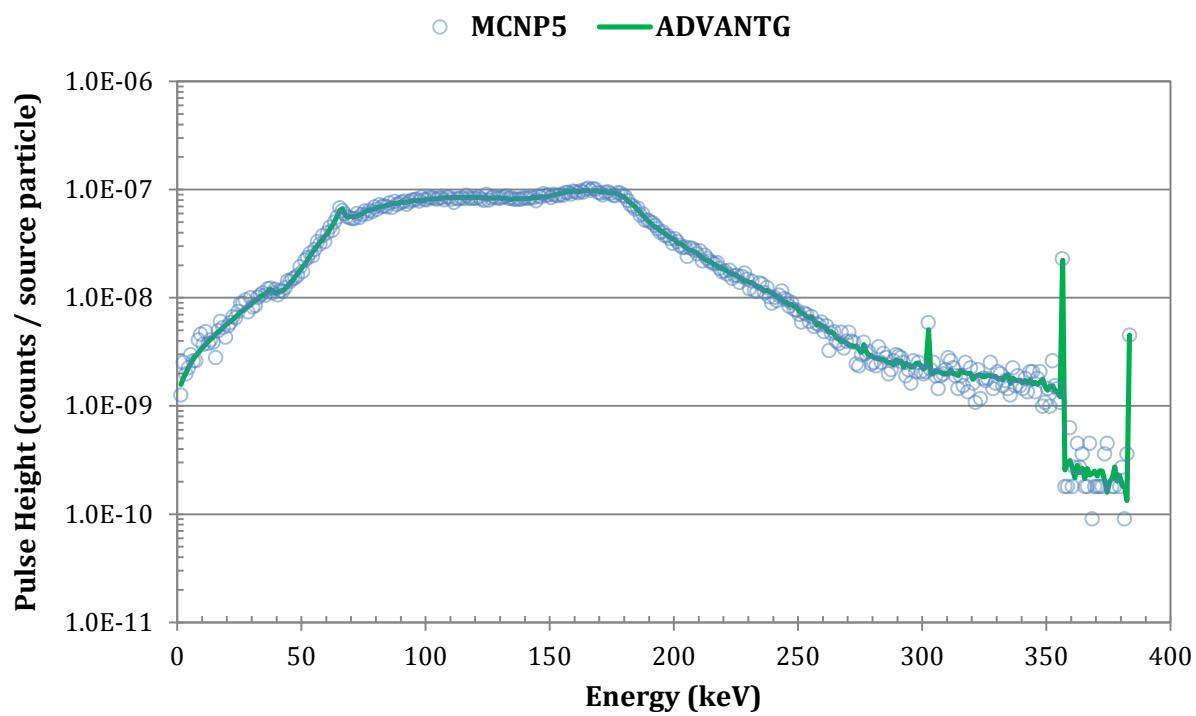


Fig. 7-16. Pulse height spectra for the portal problem.

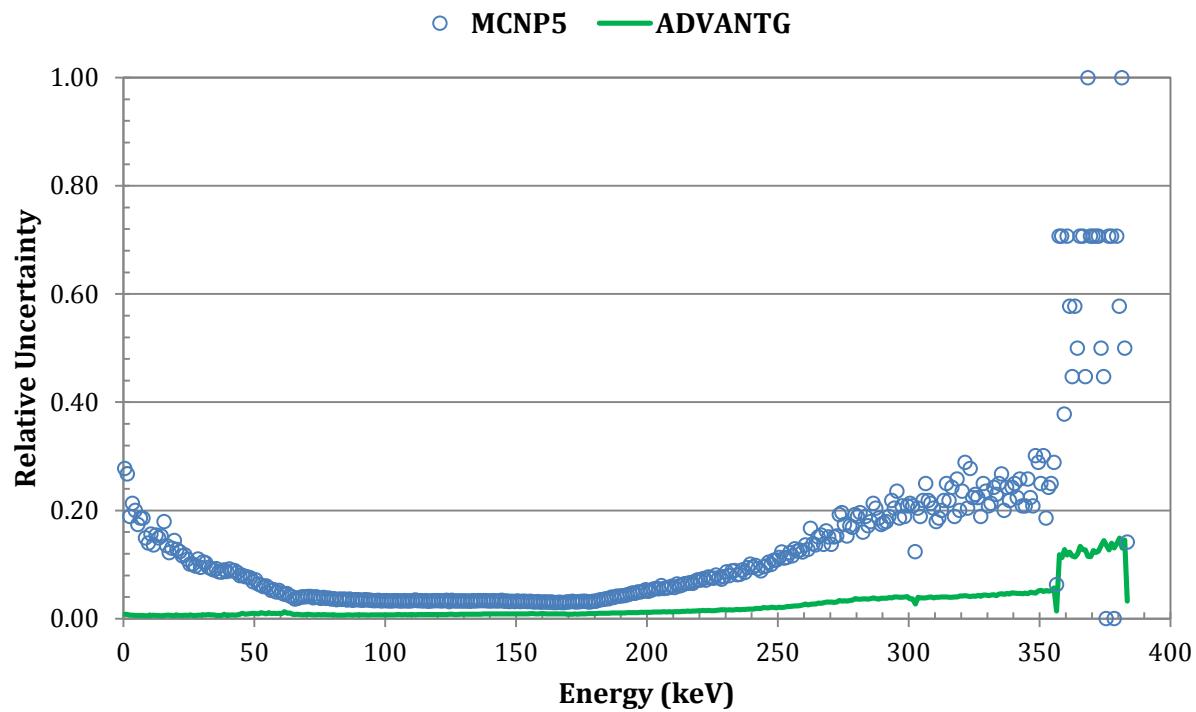


Fig. 7-17. Pulse-height relative uncertainties for the portal problem.

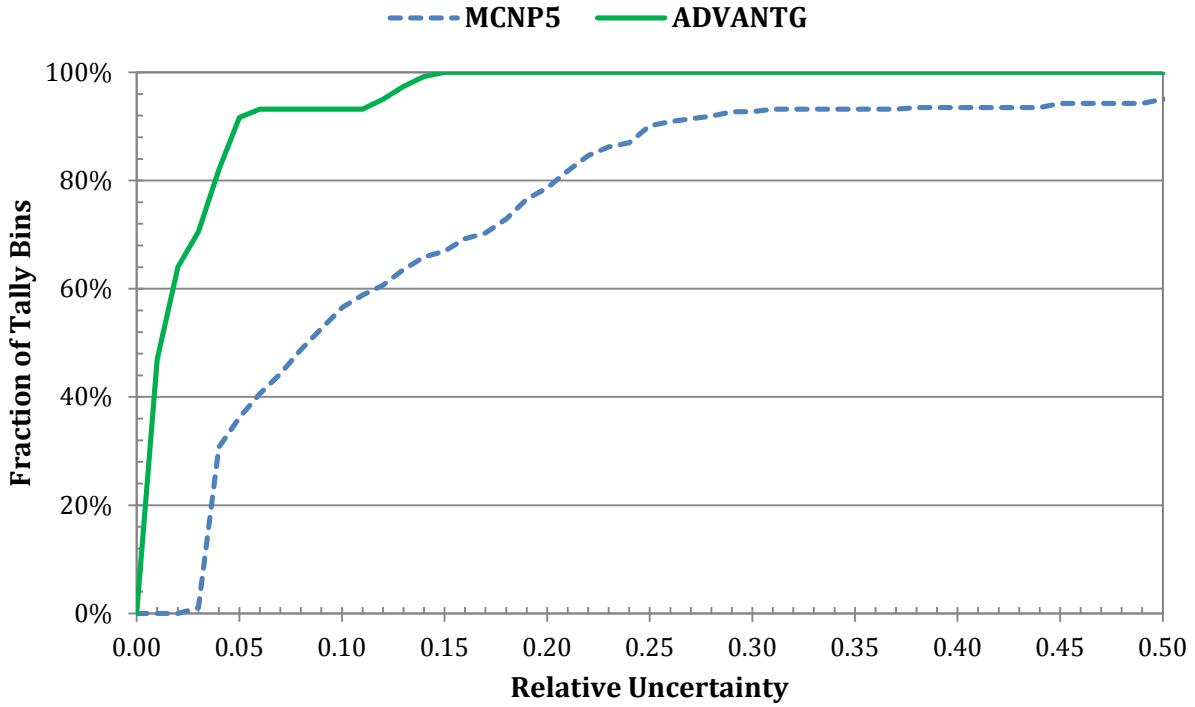


Fig. 7-18. Fraction of tally bins with less than a given uncertainty for the portal problem.

The difference between the pulse-height results estimated with and without the ADVANTG-based parameters was calculated for each bin according to

$$\text{Difference} = \frac{|ADVANTG - MCNP|}{\sqrt{\sigma_{ADVANTG}^2 + \sigma_{MCNP}^2}}. \quad (7-3)$$

The distribution of differences for this problem is shown in Table 7-8 below. The differences are approximately normally distributed; this supports the assertion that both sets of pulse-height results were drawn from the same underlying distribution.

Table 7-8. Distribution of differences for the simplified portal problem

Difference level	Fraction of bins within (%)	Normal distribution (%)
$< 1\sigma$	68.06	68.27
$< 2\sigma$	95.03	95.45
$< 3\sigma$	99.48	99.73
$< 4\sigma$	100.00	99.99

7.3 JAPAN POWER DEMONSTRATION REACTOR

7.3.1 Background

The Japan Power Demonstration Reactor (JPDR) was a prototype boiling water reactor that operated from 1963 to 1976. It was the first nuclear reactor to produce electricity in Japan and provided data and experience for later commercial reactors. It also provided a test bed for reactor decommissioning. A program to decommission the reactor began in 1981 and was completed by the mid-1990s. A schematic of the reactor enclosure and associated specifications are shown in Fig. 7-19 (Fig. 1 and Table A.1 from Sukegawa et al. 1993).

A fundamental task in planning for a decommissioning effort is estimating the radionuclide inventory in structural materials and equipment within the reactor building. Sukegawa et al. describes a 2-D, cylindrical (r, z) benchmark model of the JPDR. The benchmark was developed by the Japan Atomic Energy Research Institute to verify radionuclide inventory estimation techniques. This problem is challenging because even highly attenuated fluxes deep within concrete can contribute significantly to the build-up of ^{60}Co (from ^{59}Co impurities in steel rebar) and other radionuclides over many years of operation.

7.3.2 Objective

The objective is to: (1) use ADVANTG to accelerate a continuous-energy MCNP neutron simulation of the JPDR model, (2) estimate the energy-integrated flux distribution throughout the model with relatively uniform uncertainties over a rectangular mesh tally, and (3) compare the results obtained with and without the ADVANTG-generated weight-window and source biasing parameters. For the purpose of this example problem, the source will be modeled as emitting particles uniformly from the cylinder that encompasses the fuel region.

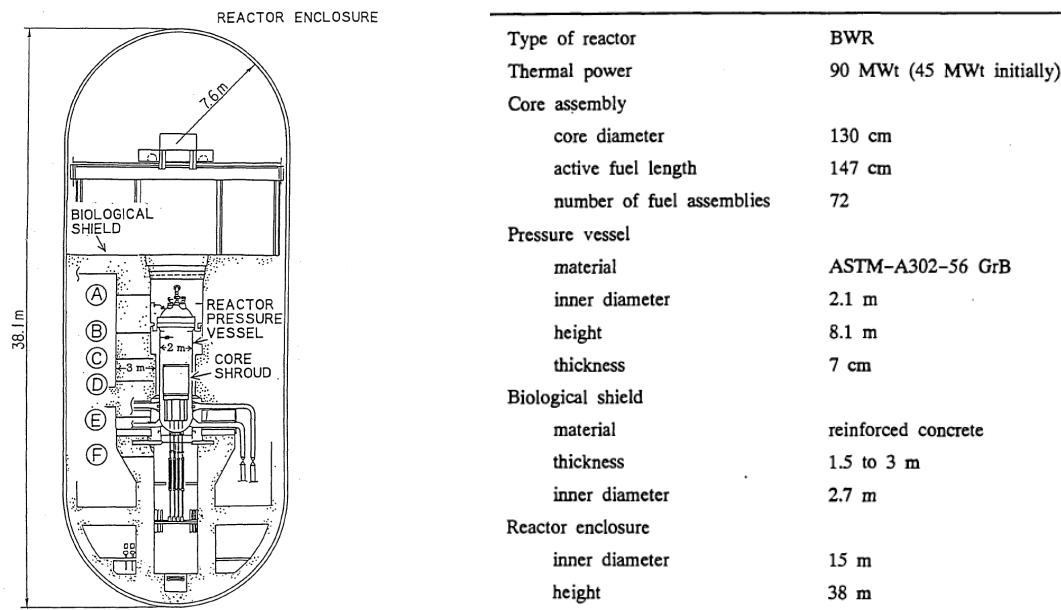


Fig. 7-19. JPDR reactor enclosure and specifications (Fig. 1 and Table A.1 from Sukegawa et al.).

7.3.3 MCNP Model and Results

The geometry of the benchmark model is shown in Fig. 7-20 (Figs. A.5[1] – [3] from Sukegawa et al.). Material compositions and temperatures are given in tables and in text, respectively, in the reference. MCNP input files were constructed based on all of this information. Continuous energy ENDF/B-VII.0 cross sections at 293, 600, 900, and 1200 K were used. The `lwtr.16t` $S(\alpha,\beta)$ tables were used for all materials that contained a significant fraction of water and for the concrete material. The geometry of the MCNP model is shown in Fig. 7-21.

The source distribution defined by the benchmark problem models the nonuniform radial and axial flux profiles that existed within the reactor core. For the purpose of this example, a simplified source description was used to distribute particles uniformly within the cylinder that encompasses the fuel region (64.79 cm radius and 146.7 cm height). The source spectrum was modeled as a Watt fission distribution with default parameters. A rectangular mesh tally with 10.7-cm-thick cells was used to tally the spatial distribution of the total flux. The mesh tally contained a total of about 500,000 voxels.

An MCNP simulation was performed on a hex-core desktop system with a run-time limit of 720 min. The simulation was performed in an hour of wall-clock time using 12 threads. Plots of the total flux per source neutron and associated relative uncertainties are shown in Fig. 7-22. In constructing the relative uncertainty plot, uncertainties of zero were set to one so that the voxels that received no tally scores do not appear to be highly converged. As is typical of conventional Monte Carlo simulations, low uncertainties were obtained near the source region. The mesh tally voxels far from the source received very few or no contributions at all. A very long run time would be required to obtain accurate estimates of the flux deep within the concrete.

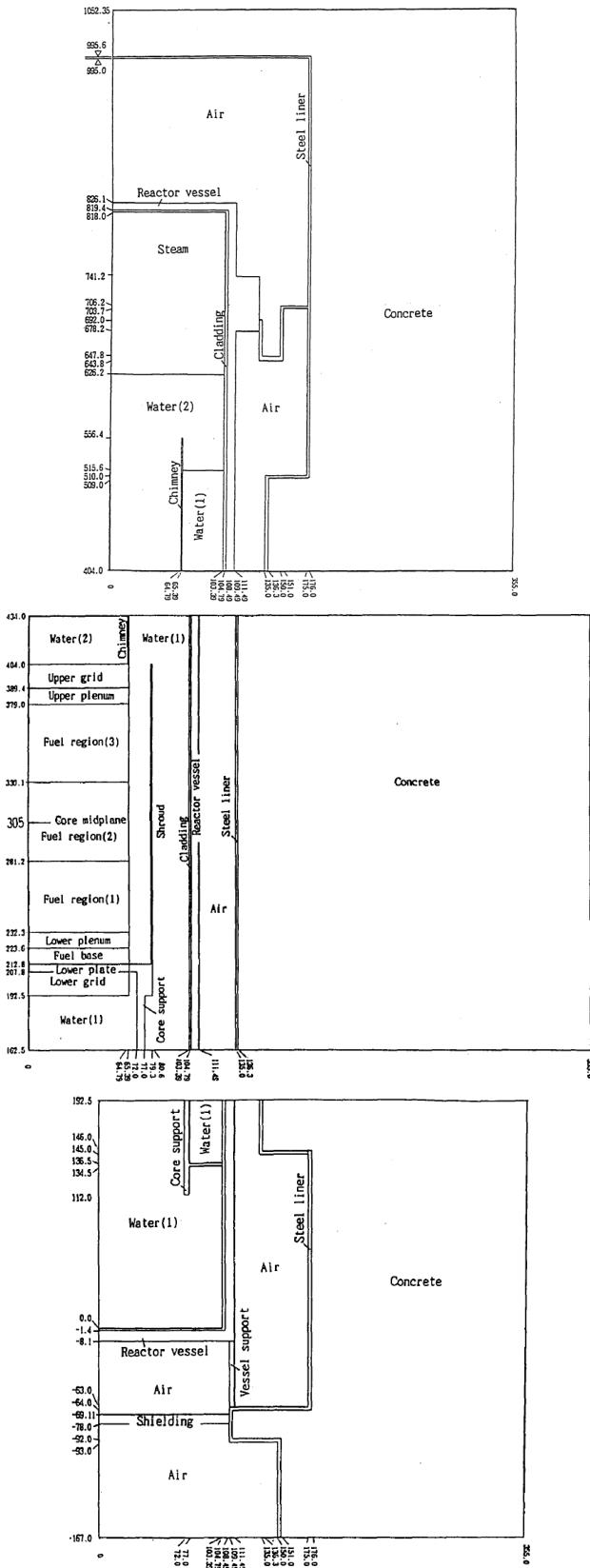


Fig. 7-20. JPDR 2-D (r , z) benchmark geometry (Figs. A.5[1] - [3] from Sukegawa et al.).

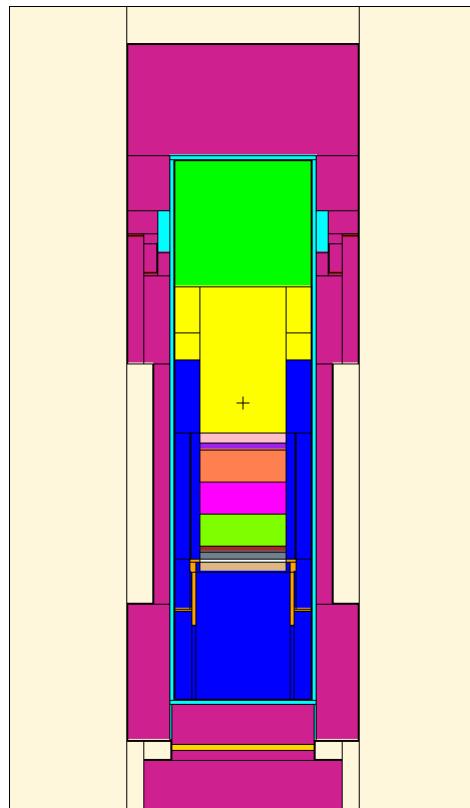


Fig. 7-21. JPDR MCNP model.

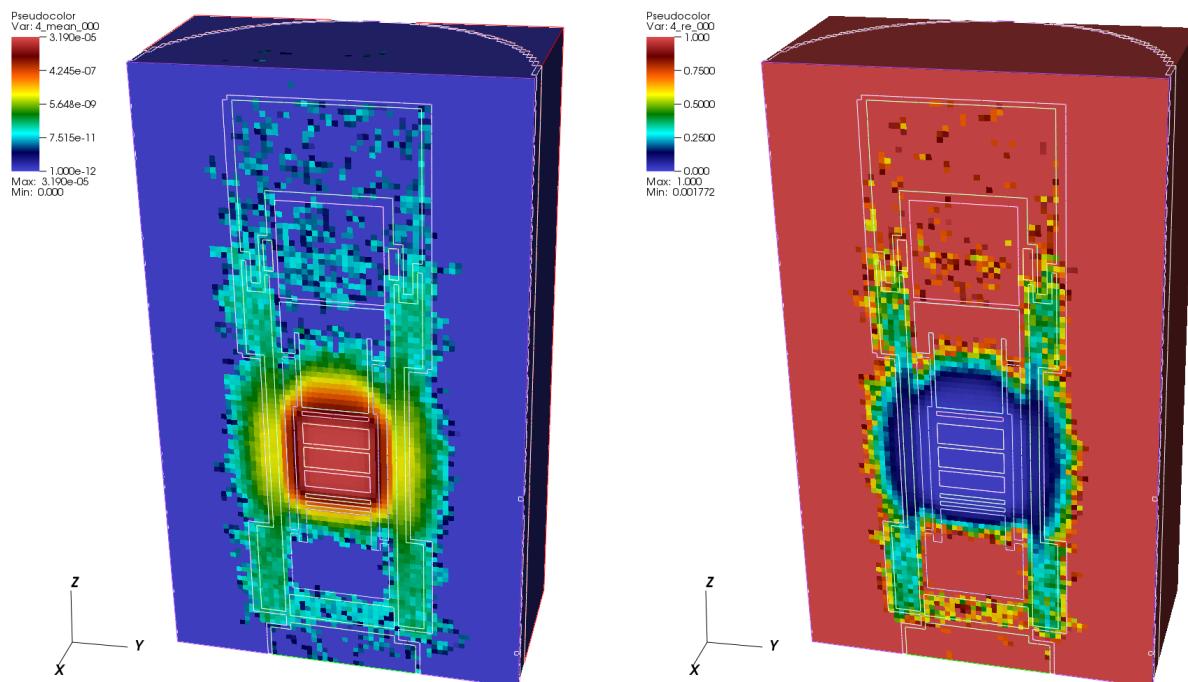


Fig. 7-22. MCNP total flux (left) and relative uncertainty (right) for the JPDR problem.

7.3.4 ADVANTG Calculations

The ADVANTG input file for the JPDR problem is shown in Fig. 7-23. The response of interest for the FW-CADIS calculation is the mesh tally defined by the FMESH4:n card in the MCNP input file. To obtain relatively uniform statistical uncertainties throughout the voxels of the mesh tally, the global weighting option was used. Because the objective of this example is to estimate total fluxes, the default response-weighting treatment was left on.

A uniform mesh with 8.5-cm-thick voxels was constructed for Denovo forward and adjoint S_N calculations. The entire mesh contains a total of about one million voxels. A P_1 scattering expansion and the default QR quadrature set with four polar and four azimuthal angles per octant were used. Because the problem is relatively large, the Denovo calculations were performed in parallel on four cores.

```

method          fwcadis
fwcadis_spatial_treatment    global

mcnp_input      jpdr
mcnp_tallies   4          # Mesh tally
mcnp_material_names
                1  air           26 "fuel base"
                11 water1        27 "lower plate"
                12 water2        28 "lower grid"
                13 steam          31 fuel1
                21 vessel         32 fuel2
                22 shroud         33 fuel3
                23 "upper grid"   41 concrete
                24 "upper plenum" 42 "steel liner"
                25 "lower plenum" 43 "lower shield"

anisn_library  bplus

denovo_pn_order 1

denovo_x_blocks 2          # Parallel job on 4 cores
denovo_y_blocks 2
denovo_z_blocks 3          # Use 3 pipelining blocks

# Uniform mesh with 8.5cm-thick cells.
mesh_x       -355 355
mesh_x_ints  83

mesh_y       -355 355      # Same as x dimension
mesh_y_ints  83

mesh_z       -167 1052.35
mesh_z_ints 144

```

Fig. 7-23. ADVANTG input for the JPDR problem.

As in previous examples, the dx option was initially used to study the discretized material map, shown in Fig. 7-24. For this problem, the 8.5-cm uniform mesh captures most of the details of the geometry model.

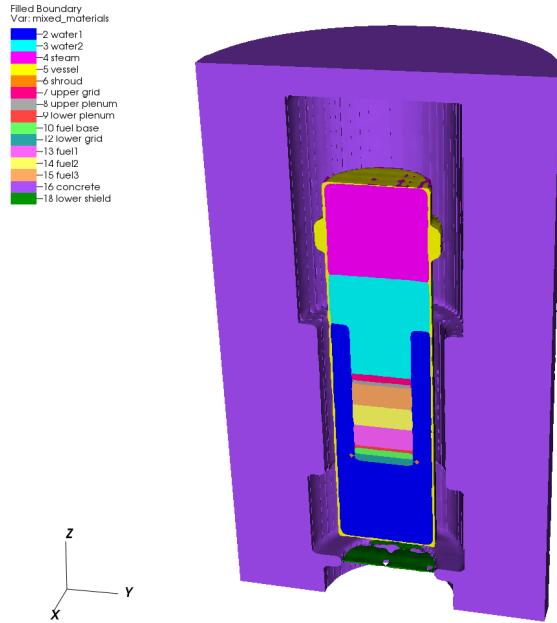


Fig. 7-24. Discretized material map for the JPDR problem.

To obtain effective source biasing parameters, the SDEF distribution bins defined in the original MCNP input file were partitioned into multiple bins. The original SDEF cards defined a cylindrical volume source with a Watt fission spectrum, as shown at the top of Fig. 7-25. Before running ADVANTG, SI and SP cards were added, as shown at the bottom of the figure. The radial and axial distributions were each subdivided into 10 equal-width bins. As in the Ueki shielding experiment problem, histogram bins were added for the Watt fission spectrum. Because of a limitation in MCNP5, only one continuous distribution can be biased at a time (see Section 9.4.1 for details). For this problem, the radial distribution was left as a continuous power law distribution. The Watt spectrum was converted from a continuous to a discrete distribution by replacing the original SP card with one containing bin-wise probabilities.

```

sdef pos=0 0 232.2 axs=0 0 1 rad=d1 ext=d2 erg=d3
sp1 -21 1
si1 0 64.79
si2 0 146.7
sp3 -3 0.965 2.29

```

original SDEF cards

```

sdef pos=0 0 232.2 axs=0 0 1 rad=d1 ext=d2 erg=d3
sp1 -21 1
si1 0 9i 64.79
c
si2 0 9i 146.7
sp2 0 1 9r
c
c   sp3 -3 0.965 2.29 $ Removed for ADVANTG
c     Watt spectrum (a = 0.96500, b = 2.29000)
c       from 1.000e-11 to 20.0 MeV
c       with 100 equiprobable bins below 6.000 MeV
c       with 28      uniform bins above 6.000 MeV
si3 1.000000e-11 7.858811e-02 1.263175e-01 1.673049e-01 2.046618e-01
2.396678e-01 2.730056e-01 3.050934e-01 3.362108e-01 3.665564e-01
3.962780e-01 4.254896e-01 4.542815e-01 4.827270e-01 5.108870e-01
5.388127e-01 5.665480e-01 5.941312e-01 6.215957e-01 6.489714e-01
6.762850e-01 7.035610e-01 7.308216e-01 7.580875e-01 7.853778e-01
8.127106e-01 8.401029e-01 8.675711e-01 8.951307e-01 9.227968e-01
9.505839e-01 9.785065e-01 1.006579e+00 1.034814e+00 1.063226e+00
1.091829e+00 1.120637e+00 1.149663e+00 1.178920e+00 1.208425e+00
1.238189e+00 1.268229e+00 1.298559e+00 1.329194e+00 1.360151e+00
1.391445e+00 1.423093e+00 1.455113e+00 1.487522e+00 1.520340e+00
1.553587e+00 1.587282e+00 1.621448e+00 1.656106e+00 1.691282e+00
1.726999e+00 1.763285e+00 1.800167e+00 1.837675e+00 1.875841e+00
1.914698e+00 1.954281e+00 1.994630e+00 2.035786e+00 2.077791e+00
2.120693e+00 2.164543e+00 2.209396e+00 2.255312e+00 2.302354e+00
2.350593e+00 2.400106e+00 2.450975e+00 2.503294e+00 2.557164e+00
2.612695e+00 2.670013e+00 2.729255e+00 2.790575e+00 2.854147e+00
2.920166e+00 2.988854e+00 3.060464e+00 3.135286e+00 3.213656e+00
3.295965e+00 3.382672e+00 3.474322e+00 3.571566e+00 3.675194e+00
3.786177e+00 3.905727e+00 4.035382e+00 4.177138e+00 4.333654e+00
4.508577e+00 4.707110e+00 4.937053e+00 5.210882e+00 5.550488e+00
6.000000e+00 27i 2.000000e+01
sp3 0 9.756146e-03 99r 7.692682e-03 5.316894e-03 3.654824e-03
2.500049e-03 1.702589e-03 1.154847e-03 7.804437e-04 5.256420e-04
3.529249e-04 2.362746e-04 1.577544e-04 1.050638e-04 6.980737e-05
4.627946e-05 3.061754e-05 2.021614e-05 1.332351e-05 8.765429e-06
5.757063e-06 3.775179e-06 2.471812e-06 1.616088e-06 1.055150e-06
6.880009e-07 4.480366e-07 2.914139e-07 1.893216e-07 1.228578e-07

```

modified SDEF cards for ADVANTG

Fig. 7-25. User changes to SDEF cards for the JPDR problem.

ADVANTG was executed to generate variance reduction parameters using the FW-CADIS method. The forward and adjoint Denovo calculations took 71 and 74 CPU min, respectively. By executing the transport sweeps on multiple cores, the calculations took only about 36 min of wall-clock time. The other tasks performed by ADVANTG consumed only a few minutes.

The total forward and adjoint scalar fluxes from the Denovo calculations are shown in Fig. 7-26. From the center of the core to the far corner of the model, the forward flux varies over about 13 orders of magnitude. The default step characteristics differencing scheme was used, so it is reasonable to expect that the drop in the flux deep within the concrete is somewhat underestimated. Denovo also provides linear-discontinuous and trilinear-discontinuous differencing options, which are more accurate, but consume more memory and run time. For the purpose of generating variance reduction parameters, the step characteristics scheme has generally been found to be sufficient.

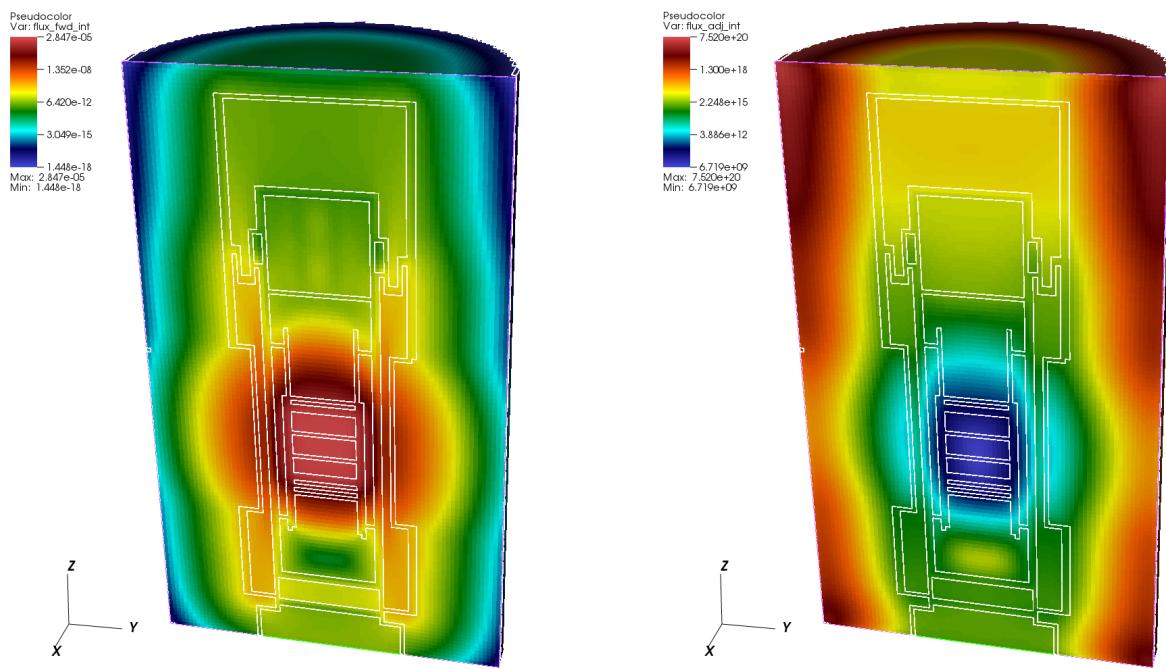


Fig. 7-26. Denovo forward (left) and adjoint (right) total flux for JPDR problem.

ADVANTG created a new MCNP input file and a WWINP file. The changes made to the MCNP input are summarized in Fig. 7-27. ADVANTG generated SB cards with importance-weighted biased probabilities for distributions 1, 2, and 3 and a WWP card. The radial biasing increases the probability of sampling particles in the two outermost radial bins, as shown in Table 7-9. The axial biasing mildly increases the probability of sampling particles near the top and bottom of the core, as shown in Table 7-10. Finally, the energy biasing parameters ensure that high-energy particles are sampled much more frequently than in an analog calculation. All of the biasing parameters encourage the sampling of particles that have a high probability of escaping the core.

```

sdef pos=0 0 232.2 axs=0 0 1 rad=d1 ext=d2 erg=d3
sp1 -21 1
si1 0 9i 64.79
c * added by ADVANTG
sb1 0.00000e+00 1.18068e-03 3.81923e-03 7.71686e-03 1.38558e-02
    2.38491e-02 4.24240e-02 7.66241e-02 1.38637e-01 2.44619e-01
    4.47274e-01
si2 0 9i 146.7
sp2 0 1 9r
c * added by ADVANTG
sb2 0.00000e+00 1.19687e-01 9.19858e-02 7.67686e-02 6.92604e-02
    6.85105e-02 7.27743e-02 8.31459e-02 1.02065e-01 1.32826e-01
    1.82977e-01
c sp3 -3 0.965 2.29 $ Removed for ADVANTG
c Watt spectrum (a = 0.96500, b = 2.29000)
si3 1.000000e-11 7.858811e-02 1.263175e-01 1.673049e-01 2.046618e-01
    ... omitted remainder of si3 card (next 20 lines)
sp3 0 9.756146e-03 99r 7.692682e-03 5.316894e-03 3.654824e-03
    ... omitted remainder of sp3 card (next 5 lines)
c * added by ADVANTG
sb3 0.00000e+00 7.92264e-05 9.98256e-05 1.09474e-04 1.19449e-04
    1.29539e-04 1.31757e-04 1.42252e-04 1.80775e-04 1.76305e-04
    1.77003e-04 1.74780e-04 1.78165e-04 1.73515e-04 1.84027e-04
    2.10024e-04 2.02329e-04 2.04453e-04 2.32284e-04 2.48785e-04
    2.57877e-04 2.51704e-04 2.49782e-04 2.75101e-04 2.99163e-04
    2.98298e-04 3.37556e-04 3.62211e-04 3.51235e-04 3.54748e-04
    3.57874e-04 3.58058e-04 3.78421e-04 5.26996e-04 5.39263e-04
    5.10395e-04 5.33903e-04 5.17355e-04 5.33640e-04 5.36325e-04
    5.34534e-04 5.37404e-04 5.25441e-04 5.32237e-04 6.46210e-04
    1.01776e-03 9.96315e-04 1.02320e-03 9.78025e-04 9.83560e-04
    1.04159e-03 9.76216e-04 1.02847e-03 1.06559e-03 1.71335e-03
    1.67775e-03 1.64756e-03 1.68783e-03 1.72450e-03 1.71231e-03
    1.73211e-03 2.74472e-03 3.04432e-03 3.05029e-03 3.07241e-03
    3.01304e-03 3.06688e-03 3.01714e-03 4.05700e-03 4.97481e-03
    5.13923e-03 6.92103e-03 7.19063e-03 8.33085e-03 9.07799e-03
    8.93173e-03 8.83296e-03 8.94436e-03 1.14671e-02 1.13922e-02
    1.17834e-02 1.14200e-02 1.30118e-02 1.35231e-02 1.35003e-02
    1.39870e-02 1.39522e-02 1.35874e-02 1.39041e-02 1.34226e-02
    2.72485e-02 2.78563e-02 2.74854e-02 2.75089e-02 2.70677e-02
    2.78565e-02 2.72168e-02 2.77040e-02 5.34031e-02 5.70908e-02
    5.76637e-02 8.45271e-02 6.18398e-02 4.46925e-02 3.81420e-02
    2.62320e-02 2.09088e-02 1.50367e-02 9.72226e-03 8.41462e-03
    5.75061e-03 3.73982e-03 2.51392e-03 2.06273e-03 1.58654e-03
    1.03821e-03 6.84569e-04 5.92152e-04 4.48678e-04 2.96386e-04
    1.92311e-04 1.25572e-04 8.08255e-05 5.39970e-05 3.58850e-05
    2.29718e-05 1.50102e-05 9.58401e-06 6.24761e-06
c * added by ADVANTG
wwp:n 5.0 j 100 j -1 0 4.161083969e+08

```

Fig. 7-27. ADVANTG changes to SDEF cards for the JPDR problem.

Table 7-9. ADVANTG radial biasing for the JPDR problem

Outer radius (cm)	Unbiased probability	Biased probability	Biased / unbiased
6.479	0.01	0.00118	0.118
12.958	0.03	0.00382	0.127
19.437	0.05	0.00772	0.154
25.916	0.07	0.0139	0.198
32.395	0.09	0.0238	0.265
38.874	0.11	0.0424	0.386
45.353	0.13	0.0766	0.589
51.832	0.15	0.139	0.924
58.311	0.17	0.245	1.439
64.79	0.19	0.447	2.354

Table 7-10. ADVANTG axial biasing for the JPDR problem

Center height (cm)	Unbiased probability	Biased probability	Biased / unbiased
7.335	0.1	0.120	1.197
22.005	0.1	0.0920	0.920
36.675	0.1	0.0768	0.768
51.345	0.1	0.0693	0.693
66.015	0.1	0.0685	0.685
80.685	0.1	0.0728	0.728
95.355	0.1	0.0831	0.831
110.025	0.1	0.102	1.021
124.695	0.1	0.133	1.328
139.365	0.1	0.183	1.830

7.3.5 Results

An MCNP simulation of the JPDR problem with the ADVANTG-generated variance reduction parameters was performed. As with the reference simulation, a run-time limit of 720 min was reached in an hour of wall-clock time using 12 threads. Plots of the total flux and associated relative uncertainties are shown in Fig. 7-28. The fraction of pulse-height bins with a relative uncertainty level at or below a given level is plotted in Fig. 7-29.

The FW-CADIS variance reduction parameters have a dramatic impact on the mesh tally results for this problem. Relatively uniform statistical uncertainties are obtained throughout the problem. Without variance reduction, only 1.7, 3, and 4% of the mesh tally voxels have relative errors less than or equal to 5, 10, and 15%. With the FW-CADIS variance reduction parameters, 36, 67, and 90% of the voxels have uncertainties at or below these levels. A conventional MCNP simulation would require an extremely long run time to obtain results converged at this level. With ADVANTG, the calculation of the variance reduction parameters and MCNP simulation could be performed on a desktop system in less than two hours of wall-clock time.

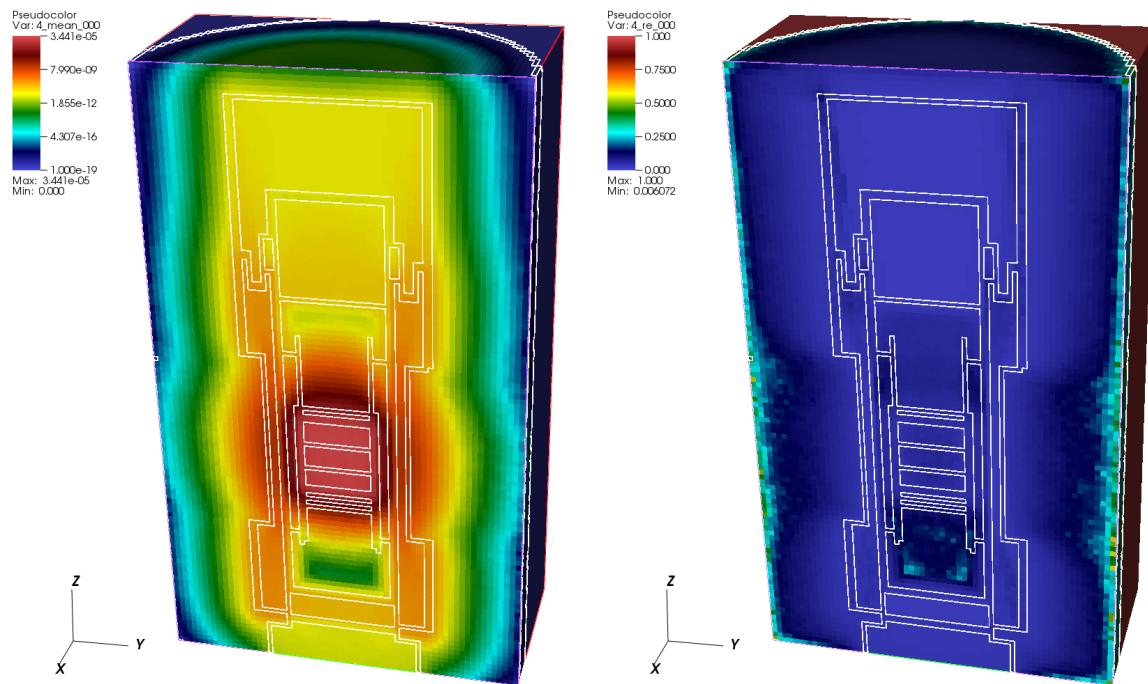


Fig. 7-28. ADVANTG-based total flux (left) and relative uncertainty (right) for the JPDR problem.

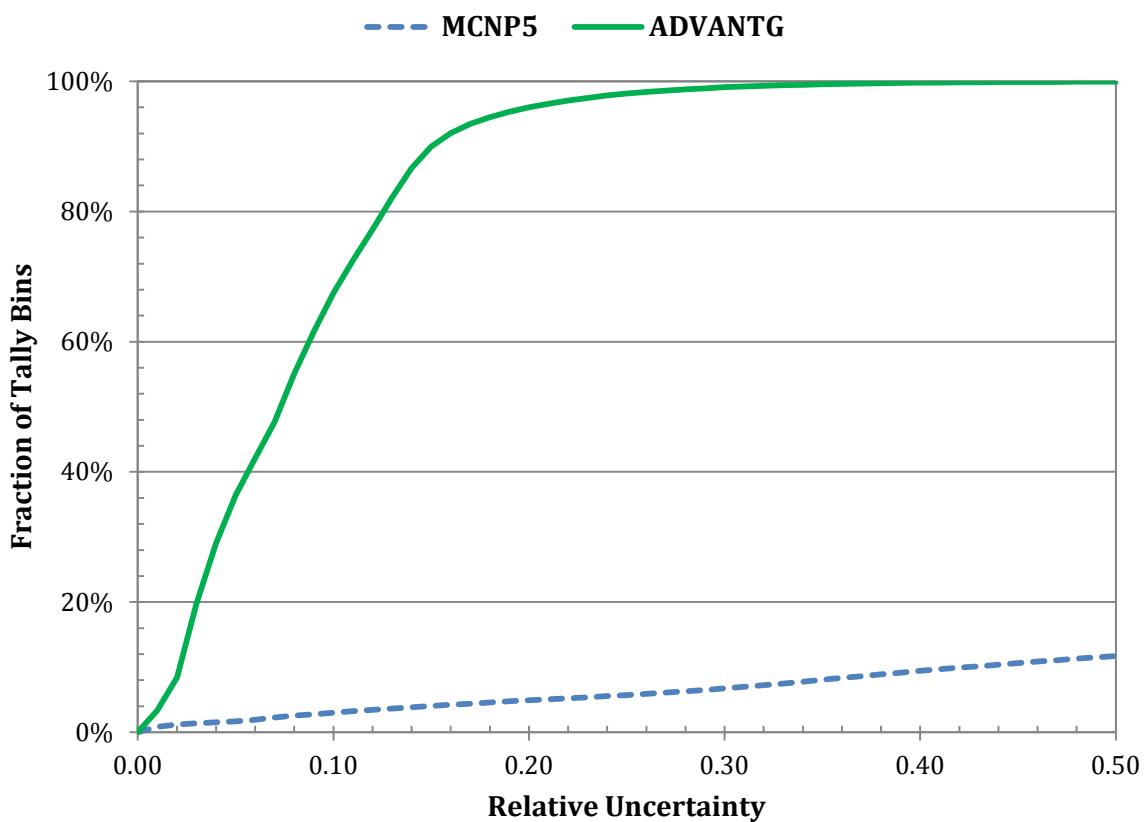


Fig. 7-29. Fraction of tally bins with less than a given uncertainty for the JPDR problem.

8. TIPS AND SUGGESTIONS

The authors have both applied and assisted with the application of ADVANTG in many challenging radiation shielding, radiation detection, neutron activation, and material damage/heating analyses and studies. This section provides suggestions, based on the authors' experience, for using ADVANTG to obtain accurate and precise tally estimates in MCNP simulations. The advice offered here is necessarily general in nature. However, the discussions may help to identify issues worth considering relative to a specific problem at hand.

8.1 SCOPING SIMULATIONS

Before applying ADVANTG to a problem, it is strongly recommended to perform an analog MCNP simulation with a time limit of at least 30 CPU minutes. Output from this initial run will provide insight into the characteristics of the transport problem and potentially indicate rates of tally convergence. In this regard, the problem summary table, the particle activity table, and the tally output tables all contain useful information. (Note that it may be necessary to add a PRINT card to the MCNP input file to obtain all relevant output.)

The problem summary table displays counts of various types of events in which particle tracks are created or destroyed. For example, the number of photon tracks created from neutron collisions, fluorescence, and bremsstrahlung are listed. Counts of tracks terminated due to escape, capture (for photons), and weight cutoff (for neutrons, due to implicit capture) are listed as well. The summary table also lists the average number of collisions per source particle. This quantity typically varies from a few to hundreds or even thousands of collisions per history, depending on type of particles being simulated and the composition and density of the material medium. Its value is highly correlated to the average cost of simulating a history.

The particle activity table (print table 126) lists the number of tracks entering each geometry cell, which can be used to gauge how frequently particles reach tally regions and their surroundings. The table also lists the total number of collisions sampled within each cell. From this information, the relative amount of simulation time spent tracking particles in the various parts of the geometry can be inferred.

The tally output tables, of course, provide a great deal of important information. At this initial stage, it is useful to examine the fraction of histories that contribute nonzero tally scores and the relative uncertainties of the tallies. In later stages, much more information should be reviewed; in particular, the ten statistical checks of tally convergence deserve careful study and consideration. At this point, if the relative uncertainty is less than about 20-25%, then the run time required to obtain a given level of uncertainty can generally be estimated with reasonable confidence.

Once a review of the scoping simulation is completed, an informed decision can be made as to whether or not it might be beneficial to apply ADVANTG to the problem. Certainly not all simulations require very long run times to obtain acceptably precise tally estimates (e.g., see the problem described in Section 7.1). If acceptably low relative uncertainties can be obtained in, say, a few to a half-dozen hours, and if most statistical checks are passed in the scoping run, then it is unlikely that there will be much to be gained by applying ADVANTG. In contrast, if very few tally scores are observed in the initial run, then it is worth considering whether or not applying weight windows and source biasing is likely to be effective at accelerating tally convergence.

The weight window technique consists of splitting and rouletting particle tracks based on space- and energy-dependent weight bounds. Splitting is helpful in problems, for example, where very few tracks reach regions of interest because of collisions in intervening materials. Similarly, rouletting is useful in problems where particles frequently collide in unimportant regions. On the other hand, neither splitting nor rouletting is particularly effective in problems where particles suffer only a few collisions on average and escape is the dominant mechanism by which particle tracks are terminated.

Source biasing is generally much more effective when used in combination with weight windows than when used by itself, though of course there are exceptions. Positional source biasing tends to be most effective in problems with large, distributed, optically-thick sources. Energy source biasing tends to be effective in problems where the tally regions are shielded from the source.

8.2 MCNP INPUT PREPARATION

ADVANTG currently requires MCNP input that is compatible with MCNP5-1.60. Inputs intended for MCNPX or MCNP6 may need to be modified before running ADVANTG. If this is the case, the `mcnp_input_template` option can be used to direct ADVANTG to merge generated SB and WWP cards into the original input file. See Section 5.8.1 for details.

It is recommended to review MCNP inputs that have been altered in the past to improve simulation performance. Not all modifications are neutral with regard to tally mean values. For example, truncating the geometry may eliminate regions that backscatter particles toward the tally or tallies of interest (e.g., room return or skyshine). Truncating the source, if not done carefully, can directly bias tally results. When effective weight windows and source biasing parameters are used, regions of the geometry and source that do not contribute strongly to tally scores will be sampled relatively less often than regions that contribute more strongly. Thus there is no need to cut off potentially important parts of the geometry or source.

When using ADVANTG, it is often necessary to partition source distributions in order to obtain effective biased probabilities (e.g., see the problem described in Section 7.3). Even continuous distributions (e.g., Watt fission spectra and power-law distributions) can be effectively subdivided by specifying histogram bins on an SI card. When these bins are present, ADVANTG will generate biased histogram probabilities for the continuous distribution. However, be careful to avoid biasing more than one continuous distribution at a time. See Sections 9.4.1 for details.

When using FW-CADIS path-length weighting to obtain relatively uniform Monte Carlo particle flux among multiple cell or surface tallies, it is important to be aware of the fact that ADVANTG treats each tally card (e.g., F4:n) as defining a single response region. This is true even if the tally card defines multiple cell or surface bins. If the cells or surfaces listed on a tally card differ significantly in volume or area, then the final relative uncertainties may be significantly nonuniform. To obtain relatively uniform uncertainties among multiple cells/surfaces it is necessary to define a tally for each cell/surface and to specify all of the tallies on the `mcnp_tallies` card.

There are implementation details to be aware of with regard to ADVANTG's treatment of tally multipliers. Though MCNP allows a tally to have more than one type of multiplier, ADVANTG will select and use only a single multiplier card to define the response spectrum. The order of precedence is DE/DF, FM, then EM. For example, if a tally has DE/DF cards, any FM or EM cards will be ignored. If no multiplier is defined, a uniform spectrum is used. It is also important to be aware that ADVANTG will use the first, and only the first, multiplier bin when constructing the tally response spectrum.

8.3 DISCRETE ORDINATES CALCULATIONS

In general, the most challenging aspect of using ADVANTG to generate effective variance reduction parameters is defining the spatial mesh, selecting the quadrature set, etc. for the discrete ordinates calculation(s). A typical starting point is a set of reasonable guesses informed by the physics of the problem and, where applicable, past experience. The default settings in ADVANTG, which cover everything except the multigroup library and spatial mesh, were selected based on the authors' experience with generating variance reduction parameters for a broad range of transport problems. Nonetheless, the defaults will not always provide solutions with sufficient fidelity.

With regard to defining the spatial mesh, it is recommended to use the `dx` option on the `method` card to generate visualization output before proceeding to execute the S_N solver. The mesh and material map can be inspected using VisIt to ensure that all important features are adequately represented in the discretized model. The edge of the mesh represents the boundary of the system in the discrete ordinates calculation. Hence it is important to include within the mesh boundaries all regions from which significant backscatter may occur.

It is generally advantageous to convert a very small volume source into a point source in order to activate a first-collision source calculation in Denovo. This treatment will help to mitigate ray effects and will produce a more accurate deterministic solution. For forward sources, the `mcnp_force_point_source` card can be used to force ADVANTG to treat any SDEF source as a point source. For adjoint sources (tallies), this must be done by manually defining a new F5 tally in the MCNP input.

Avoid placing mesh planes on or near point sources and point detectors. The first-collision source algorithm implemented in Denovo may produce nonphysical results if a forward or adjoint point source lies very close to a corner of a voxel. Ideally, point sources should be located at or near the center of a voxel.

Once a discrete ordinates solution has been generated, it is strongly recommended to visualize at least the energy-integrated fluxes using VisIt. The solution should be inspected for nonphysical features (e.g., ray effects and negativities) before executing any Monte Carlo simulations. If such features are present to a degree that they may impact the effectiveness of the variance reduction parameters, then the S_N solver settings should be refined and the calculation(s) should be re-executed. In addition, it is often useful to compare the structure of the spatial mesh to the adjoint flux in order to identify, and potentially refine, any large mesh intervals located in important regions of the problem.

ADVANTG can be used as a front end to drive Denovo calculations in situations where the discrete ordinates results are intended to be the final results. However, please note that the default settings for S_N calculations are intended for variance reduction parameter generation, where the accuracy requirements are generally not very high. For deterministic-only calculations, it is almost always necessary to increase the number of quadrature directions and Legendre scattering moments and to reduce convergence tolerances. Moreover, the broad-group cross section libraries distributed with ADVANTG may not provide sufficient fidelity for some applications. It is strongly recommended to compare the results of discrete ordinates calculations to continuous-energy Monte Carlo results for representative cases.

8.4 ADVANTG CALCULATIONS

The source biasing parameters generated by ADVANTG should be examined before proceeding to run an MCNP simulation. The difference between biased and unbiased probabilities should be proportional to the difference in expected tally contributions. If the shape of any of the biased probability distributions is

not compatible with an understanding of the transport physics of the problem, then new parameters should be generated before running MCNP. It may be necessary, for example, to refine the discrete ordinates spatial mesh in the vicinity of a source, or it may be necessary to select a more appropriate multigroup cross section library.

In some situations, it may be worthwhile to regenerate biased probabilities based on a different bin structure or an increased number of samples. Note that in these cases it is possible to recalculate biased source probabilities without re-running the discrete ordinates calculation(s). To do so, use the **-f** command line option when starting ADVANTG from the command line. See Section 4 for details.

In cases where spatial source biasing is used with cell or cookie-cutter rejection, it is extremely important to examine the relative uncertainty of the corrected SDEF **wgt** parameter, because this uncertainty directly impacts tally results. (See Section 3.8 for a discussion of issues related to using source biasing with cell rejection.) If the uncertainty is too large, it can be reduced by increasing the number of samples on the **mcnp_num_wgt_samples** card. As with source biasing parameters, the **wgt** parameter can be recalculated without re-running the discrete ordinates calculation(s).

8.5 MCNP SIMULATIONS

When using ADVANTG-generated weight windows and source biasing parameters, it is strongly recommended to perform at least one short MCNP simulation before proceeding to longer runs. There are two primary motivations for this. First, the average run time per history can increase significantly from an analog simulation. This is particularly true of FW-CADIS problems using global weighting with large tally regions. Second, tally convergence issues (as described in Section 8.6 below) can sometimes be identified at an early stage. When tally convergence is not robust, it is nearly always necessary to generate new variance reduction parameters, for example, by refining the discrete ordinates calculation(s), or perhaps by extending the mesh boundaries to include more of the problem domain. By checking for this type of problem early, the expense of a long and ultimately useless Monte Carlo run can be avoided.

It is not uncommon for ADVANTG-generated variance reduction parameters to cause an excessive amount of splitting in the MCNP simulation such that individual particle histories, either frequently or infrequently, take a very long time to complete. The presence of long histories can generally be confirmed by setting a simulation time limit, using the **CTME** card, and then observing that the simulation runs well beyond the limit. The MCNP **WWP** card provides a **MXSPLN** parameter that determines the maximum number of splits per event (i.e., surface crossing, collision, or mean free path of travel). However, reducing this limit is not generally effective at eliminating long histories.

In the authors' experience, there are two pathologies that most frequently cause long histories. The first is a discrepancy between the combinatorial and discretized geometries such that a streaming pathway or gap in the former is filled with significantly more dense material in the latter. In this case, the solution is to refine the spatial mesh in order to better resolve the feature and/or to trace a larger number of rays during the geometry mapping process to obtain more accurate mixed material fractions. The second pathology is an inaccurate adjoint solution in the low-energy groups where meshes tend to be large relative to the mean free path. The solution in this case is generally to omit one or more low-energy groups from the deterministic calculation(s) using the **denovo_last_group** card. In MCNP, the weight window parameters for the lowest given energy group extend to zero energy, so splitting and rouletting will still be applied to particles at these lower energies.

8.6 TALLY CONVERGENCE

When applying variance reduction (by any technique) or when using point detector tallies, it is extremely important to carefully review all tally output information. In particular, the ten statistical checks of tally convergence deserve careful study and consideration. See Volume I, Section 2.VI.I of the MCNP5 manual for details. Test failures serve as warnings that the tally confidence intervals may not be valid. In the authors' experience, failures of the variance-of-variance (VOV) and PDF slope tests are the strongest indicators of serious tally convergence problems in weight-window simulations.

It is extremely important to study the VOV trend printed in the tally fluctuation chart. The VOV is expected to decrease as $1/N$ after some early fluctuations, where N is the number of histories. If periodic increases in the VOV are observed throughout the simulation, or if the VOV remains at a high level, then this indicates that the tally scores include infrequent contributions from high-weight particles. This generally results from a deficiency in the discrete ordinates adjoint results, which should be examined to identify scoring pathways that are not accurately captured. This type of problem cannot be resolved by simply running more histories. New variance reduction parameters must be generated in order to obtain robust tally convergence.

Failure of the PDF slope test indicates that the high-score tail of the empirical tally score distribution is not well sampled. This may be due to a deficiency in the variance reduction parameters, or it may be due to an inadequate sample size (i.e., it may be resolved by simulating more particles). As with VOV test failures, if this test does not pass after running significantly more particles, one should look for scoring pathways that may not be accurately captured in the discrete ordinates calculation(s).

Mesh tally convergence is difficult to assess in a detailed way, because MCNP mesh tally output consists only of means and relative uncertainties. For this reason, it is strongly recommended to add one or more cell tallies within a mesh tally region in order to obtain more information about local tally convergence. If any convergence issues are observed with the cell tally, then the validity of the mesh tally results must be treated with skepticism until the issues can be resolved.

Point detectors can have interesting pathologies (see Volume I, Sections 2.V.E and 2.VI.J of the MCNP5 manual for details). Because point detector contributions are scored from every source and scattering event, point detector tallies rarely suffer from a lack of scores. However, the spatial distribution of the sampled scattering events must be considered when assessing the accuracy of point detector results. ADVANTG-generated variance reduction parameters will generally change the distribution of sampled scatters, sometimes with a surprisingly negative impact on apparent point detector convergence. This is often due to the sampling of more scattering events near the point detector, which can lead to a higher and more accurate estimate of the variance relative to an analog simulation. In cases where point detector convergence seems elusive, ensure that the point is not placed within or near a scattering material. Even when this condition is met, it may be necessary to move the point detector, or to replace it with a small, but not too small, cell tally.

9. LIMITATIONS

The implementations of the CADIS and FW-CADIS methods in ADVANTG are based on the use of scalar adjoint flux estimates from Denovo calculations. As a result, no directional biasing, in either the weight-window parameters or the biased source distributions, is currently implemented.

Support for various MCNP SDEF source and tally features is discussed in Sections 9.1 and 9.2, respectively. Limitations in Denovo and MCNP5 are described in Sections 9.3 and 9.4.

9.1 SUPPORTED MCNP SDEF DISTRIBUTION TYPES

ADVANTG samples from SDEF source distributions on two occasions during FW-CADIS calculations: once to map the source onto the deterministic mesh and once to estimate source biasing parameters. In CADIS calculations, only the latter is required. ADVANTG does not currently support SDEF sources that use transformations. In addition, sources that have circular dependencies within distribution trees are not supported.

The use of **DS H**, **DS L**, and **DS T** cards for the **ERG** variable is not currently supported for source mapping, because this would require partitioning the energy distribution bins in a spatially dependent manner. For the same reason, if **ERG** is an independent variable that is depended upon by another variable, it must be specified as an **S** distribution.

The algorithm implemented in ADVANTG for estimating a corrected SDEF **WGT** parameter when spatial source biasing is used with cell or cookie-cutter rejection does not support:

- Built-in function biasing of the **RAD** or **EXT** variables using **SB -21** or **-31** cards
- Rejection over multiple geometry or cookie-cutter cells

An error message will be printed in either of these cases. It is possible to disable the generation of spatial source biasing parameters using the **mcnp_sb_type** card. However, this may cause a significant fraction of source particles to be created with weights outside of the weight-window bounds, which can negatively impact tally convergence. Use this workaround with caution.

9.2 SUPPORTED MCNP TALLIES

ADVANTG maps tally spatial regions and response spectra onto the deterministic space-energy mesh for use as the adjoint source. The tally types that are and are not supported by ADVANTG are listed in Table 9-1. Note that ADVANTG supports rectangular mesh tallies (**FMESHn** card with **GEOM=xyz** keyword option), but it does not currently support cylindrical mesh tallies (**GEOM=cyl**) or mesh tally transformations.

ADVANTG supports the use of **FM** card tally multipliers with multiplier sets that include either the special negative reaction numbers listed in Table 3.5 in MCNP Manual, Vol. II, Sec. 3.IV.E.7, or arbitrary **MT** reaction numbers. ADVANTG does not support **FM** card attenuator sets or the zero material number option for mesh tallies.

Because pulse-height estimation is not generally available in deterministic solvers, **F8** tallies are treated as track-length cell tallies that have a total cross section multiplier. This treatment seems to be sufficient for most problems.

Table 9-1. Supported MCNP5 tally types and tally modifiers

Supported	Not supported
F1	*F1, *F2, *F4, *F5
F2	F5X, F5Y, F5Z
F4	FIP5, FIR5, FIC5
F5	F7
F6	+F8
F8	FMESHn GEOM=cyl
FMESHn GEOM=xyz	
En	DXT
FMn	
DE/DFn	
EMn	

When a tally must be modified solely for the purpose of running ADVANTG, the `mcnp_input_template` card can be used to direct ADVANTG to merge the generated `WWP` and `SB` cards into the original MCNP input deck (i.e., the deck that contains the original tally cards).

9.3 LIMITATIONS IN DENOVO

In Denovo, fission is treated as capture when performing fixed-source calculations. As a result, the variance reduction parameters generated by ADVANTG do not account for particles emitted by induced fission events. This limitation is particularly relevant to certain types of active interrogation and nondestructive assay problems. The recommended method to apply ADVANTG to these types of problems is to break the calculation into two steps: first estimate the distribution of induced fissions, and then transport the particles emitted from fission to the detector. This process is labor intensive, because it requires converting a fission neutron production tally into an SDEF source specification for MCNP. This approach also introduces some discretization error as a result of tallying the fission source in the first step.

Denovo does not treat photonuclear reactions. None of the multigroup libraries that are distributed with ADVANTG contain photonuclear data.

9.4 LIMITATIONS IN MCNP5

9.4.1 Biasing Continuous Distributions

MCNP5 supports the biasing of a continuous distribution (e.g., Watt fission spectrum) with a histogram (`SI H`) or linearly interpolated (`SI A`) distribution. In this case, MCNP replaces the continuous distribution with an equiprobable bin representation that uses at most 300 bins (X-5 Monte Carlo Team 2003, Sec. 2.VII.B.10.c). MCNP constructs an equal number of equiprobable bins in each histogram interval. This implementation introduces a bias that can be significant when the histogram contains one or more broad intervals. (MCNP prints a warning message in this case.) For this reason, caution should be exercised when applying source biasing parameters to continuous distributions. Using fine-grained histogram intervals is recommended for the biased distribution (e.g., using the `watt.py` script from the `share/` directory of the ADVANTG installation).

MCNP5 versions up to and including 1.60 do not correctly store equiprobable bin boundaries for more than one distribution. This issue causes source particles to be sampled from an incorrect distribution and can usually be observed by studying the starting coordinates of the first 50 source particles (MCNP print table 110). Because of this behavior, only one continuous source distribution should be biased at a time. For problems where it is desired to bias more than one continuous distribution, converting the remaining distributions to histogram or interpolated distributions is recommended. Caution should be exercised to ensure that this substitution does not introduce a significant bias into the final results.

9.4.2 Pulse-Height Tallies

Recent versions of MCNP5 support the use of the weight-window variance reduction technique in problems with pulse-height (F8) tallies. Pulse-height tally variance reduction requires extra bookkeeping to produce correct tally estimates in the presence of particle splitting and consumes additional memory. In most problems, the extra memory usage is not noticeable. However, in problems where there are large numbers of splitting events per history, the memory usage can be quite high. For example, in a simulation where a pulse-height tally was placed 50 m from the source, MCNP5 consumed more than 16 GB of memory when ADVANTG-generated weight windows were used. The memory usage dropped to less than 100 MB when the F8 tally was replaced with an F4 tally. For this reason, monitoring the memory consumed by MCNP5 when using the weight-window technique with pulse-height tallies is recommended.

MCNP does not allow weight windows to be used with pulse-height tallies in (n, γ) problems. Only source biasing parameters can be used in these cases.

10. REFERENCES

- Abu-Shumays, I. K. 2001. "Angular Quadratures for Improved Transport Computations." *Transport Theory and Statistical Physics*. 30 (2): 169–205.
- Bell, G. I. and S. Glasstone. 1970. *Nuclear Reactor Theory*, Van Nostrand Reinhold Co., New York.
- Childs, H., E. S. Brugger, K. S. Bonnell, J. S. Meredith, M. Miller, B. J. Whitlock, and N. Max. 2005. "A Contract-Based System for Large Data Visualization." *Proceedings of IEEE Visualization*. 190–198.
- Cooper, M. A. and E. W. Larsen. 2001. "Automated Weight Windows for Global Monte Carlo Particle Transport Calculations." *Nuclear Science and Engineering*. 137 (1): 1–13.
- Evans, T. M., A. S. Stafford, R. N. Slaybaugh, and K. T. Clarno. 2010. "Denovo: A New Three-Dimensional Parallel Discrete Ordinates Code in SCALE." *Nuclear Technology*. 171 (2): 171–200.
- Heroux, M., R. Bartlett, V. Howle, R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, and A. Williams. 2003. *An Overview of Trilinos*, SAND2003-2927. Sandia National Laboratory, Albuquerque, NM.
- Ibrahim, A. M., S. W. Mosher, T. M. Evans, D. E. Peplow, M. E. Sawan, P. P. H. Wilson, and J. C. Wagner. 2011. "ITER Neutronics Modeling Using Hybrid Monte Carlo/S_N and CAD-based Monte Carlo Methods." *Nuclear Technology*. 175 (1): 251–258.
- Ingersoll, D. T., R. W. Roussin, C. Y. Fu, and J. E. White. 1989. *DABL69: A Broad-Group Neutron/Photon Cross-Section Library for Defense Nuclear Applications*. ORNL/TM-10568. Oak Ridge National Laboratory, Oak Ridge, TN.
- López Aldama, D. and A. Trkov. 2004. *FENDL-2.1: Update of an evaluated nuclear data library for fusion applications*. INDC(NDS)-467. IAEA International Nuclear Data Committee.
- Jarrell, J. J. 2010. "An Adaptive Angular Discretization Method for Neutral-Particle Transport in Three-Dimensional Geometries." PhD dissertation, Texas A&M University.
- McKinney, G. W. and J. L. Iverson. 1996. *Verification of the Monte Carlo Differential Operator Technique for MCNP™*. LA-13098. Los Alamos National Laboratory, Los Alamos, NM.
- Novikova, E. I., M. S. Strickman, C. Gwon, B. F. Phlips, E. A. Wulf, C. Fitzgerald, L. S. Waters, and R. C. Johns. 2006. "Designing SWORD—SoftWare for Optimization of Radiation Detectors." *IEEE Nuclear Science Symposium Conference Record*. 1: 607–612.
- Pantelias, M. and S. W. Mosher. 2013. "Monte Carlo, Hybrid and Deterministic Calculations for the Activation Neutronics of Swiss LWRs." *Transactions of the American Nuclear Society*. 109 (1): 1204–1205.
- Risner, J. M. and E. D. Blakeman. 2013. *Analysis of DPA Rates in the HFIR Reactor Vessel Using a Hybrid Monte Carlo/Discrete Ordinates Methodology*. ORNL/TM-2013/515. Oak Ridge National Laboratory, Oak Ridge, TN.

- Rosman, K. J. R. and P. D. P. Taylor. 1998. "Isotopic Compositions for the Elements 1997." *Pure and Applied Chemistry*. 70 (1): 217–235.
- Shaver, M. W., E. A. Miller, R. S. Wittman, and B. S. McDonald. 2011. *Transport Test Problems for Hybrid Methods Development*. PNNL-21026. Pacific Northwest National Laboratory, Richland, WA.
- Sukegawa, T., N. Sasamoto, and K. Fujiki. 1993. *Accuracy Verification for Calculation of Inventory in JPDR Due to Neutron Activation*. INDC-JPN-0164. IAEA International Nuclear Data Committee. Available from <http://www-nds.iaea.org/publications/indc/jpn-0164.pdf>.
- Ueki, K., A. Ohashi, and Y. Anayama. 1992. "Neutron Shielding Ability of KRAFTON N2 – Mannan – KRAFTON N2 Sandwich-type Materials and Others." *ANS Radiation Protection and Shielding Division Topical Meeting*, April 26 – May 1, Pasco, WA.
- Wagner, J. C. and A. Haghishat. 1998. "Automated Variance Reduction of Monte Carlo Shielding Calculations Using the Discrete Ordinates Adjoint Function." *Nuclear Science and Engineering*. 128 (2): 186–208.
- Wagner, J. C., D. E. Peplow, S. W. Mosher, and T. M. Evans. 2010. "Review of Hybrid (Deterministic/Monte Carlo) Radiation Transport Methods, Codes, and Applications at Oak Ridge National Laboratory." *Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo (SNA + MC2010)*, October 17–21, Tokyo, Japan.
- Wagner, J. C., D. E. Peplow, and S. W. Mosher. 2014. "FW-CADIS Method for Global and Regional Variance Reduction of Monte Carlo Radiation Transport Calculations." *Nuclear Science and Engineering*. 176 (1): 37–57.
- White, J. E., D. T. Ingersoll, R. Q. Wright, H. T. Hunter, C. O. Slater, N. M. Greene, R. E. MacFarlane, and R. W. Roussin. 1995. *Production and Testing of the Revised VITAMIN-B6 Fine-Group and the BUGLE-96 Broad-Group Neutron/Photon Cross-Section Libraries Derived from ENDFB-VI.3 Nuclear Data*. ORNL-6795, NUREG/CR-6214, Revision 1. Oak Ridge National Laboratory, Oak Ridge, TN.
- Wiarda, D., M. E. Dunn, D. E. Peplow, T. M. Miller, and H. Akkurt. 2008. *Development and Testing of ENDF/B-VI.8 and ENDF/B-VII.0 Coupled Neutron-Gamma Libraries for SCALE 6*. ORNL/TM-2008/047, NUREG/CR-6990. Oak Ridge National Laboratory, Oak Ridge, TN.
- X-5 Monte Carlo Team. 2003. *MCNP—A General Monte Carlo N-Particle Transport Code, Version 5. Volume I: Overview and Theory*. LA-UR-03-1987. Los Alamos National Laboratory, Los Alamos, NM.

APPENDIX A. MULTIGROUP ENERGY BOUNDS

Table A-1. 27n19g library groups

Neutron groups		Photon groups	
Group	Upper energy boundary (MeV)	Group	Upper energy boundary (MeV)
0	2.0000E+01	27	2.0000E+01
1	6.3763E+00	28	1.0000E+01
2	3.0119E+00	29	8.0000E+00
3	1.8268E+00	30	6.5000E+00
4	1.4227E+00	31	5.0000E+00
5	9.0718E-01	32	4.0000E+00
6	4.0762E-01	33	3.0000E+00
7	1.1109E-01	34	2.5000E+00
8	1.5034E-02	35	2.0000E+00
9	3.0354E-03	36	1.6600E+00
10	5.8295E-04	37	1.3300E+00
11	1.0130E-04	38	1.0000E+00
12	2.9023E-05	39	8.0000E-01
13	1.0677E-05	40	6.0000E-01
14	3.0590E-06	41	4.0000E-01
15	1.8554E-06	42	3.0000E-01
16	1.3000E-06	43	2.0000E-01
17	1.1253E-06	44	1.0000E-01
18	1.0000E-06	45	4.5000E-02
19	8.0000E-07		1.0000E-02
20	4.1399E-07		
21	3.2500E-07		
22	2.2500E-07		
23	1.0000E-07		
24	5.0000E-08		
25	3.0000E-08		
26	1.0000E-08		
	1.0000E-11		

Table A-2. 200n47g library groups

Neutron groups				Photon groups	
Group	Upper energy boundary (MeV)	Group	Upper energy boundary (MeV)	Group	Upper energy boundary (MeV)
0	2.0000E+01	50	2.0190E+00	200	2.0000E+01
1	1.9640E+01	51	1.9205E+00	201	1.4000E+01
2	1.7332E+01	52	1.8268E+00	202	1.2000E+01
3	1.6905E+01	53	1.7377E+00	203	1.0000E+01
4	1.6487E+01	54	1.6530E+00	204	8.0000E+00
5	1.5683E+01	55	1.5724E+00	205	7.5000E+00
6	1.4918E+01	56	1.4957E+00	206	7.0000E+00
7	1.4550E+01	57	1.4227E+00	207	6.5000E+00
8	1.4191E+01	58	1.3534E+00	208	6.0000E+00
9	1.3840E+01	59	1.2874E+00	209	5.5000E+00
10	1.3499E+01	60	1.2246E+00	210	5.0000E+00
11	1.2840E+01	61	1.1648E+00	211	4.5000E+00
12	1.2523E+01	62	1.1080E+00	212	4.0000E+00
13	1.2214E+01	63	1.0026E+00	213	3.5000E+00
14	1.1618E+01	64	9.6164E-01	214	3.0000E+00
15	1.1052E+01	65	9.0718E-01	215	2.7500E+00
16	1.0513E+01	66	8.6294E-01	216	2.5000E+00
17	1.0000E+01	67	8.2085E-01	217	2.3500E+00
18	9.5123E+00	68	7.8082E-01	218	2.1500E+00
19	9.0484E+00	69	7.4274E-01	219	2.0000E+00
20	8.6071E+00	70	7.0651E-01	220	1.8000E+00
21	8.1873E+00	71	6.7206E-01	221	1.6600E+00
22	7.7880E+00	72	6.3928E-01	222	1.5700E+00
23	7.4082E+00	73	6.0810E-01	223	1.5000E+00
24	7.0469E+00	74	5.7844E-01	224	1.4400E+00
25	6.7032E+00	75	5.5023E-01	225	1.3300E+00
26	6.5924E+00	76	5.2340E-01	226	1.2000E+00
27	6.3763E+00	77	4.9787E-01	227	1.0000E+00
28	6.0653E+00	78	4.5049E-01	228	9.0000E-01
29	5.7695E+00	79	4.0762E-01	229	8.0000E-01
30	5.4881E+00	80	3.8774E-01	230	7.0000E-01
31	5.2205E+00	81	3.6883E-01	231	6.0000E-01
32	4.9659E+00	82	3.3373E-01	232	5.1200E-01
33	4.7237E+00	83	3.0197E-01	233	5.1000E-01
34	4.4933E+00	84	2.9849E-01	234	4.5000E-01
35	4.0657E+00	85	2.9721E-01	235	4.0000E-01
36	3.6788E+00	86	2.9452E-01	236	3.0000E-01
37	3.3287E+00	87	2.8725E-01	237	2.6000E-01
38	3.1664E+00	88	2.7324E-01	238	2.0000E-01
39	3.0119E+00	89	2.4724E-01	239	1.5000E-01
40	2.8651E+00	90	2.3518E-01	240	1.0000E-01
41	2.7253E+00	91	2.2371E-01	241	7.5000E-02
42	2.5924E+00	92	2.1280E-01	242	7.0000E-02
43	2.4660E+00	93	2.0242E-01	243	6.0000E-02
44	2.3852E+00	94	1.9255E-01	244	4.5000E-02
45	2.3653E+00	95	1.8316E-01	245	3.0000E-02
46	2.3457E+00	96	1.7422E-01	246	2.0000E-02
47	2.3069E+00	97	1.6573E-01		1.0000E-02
48	2.2313E+00	98	1.5764E-01		
49	2.1225E+00	99	1.4996E-01		

continued on next page

Table A-2 (cont'd)

Neutron groups			
Group	Upper energy boundary (MeV)	Group	Upper energy boundary (MeV)
100	1.4264E-01	150	1.6702E-04
101	1.3569E-01	151	1.3007E-04
102	1.2907E-01	152	1.0130E-04
103	1.2277E-01	153	7.8893E-05
104	1.1679E-01	154	6.1442E-05
105	1.1109E-01	155	4.7851E-05
106	9.8037E-02	156	3.7266E-05
107	8.6517E-02	157	2.9023E-05
108	8.2503E-02	158	2.2603E-05
109	7.9499E-02	159	1.7604E-05
110	7.1998E-02	160	1.3710E-05
111	6.7379E-02	161	1.0677E-05
112	5.6562E-02	162	8.3153E-06
113	5.2475E-02	163	6.4760E-06
114	4.6309E-02	164	5.0435E-06
115	4.0868E-02	165	3.9279E-06
116	3.4307E-02	166	3.0590E-06
117	3.1828E-02	167	2.3824E-06
118	2.8501E-02	168	1.8554E-06
119	2.7000E-02	169	1.4450E-06
120	2.6058E-02	170	1.3000E-06
121	2.4788E-02	171	1.1253E-06
122	2.4176E-02	172	1.0800E-06
123	2.3579E-02	173	1.0400E-06
124	2.1875E-02	174	1.0000E-06
125	1.9305E-02	175	8.7643E-07
126	1.5034E-02	176	8.0000E-07
127	1.1709E-02	177	6.8256E-07
128	1.0595E-02	178	6.2506E-07
129	9.1188E-03	179	5.3158E-07
130	7.1017E-03	180	5.0000E-07
131	5.5308E-03	181	4.1399E-07
132	4.3074E-03	182	3.6680E-07
133	3.7074E-03	183	3.2500E-07
134	3.3546E-03	184	2.7500E-07
135	3.0354E-03	185	2.2500E-07
136	2.7465E-03	186	1.8400E-07
137	2.6126E-03	187	1.5000E-07
138	2.4852E-03	188	1.2500E-07
139	2.2487E-03	189	1.0000E-07
140	2.0347E-03	190	7.0000E-08
141	1.5846E-03	191	5.0000E-08
142	1.2341E-03	192	4.0000E-08
143	9.6112E-04	193	3.0000E-08
144	7.4852E-04	194	2.1000E-08
145	5.8295E-04	195	1.4500E-08
146	4.5400E-04	196	1.0000E-08
147	3.5357E-04	197	5.0000E-09
148	2.7536E-04	198	2.0000E-09
149	2.1445E-04	199	5.0000E-10
			1.0000E-11

Table A-3. BUGLE-96 and BPLUS library groups

Neutron groups		Photon groups	
Group	Upper energy boundary (MeV)	Group	Upper energy boundary (MeV)
0	1.7332E+01	47	1.4000E+01
1	1.4191E+01	48	1.0000E+01
2	1.2214E+01	49	8.0000E+00
3	1.0000E+01	50	7.0000E+00
4	8.6071E+00	51	6.0000E+00
5	7.4082E+00	52	5.0000E+00
6	6.0653E+00	53	4.0000E+00
7	4.9659E+00	54	3.0000E+00
8	3.6788E+00	55	2.0000E+00
9	3.0119E+00	56	1.5000E+00
10	2.7253E+00	57	1.0000E+00
11	2.4660E+00	58	8.0000E-01
12	2.3653E+00	59	7.0000E-01
13	2.3457E+00	60	6.0000E-01
14	2.2313E+00	61	4.0000E-01
15	1.9205E+00	62	2.0000E-01
16	1.6530E+00	63	1.0000E-01
17	1.3534E+00	64	6.0000E-02
18	1.0026E+00	65	3.0000E-02
19	8.2085E-01	66	2.0000E-02
20	7.4274E-01		1.0000E-02
21	6.0810E-01		
22	4.9787E-01		
23	3.6883E-01		
24	2.9721E-01		
25	1.8316E-01		
26	1.1109E-01		
27	6.7379E-02		
28	4.0868E-02		
29	3.1828E-02		
30	2.6058E-02		
31	2.4176E-02		
32	2.1875E-02		
33	1.5034E-02		
34	7.1017E-03		
35	3.3546E-03		
36	1.5846E-03		
37	4.5400E-04		
38	2.1445E-04		
39	1.0130E-04		
40	3.7266E-05		
41	1.0677E-05		
42	5.0435E-06		
43	1.8554E-06		
44	8.7643E-07		
45	4.1399E-07		
46	1.0000E-07		
	1.0000E-11		

Table A-4. DABL69 and DPLUS library groups

Neutron groups		Photon groups	
Group	Upper energy boundary (MeV)	Group	Upper energy boundary (MeV)
0	1.9640E+01	46	2.0000E+01
1	1.6905E+01	47	1.4000E+01
2	1.4918E+01	48	1.2000E+01
3	1.4191E+01	49	1.0000E+01
4	1.3840E+01	50	8.0000E+00
5	1.2523E+01	51	7.0000E+00
6	1.2214E+01	52	6.0000E+00
7	1.1052E+01	53	5.0000E+00
8	1.0000E+01	54	4.0000E+00
9	9.0484E+00	55	3.0000E+00
10	8.1873E+00	56	2.5000E+00
11	7.4082E+00	57	2.0000E+00
12	6.3763E+00	58	1.5000E+00
13	4.9659E+00	59	1.0000E+00
14	4.7237E+00	60	7.0000E-01
15	4.0657E+00	61	4.5000E-01
16	3.0119E+00	62	3.0000E-01
17	2.3852E+00	63	1.5000E-01
18	2.3069E+00	64	1.0000E-01
19	1.8268E+00	65	7.0000E-02
20	1.4227E+00	66	4.5000E-02
21	1.1080E+00	67	3.0000E-02
22	9.6164E-01	68	2.0000E-02
23	8.2085E-01		1.0000E-02
24	7.4274E-01		
25	6.3928E-01		
26	5.5023E-01		
27	3.6883E-01		
28	2.4724E-01		
29	1.5764E-01		
30	1.1109E-01		
31	5.2475E-02		
32	3.4307E-02		
33	2.4788E-02		
34	2.1875E-02		
35	1.0333E-02		
36	3.3546E-03		
37	1.2341E-03		
38	5.8295E-04		
39	2.7536E-04		
40	1.0130E-04		
41	2.9023E-05		
42	1.0677E-05		
43	3.0590E-06		
44	1.1253E-06		
45	4.1399E-07		
	1.0000E-11		

Table A-5. FENDL67 library groups

Neutron groups		Photon groups	
Group	Upper energy boundary (MeV)	Group	Upper energy boundary (MeV)
0	1.4191e+01	46	1.4000e+01
1	1.3499e+01	47	1.2000e+01
2	1.2214e+01	48	1.0000e+01
3	1.1052e+01	49	8.0000e+00
4	1.0000e+01	50	7.5000e+00
5	9.0484e+00	51	7.0000e+00
6	8.1873e+00	52	6.5000e+00
7	7.4082e+00	53	6.0000e+00
8	6.7032e+00	54	5.5000e+00
9	6.0653e+00	55	5.0000e+00
10	5.4881e+00	56	4.5000e+00
11	4.9659e+00	57	4.0000e+00
12	4.4933e+00	58	3.5000e+00
13	4.0657e+00	59	3.0000e+00
14	3.6788e+00	60	2.5000e+00
15	3.3287e+00	61	2.0000e+00
16	3.0119e+00	62	1.5000e+00
17	2.7253e+00	63	1.0000e+00
18	2.4660e+00	64	4.0000e-01
19	1.8268e+00	65	2.0000e-01
20	1.3534e+00	66	1.0000e-01
21	1.0026e+00		1.0000e-02
22	7.4274e-01		
23	5.5023e-01		
24	4.0762e-01		
25	3.0197e-01		
26	2.2371e-01		
27	1.6573e-01		
28	1.2277e-01		
29	6.7379e-02		
30	3.1828e-02		
31	1.5034e-02		
32	7.1017e-03		
33	3.3546e-03		
34	1.5846e-03		
35	7.4852e-04		
36	3.5358e-04		
37	1.6702e-04		
38	7.8893e-05		
39	3.7267e-05		
40	1.7603e-05		
41	8.3153e-06		
42	3.9279e-06		
43	1.8554e-06		
44	8.7642e-07		
45	4.1399e-07		
	1.0000e-11		

APPENDIX B. INDEX OF FREQUENTLY USED KEYWORDS

The following table lists frequently used keywords. Keywords might be required in general (R) or only in the given context (C).

Keyword	Brief description	Req'd	Page
model	Type of transport problem model		24
method	Variance reduction method or type of computational sequence	R	24
anisn_library	Name of the ANISN-format cross section library	R	32
mesh_x , etc.	Coordinates of mesh voxel edges in the <i>x</i> , <i>y</i> , and <i>z</i> dimensions, in cm	R	30
denovo_x_blocks , etc.	Number of spatial domain partitions in the <i>x</i> and <i>y</i> dimensions, and the number of pipelining blocks in the <i>z</i> dimension		34
denovo_quadrature	Type of quadrature set. See also: <i>denovo_quad_num_azi</i> , <i>denovo_quad_num_polar</i> , <i>denovo_quad_order</i> , <i>denovo_quad_polar_axis</i>		34
denovo_pn_order	Order of Legendre scattering-angle expansion		37
denovo_first_group , denovo_last_group	The first and last energy groups to be solved		39
Context: model mcnp			
mcnp_input	Filename of the MCNP5 input file	C	25
mcnp_tallies	MCNP tally numbers for which to generate variance reduction parameters and/or to use in forming adjoint sources.	C	25
mcnp_target_sb_density	Target average number of samples per SDEF distribution bin when generating biased source probabilities.		70
Context: model sword			
sword_input	Filename of the .sword file	C	27
Context: method fwcadis			
fwcadis_spatial_treatment	Spatial treatment used in forming the FW-CADIS adjoint source.		29
fwcadis_response_weighting	If True, form the adjoint source to accelerate an energy-integrated response (e.g., dose or total flux)		29
Context: method dx			
dx_forward , dx_adjoint	If True, executes a deterministic calculation in the specified transport mode		30



Exnihilo Documentation

Release 5.4 (Dev)

Seth Johnson

Tom Evans

Greg Davidson

Steven Hamilton

Tara Pandya

July 14, 2015

CONTENTS

I Overview	1
1 Introduction	3
1.1 Package structure	3
1.2 Development team	3
2 Exnihilo Quick Start Guide	5
2.1 Building the developer documentation	5
2.2 Assembling the source code	6
2.3 Building the code	6
3 Software Testing and Verification	7
II Developer Guide	9
4 Introduction	11
5 Configuration Management	13
5.1 Getting the code	13
5.2 Compilers and platforms	14
5.3 Installation toolchain	14
5.4 Third party libraries	15
6 Exnihilo Installation Guide	17
6.1 Installation environment setup	17
6.2 Installation examples	18
6.3 Build steps	21
6.4 The install.sh script	23
7 Workflows	27
7.1 Configuration workflows	27
7.2 Design and implementation workflows	27
7.3 Deployment workflow	28
7.4 Reviews	28
8 Coding Standards	31
8.1 Organizing and writing code	31
8.2 Use of language features	34
8.3 Things to avoid	37
8.4 C++11 usage	39

9 Testing	41
9.1 C++ unit tests	41
9.2 Python unit tests	44
10 Development Environment	47
10.1 Setting up the Exnihilo development environment	47
10.2 Command line tools	48
10.3 Text editor environments	49
11 Useful tools	51
11.1 Git	51
11.2 Git annex	54
11.3 iPython notebook	57
III User Guide: Omnibus	59
12 Front End Interface	63
12.1 Running Omnibus	63
12.2 Omnibus input and output files	64
12.3 Running Omnibus manually	67
12.4 Command line tools	67
12.5 Advanced execution through Python	70
12.6 Parameter list explanation	71
12.7 Developer notes	71
13 Omnibus ASCII Input Format	73
13.1 Blocks	73
13.2 Cards	73
13.3 Other features	74
14 Omnibus Input Database Specification	75
14.1 Omnibus input file	75
14.2 [PROBLEM]	76
14.3 [RESPONSE]	77
14.4 [RESPONSE=histogram]	77
14.5 [RESPONSE=interpolated]	78
14.6 [TALLY]	79
14.7 [SOURCE]	88
14.8 [SOURCE=separable]	88
14.9 [SOURCE=fissionmesh]	96
14.10 [SOURCE=mesh]	99
14.11 [SOURCE=mcnp]	99
14.12 [GEOMETRY]	100
14.13 [GEOMETRY=mcnp]	100
14.14 [GEOMETRY=scale]	102
14.15 [GEOMETRY=rtk]	103
14.16 [GEOMETRY=mesh]	103
14.17 [GEOMETRY=sword]	104
14.18 [COMP]	105
14.19 [PHYSICS]	106
14.20 [PHYSICS=smg]	106
14.21 [PHYSICS=sce]	111
14.22 [PHYSICS=void]	117
14.23 [DEPLETION]	118

14.24 [SHIFT]	127
14.25 [DENOVO]	133
14.26 [MANUALWW]	144
14.27 [RUN]	144
14.28 [RUN=serial]	144
14.29 [RUN=mpi]	144
14.30 [RUN=pbs]	145
14.31 [RUN=titan]	148
15 Postprocessing	151
15.1 Cylindrical tally postprocessing	151
16 omnibus.converters package	153
16.1 omnibus.converters.mctal module	153
16.2 omnibus.converters.meshtal module	156
16.3 omnibus.converters.monaco module	157
16.4 omnibus.converters.opus module	158
16.5 omnibus.converters.triton module	158
16.6 omnibus.converters.vesta module	158
16.7 Module contents	158
17 omnibus.postprocess package	159
17.1 Submodules	159
17.2 omnibus.postprocess.celltally module	159
17.3 omnibus.postprocess.collisions module	162
17.4 omnibus.postprocess.compositions module	163
17.5 omnibus.postprocess.cyltally module	164
17.6 omnibus.postprocess.depletion module	166
17.7 omnibus.postprocess.field module	169
17.8 omnibus.postprocess.fissionsite module	172
17.9 omnibus.postprocess.history module	173
17.10 omnibus.postprocess.kcode module	174
17.11 omnibus.postprocess.manager module	176
17.12 omnibus.postprocess.meshtally module	177
17.13 omnibus.postprocess.pathlength module	179
17.14 omnibus.postprocess.rst module	179
17.15 omnibus.postprocess.sensitivity module	179
17.16 omnibus.postprocess.tally module	179
17.17 omnibus.postprocess.utils module	184
17.18 Module contents	185
IV User Guide: Insilico	187
18 Introduction	189
V Appendices	195
19 Shift Acceptance Test Descriptions	197
19.1 Leakage test	197
19.2 Transmission tests	197
19.3 Material scaling test	197
20 Denovo Acceptance Test Descriptions	199

20.1	Adjoint SN transport test	199
20.2	Deterministic first-collision test	199
20.3	Symmetry test	199
20.4	Two-dimensional transport test	199
20.5	AMGX plotting and XS generation	199
21	Style Guide	201
21.1	Code/Package/Library name capitalization	201
21.2	Commonly used abbreviations/acronyms	201
21.3	Commonly Used Terminology	202
22	License Information	205
22.1	Trilinos	205
22.2	Google Test	205
23	Acknowledgments	207
23.1	Copyright	207
Python Module Index		209
Index		211

Part I

Overview

INTRODUCTION

Exnihilo is a modern radiation transport framework that implements a variety of advanced solvers and solution methodologies, enabling it to solve a wide variety of nuclear engineering and applications problems with the scalability to run on both desktop machines and leadership-class supercomputers.

It supports “stand-alone” execution using its internal front ends, but its components (such as the Denovo S_N solver) are also integrated into other radiation transport codes.

1.1 Package structure

Exnihilo currently contains the following source code packages:

Nemesis: Infrastructure components This collection of utilities should be applicable to most scientific codes. Included are Design-by-Contract™ utilities, communication libraries, containers and algorithms, HDF5 and Silo interface wrappers, template-based serialization utilities, and the unit testing harness.

Transcore: Transport core components These components are specific to radiation transport and multiphysics codes. Included are cross section storage classes, libraries for reading and writing cross sections, quadrature sets, Trilinos solver wrappers, Monte Carlo samplers, and Python interface wrappers.

Geometria: Geometry packages The geometries in this class are used for meshing deterministic problems, for transporting Monte Carlo particles on, and for tallying. Geometries include mesh and cylindrical mesh, Reactor ToolKit geometry, MCNP geometry using the Lava wrapper, SCALE geometry, and DagMC geometry.

Physica: Physics packages These are geometry-agnostic physics engines used for Monte Carlo transport. Currently included are continuous-energy and multigroup physics packages.

Denovo: Deterministic transport solvers Denovo contains advanced transport solvers for fixed-source and eigenvalue problems, with discrete ordinates (S_N) and simplified spherical harmonics (SP_N), using Cartesian grids with the KBA decomposition.

Shift: Monte Carlo solver The Shift Monte Carlo framework is based on geometry- and physics-agnostic transport routines. These routines include advanced parallel algorithms, flexible tallies, and extensible source definitions.

Insilico: Neutronics front end The Insilico front end couples Denovo and Shift with cross-section processing, depletion, and thermo-hydraulics feedback for reactor analysis. This component is integrated into **VERA**, the Virtual Environment of Reactor Applications.

Omnibus: General front end Omnibus is an ASCII-driven front end to Exnihilo for general applications.

1.2 Development team

The core Exnihilo development team consists of the following ORNL scientists (listed alphabetically):

- Greg Davidson <davidsongg@ornl.gov>
- Tom Evans <evanstm@ornl.gov>
- Steven Hamilton <hamiltonsr@ornl.gov>
- Seth Johnson <johnsonsr@ornl.gov>
- Tara Pandya <pandyatm@ornl.gov>

Additional developers that have contributed to the code base to varying degrees and are active *associate developers* of Exnihilo are:

- Cihangir Celik <celikc@ornl.gov>
- Kevin Clarno <clarnokt@ornl.gov>
- Wayne Joubert <joubert@ornl.gov>
- Chris Perfetti <perfetticm@ornl.gov>
- Rachel Slaybaugh <slaybaugh@berkeley.edu>

Emeritus developers, who have moved on to other projects, include:

- Joshua Jarrell
- Brenden Mervin
- Stuart Slattery

General questions about Exnihilo software and methods should be directed to the [email list](#).

EXNIHILO QUICK START GUIDE

2.1 Building the developer documentation

Documentation in Exnihilo comes in three forms:

METHODS Papers, reports, and notes that describe numerical methods, algorithms, and results. These documents are prepared using the `latex` document system.

CODE MANUALS Manuals for code developers, the development environment, and users. This document falls into this category. We use the `Python Sphinx` system to produce multiple document formats (PDF, HTML, text, etc) from basic `reStructured (rst)` formatted markup.

CODE DOCUMENTATION Inline code documentation. This class of documentation is almost exclusively reserved for code developers. It is processed using the `Doxygen` inline code documentation system from comments that live within the source code.

Details on building **METHODS** and **CODE DOCUMENTATION** are found in *Developer Guide*. To build the full developer documentation `Python` must be available with the `Python Sphinx` module installed. Enter `Exnihilo/doc/manual` and run `make`:

```
$ make
Please use `make <target>' where <target> is one of
  html      to make standalone HTML files
  dirhtml   to make HTML files named index.html in directories
  singlehtml to make a single large HTML file
  pickle    to make pickle files
  json      to make JSON files
  htmlhelp  to make HTML files and a HTML help project
  qthelp    to make HTML files and a qthelp project
  devhelp   to make HTML files and a Devhelp project
  epub     to make an epub
  latex    to make LaTeX files, you can set PAPER=a4 or PAPER=letter
  latexpdf  to make LaTeX files and run them through pdflatex
  text     to make text files
  man      to make manual pages
  texinfo   to make Texinfo files
  info     to make Texinfo files and run them through makeinfo
  gettext   to make PO message catalogs
  changes   to make an overview of all changed/added/deprecated items
  linkcheck to check all external links for integrity
  doctest   to run all doctests embedded in the documentation (if enabled)
```

Choose the desired format and run `make <format>`. The build result will be in `_build/target`.

2.2 Assembling the source code

Exnihilo integrates with the build system of three external code repositories: copies of Exnihilo, Trilinos, and TriBITS must be located inside of the SCALE source directory in order to build:

```
$ cd /usr/local/src  
$ ls SCALE  
CMakeLists.txt ... Exnihilo TriBITS Trilinos ...
```

See [Getting the code](#) for details.

2.3 Building the code

After the necessary *Third party libraries* have been built and installed, Exnihilo can be built with CMake. The most straightforward way to get started building Exnihilo is to use the scripts in `Exnihilo/install` that are described in [Configuration Management](#). The `install.sh` script described in [The install.sh script](#) is used to manage the build process.

To build the base release of Exnihilo run the following:

```
./install.sh Exnihilo
```

To build only the Exnihilo components needed for ADVANTG, run:

```
./install.sh Exnihilo for-advantg
```

CHAPTER
THREE

SOFTWARE TESTING AND VERIFICATION

As described in [Developer Guide](#), the software in Exnihilo is highly tested. Specifically, every class/component in the code is **required** to have an individually compiled unit test verifying software correctness. This strategy has the side benefit of creating simplified *use cases* for each class that can be used by novice code developers to learn the interface and function of each class in the code.

Naturally, unit testing is only one aspect of a complete testing framework that demonstrates code verification and validation (V&V). First, we define some terms that will establish the frame-of-reference for code activities.

Verification Given a well posed and defined problem, whether the code produces the correct solution within the constraints imposed by the algorithm.

Validation Whether the model problem solved by the software matches experimental data or other accepted standards.

In essence, **Verification** determines that a given problem is being solved correctly; **Validation** determines that the correct problem is being solved. Within the framework of these definitions, most testing in Exnihilo is focused on **Verification**, not **Validation**. We can ensure that the code produces the correct output for a given set of inputs; however, we cannot ensure that the methods are being applied appropriately to a given problem. That information must, by definition, be performed for a given application.

For example, the **Denovo** package contains a simplified harmonics solver (SP_N). This method can have issues in void and near-void regions. Thus, a user wishing to solve transport problems in an application space that includes voids should probably consider using another method. However, the fact that SP_N is inappropriate, or *unvalidated*, for this class of problems does not negate the fact that the **Denovo** SP_N solver is **Verified**, ie. the weakness of SP_N for this class of problems lies with the method and model, not in the manner in which it is solved. In other words, the fitness of a given model, in this case SP_N , to a given problem is determined by **Validation**.

In addition to unit tests, Exnihilo contains multiple *acceptance* tests that compare against reference solutions, either analytical or from other validated codes. These are described in [Shift Acceptance Test Descriptions](#) and [Denovo Acceptance Test Descriptions](#).

Part II

Developer Guide

**CHAPTER
FOUR**

INTRODUCTION

The developer's guide lays out how to obtain Exnihilo, how to install it, and how to successfully develop for it.

1. Read [*Getting the code*](#) to learn how to access the repositories that contain Exnihilo and associated software.
2. Read the [*Exnihilo Installation Guide*](#) guide, installing any TPLs as necessary and then installing Exnihilo itself.
3. Install the Exnihilo [*Development Environment*](#) to set up the environment template and scripts.
4. Watch the [*Lego Movie*](#) and possibly some Sesame Street while everything installs.
5. Get to work!

CONFIGURATION MANAGEMENT

This section describes obtaining and configuring:

Exnihilo and build system Exnihilo must be built alongside **SCALE** reactor analysis criticality tools, **Trilinos** numerical software, and **TriBITS** build tools.

TPLs The third-party toolchain and libraries listed in *Third party libraries* that are used to build, run, and test the code.

Toolchain Software utilities that are used to build, edit, profile, document, and perform configuration management activities.

We provide scripts for building and setting up the TPLs and Toolchain in scripts. See *Configuration Management* for details on using these scripts. Instructions for additional *Useful tools* is available later.

5.1 Getting the code

Exnihilo uses the TriBITS build system, it also has a required dependency on Trilinos and SCALE, requiring all four code systems to be downloaded and built at once. Note that while the existence of these systems is required, only the requested parts of each code will be built.

Note: If building from the ADVANTG or SCALE distributions, all of these source packages are already available and in the correct locations. You may skip to the next section.

5.1.1 Exnihilo

Configuration management in Exnihilo is managed using **git**. The source repository is stored on the ORNL machine `angband.ornl.gov`:

```
git clone ssh://angband.ornl.gov/repos/git/Exnihilo.git Exnihilo
```

You will need to contact the Exnihilo Team to get access to the repository.

5.1.2 SCALE

It is also necessary to download SCALE to build Exnihilo. SCALE is managed using Mercurial and is cloned using:

```
hg clone https://fogbugz.ornl.gov/kiln/RepoAlias/scale/scale Scale
```

For access to the RNSD fogbugz server, contact [Jordan Lefebvre](#). To clone from fogbugz, you might also need to add the following to your `~/.hgrc`

[hostfingerprints]

```
fogbugz.ornl.gov = 39:81:1f:f4:82:5e:2a:64:7c:e2:6e:33:16:30:7d:be:43:e4:d3:3f
```

Note: If building for ADVANTG or Denovo solver development, it is possible to use a skeleton SCALE build system in lieu of the SCALE download above. This small “SCALE” directory is distributed with ADVANTG and only contains the essential CMake components of SCALE. (See [make-skeleton-scale](#) for details of the tool used to build it.)

SCALE data (necessary for running Insilico, Shift with CE, and other parts of the code) can be copied from the RNSD servers over SCP. We provide an [update-scale-data](#) tool to simplify loading and updating the data.

5.1.3 Trilinos and TriBITS

Finally, Trilinos and the TriBITS build system can be cloned from the angband server if at ORNL:

```
git clone ssh://angband.ornl.gov/repos/mirror/Trilinos.git Trilinos
git clone ssh://angband.ornl.gov/repos/mirror/TriBITS.git TriBITS
```

If access to the ORNL server is unavailable, [Trilinos](#) and [TriBITS](#) can be downloaded or cloned from their respective web sites.

5.1.4 Assembling the repositories

Exnihilo installation is set up with SCALE as the primary “source” directory; the Exnihilo, TriBITS, and Trilinos directories must be moved or symbolically linked inside it:

```
cd Scale
ln -s ../TriBITS
ln -s ../Trilinos
ln -s ../Exnihilo
```

5.2 Compilers and platforms

Currently, Exnihilo is tested on Linux and Mac OS X systems using both Intel and GCC compilers. Intel 14.0 and GCC 4.6 are the oldest supported versions due to the C++11 support requirement. A limited build of Exnihilo has been tested on Windows, but Windows is currently only supported as far as is required to run SCALE. Exnihilo is also compatible with Clang (i.e., the LLVM compiler on Apple systems), using `gfortran` to compile the Fortran code in Trilinos, Exnihilo, and SCALE.

5.3 Installation toolchain

To configure, install, and run all the components of Exnihilo, several tools need to be installed.

Table 5.1: Exnihilo toolchain

Tool	Required	Comments
CMake	Yes	Build system used by TriBITS to generate Makefiles.
Python	Not exactly	Needed for some advanced software configuration, for Omnibus, and for Python front end and tools.
SWIG	No	Needed for Python front end, with PCRE as a prerequisite TPL.

5.4 Third party libraries

Exnihilo makes use of several Third Party Libraries (TPLs). TPLs are distinct from *toolchain* components that are used to build, debug, profile, and test the code (e.g. the [GCC](#) compiler and [Python](#)). Libraries provide features to the compiled code. Examples include BLAS/LAPACK and HDF5. Some TPLs are provided both libraries and toolchain components, e.g. [MPI](#). The Exnihilo toolchain is described in [Development Environment](#). The following table gives a listing of the TPLs used by Exnihilo.

Table 5.2: Exnihilo TPLs

TPL	Required	Comments
MPI	No	OpenMPI and MPICH are suggested options. Vendor-provided options will also work.
QT	No	This is required by Scale. Any Exnihilo dependencies that require Scale will require QT .
BLAS/LAPACK	Yes	Use vendor-specific implementations when possible. The default implementations can be found at netlib .
HDF5	No	Both serial and parallel HDF5 implementations are supported. When building serial HDF5, parallel I/O using HDF5 is not available.
Silo	No	HDF5 is required when using Silo .
Lava	No	ORNL-developed interface library to MCNP services. Requires access to the MCNP source. Contact the Exnihilo team for access.

In addition to the above TPLs, Exnihilo supports many TPLs that are supported by Trilinos (e.g. SuperLU) solvers. See the [Trilinos build manual](#) for details.

Generic directions for building the TPLs are provided in [Development Environment](#). Specific instructions for systems that differ from the general instructions are provided in [Exnihilo Installation Guide](#).

5.4.1 Python TPLs

The following Python package are also recommended:

Table 5.3: Exnihilo Python packages

TPL	Required	Comments
Numpy	No	Advanced Python numeric manipulation, used by Python tools and Omnibus postprocessing.
h5py	No	HDF5 reading and writing in Python.
pandas	No	Python Data Analysis Library, used in some Omnibus postprocessing.

EXNIHILO INSTALLATION GUIDE

This chapter documents a complete, from-scratch installation of Exnihilo. Most users will not need to read or understand this chapter, and even most developers (assuming they're on a pre-configured server) will not most of this reference. We provide detailed steps for most of the typical use cases.

6.1 Installation environment setup

Included in the Exnihilo/install directory is a set of build scripts developed to build Exnihilo, ADVANTG, and related prerequisites on various Linux and Mac systems. As the number of permutations of systems and codes and options has increased, the compilation scripts were restructured to allow for more flexibility. The file listing is:

codes cmake configurations and install scripts for relevant codes
rc environemnt settings for each platform/system (see *Install environment description*)
tools collection of scripts used for building and running regression tests
install.sh generic installer driver (see description and options below)
ctest-driver.sh testing/install driver for SCALE/Exnihilo
README.rst this readme file
setup_macports.sh recommended initial MacPorts installation script (see *Build toolchain with MacPorts*)

The recommended install process for a new system starts by setting up the installation environment for your machine:

```
cd Exnihilo/install/rc
cp -R $(uname) $(hostname -s)
cd $(hostname -s)
$EDITOR base.sh
```

Note: On my mac (Darwin system) with hostname `sierrajuliet`, the above simply evaluates to:

```
cp -R Darwin sierrajuliet
cd sierrajuliet
mvim base.sh
```

The scripts inside these files should be modified to reflect the base path for your source files, the path to SCALE, MCNP, and/or ADVANTG data files, etc. For a description of their contents, see *Install environment description*.

6.2 Installation examples

Depending on what system you're using, whence your source code originates, and how customized you need your build, you have several options of what TPLs and tools need to be installed. We give several typical use cases below.

6.2.1 Building with Macports GCC

This section applies to most of the Exnihilo users at ORNL, but it could also be adapted for users with advanced package management systems.

1. *Build toolchain with MacPorts* to install GCC and the most of the TPLs.
2. Install Lava, the only TPL you must compile yourself, if using ADVANTG or Exnihilo with MCNP geometry support:

```
cd Exnihilo/install  
.install.sh lava
```

You can also install other TPLs such as MOAB for CAD geometry support:

```
./install.sh moab
```

3. *Build Exnihilo from source*
4. If using ADVANTG, you can now install it too:

```
./install.sh advantg
```

6.2.2 Building Exnihilo on Titan

The [Titan](#) system uses the module system to load package support. Several packages must be loaded (preferably in your `.bashrc`) for TPLs and the GCC build chain to work:

```
module load gcc  
module load git  
module load mercurial  
module load cmake3  
module load python  
module load cray-hdf5  
module load cudatoolkit  
module load python_numpy  
module load python_h5py  
module load python_matplotlib
```

The Titan system configure directory has variants for each project allocation. These can be used like:

```
./install -t nfi004 Exnihilo scale-debug
```

Note that to run unit tests and executables on Titan, it's necessary to log in to a compute node and execute the process using the `aprun` command.

6.2.3 Building the entire toolchain from scratch

If you're on a brand-new system using outdated software (such as Red Hat, whose compilers are frozen at several years behind the cutting edge for the sake of stability), it's typical to install the entire toolchain manually.

1. *Build GCC from source* to install the compilers.
2. *Build tools from source* for other configuration requirements.
3. *Build BLAS/LAPACK from source* if needed.
4. *Build public TPLs from source* for other Exnihilo requirements.
5. Install Lava and other feature-specific TPLs:

```
./install.sh lava
```

6. *Build Exnihilo from source*
7. If using ADVANTG, you can now install it too:

```
./install.sh advantg
```

6.2.4 Building Exnihilo from an ADVANTG distribution

For this example, we suppose that you are on a Linux system, and you want all of the ADVANTG/Denovo components to live together in /advantg. You might set up an /advantg/source directory for the source files, /tmp/build-advantg to do the actual build, and /advantg/install as the prefix directory.

Assembling the source

The ADVANTG source distribution includes:

- a copy of the current ADVANTG source,
- a copy of the current Exnihilo source,
- a copy of the current Lava library source, and
- a skeleton copy of the SCALE build system.

It is also necessary to download the Trilinos and TriBITS source code in the same directory as these sources.

The Scale directory contains the build system infrastructure but none of the SCALE components. It has the required symlinks to ../Trilinos, ../TriBITS, and ../Exnihilo.

So, to prepare your source directory:

1. `mkdir -p /advantg/source`, then `cd` into it.
2. Expand the ADVANTG/Exnihilo tarball into the current directory.
3. Download and expand both Trilinos and TriBITS into that directory as well.

At this point, if you run `pwd; ls -l`, you should see:

```
/advantg/source
Exnihilo
Scale
TriBITS
Trilinos
advantg
lava
```

If your system lacks a working version of HDF5, Silo, and OpenMPI, you'll need to download those into that directory as well.

Setting up your system environment

Exnihilo provides installation scripts to help automate the build process (and allow faster reinstallation). Enter the Exnihilo/install directory, which is where these install scripts live. Go to the `rc` subdirectory, which is where system-specific configuration profiles reside. This directory contains generic profiles (Darwin, Linux) as well as individual systems configured by Exnihilo users.

To make a profile for your system, copy the generic profile to your system's name:

```
cp -R $(uname) $(hostname -s)
```

For the purposes of this example we'll assume your Linux system name is grumpycat, so effectively we just ran:

```
cp -R Linux grumpycat
```

Now `cd grumpycat` and edit the `base.sh` file. The contents of this file are described in [Install environment description](#), but for now we'll just modify a couple of key variables.

1. Since the base directory for the ADVANTG/Exnihilo source is located at `/advantg/source`, and we have other plans for the build and install directories, we must change the `source_base`, `build_base`, and `prefix_base` variables:

```
export source_base=/advantg/source
export build_base=/tmp/build-advantg
export prefix_base=/advantg/install
```

2. It's likely also that you are using a specific version of GCC, and you might have existing installations of required third party libraries (TPLs, such as Silo and HDF5). You'll want to modify `CC`, `CXX`, etc. to reflect your compilers, and you might also want to adjust your `LD_RUN_PATH` and `PATH` inside the script. For CMake to automatically locate a library installation, you can set the `CMAKE_PREFIX_PATH`. You'll want to update your prefix path to include the Lava installation being used (see the following tip).
3. Finally, suppose the ADVANTG multigroup libraries live in `/advantg/data`. You will then have to set the `anisn_path` variable:

```
export anisn_path=/advantg/data/anisnfmt
```

Tip: As an example of how to use the `CMAKE_PREFIX_PATH`, suppose we have an HDF5 installation at `/opt/hdf5`, where `/opt/hdf5/bin/h5dump` is a binary, `/opt/hdf5/lib/libhdf5.so`, and `/opt/hdf5/include/hdf5.h` is a header file. Your `CMAKE_PREFIX_PATH` would then be set as:

```
export CMAKE_PREFIX_PATH=/opt/hdf5
```

and then CMake will be able to automatically detect the HDF5 installation when needed.

If you're installing your own version of Lava, MPI, hdf5 and silo with the Exnihilo install scripts, you should set:

```
export CMAKE_PREFIX_PATH
CMAKE_PREFIX_PATH=${prefix_base}/lava
CMAKE_PREFIX_PATH+=:${prefix_base}/openmpi
CMAKE_PREFIX_PATH+=:${prefix_base}/silo
CMAKE_PREFIX_PATH+=:${prefix_base}/hdf5
```

Building and installing

With your environment set up, you can now install lava, Exnihilo, and ADVANTG. Return to the Exnihilo/install directory.

First, you may need to install OpenMPI, HDF5, and Silo if they're not already present:

```
./install.sh openmpi
./install.sh hdf5
./install.sh silo
```

Then you'll want to install the Lava library:

```
./install.sh lava
```

Next, you can install Exnihilo using the `for-advantg` variant (which is located at `Exnihilo/scripts/codes/Exnihilo/for-advantg.cmake`):

```
./install.sh Exnihilo for-advantg
```

Tip: If the above installation fails for some reason, you can either modify the `for-advantg.cmake` script yourself, copy it to a new variant, etc. In these scripts, all CMake options must be set using the `CACHE` option to propagate the variables to CMake.

Once Exnihilo is installed, ADVANTG can be installed:

```
./install.sh advantg
```

and this completes the build process. See the *The `install.sh` script* section for details on the installer script.

For any of these steps, you of course have the option of manually creating a CMake “configure” script and running in your own build directories. Note, however, that building Exnihilo is done inside the SCALE build system, so SCALE needs to be set as the “source” directory in order to build Exnihilo.

6.3 Build steps

To install Exnihilo, it is often best to start building the compilers and third party libraries (TPLs) from scratch: the Python front-end `pykba`, because it relies on shared libraries, requires that all the prerequisite libraries be compiled with position-independent code (`-fPIC` option for GCC). This is most easily accomplished by building libraries as shared to begin with.

Installation works best with a recent build of the GNU compiler collection (gcc). On a Mac system, this is most easily accomplished using the MacPorts package distribution system. On old Linux systems (e.g. RHEL 4), it is best to download the prerequisite GCC libraries, build them, build and install GCC, and then install the remaining packages. Newer Linux systems (e.g. Ubuntu) often have up-to-date installations of GCC as well as package managers that can ease the installation process.

6.3.1 Build GCC from source

If building GCC from scratch on a Linux box, download these libraries into your source base directory (e.g. `/usr/local/src`):

- `mpfr` : GNU MPFR Library
- `gmp` : The GNU Multiple Precision Arithmetic Library
- `mpc` : The multiprecision library
- `gcc` : GNU compiler collection

To install GCC using the `bootstrap.sh` configuration you created:

```
./install.sh -t bootstrap gmp
./install.sh -t bootstrap mpfr
./install.sh -t bootstrap mpc
./install.sh -t bootstrap gcc
```

This builds a current version of GCC using whatever older version is available on the system.

6.3.2 Build tools from source

If a good package manager is unavailable on your system, you'll have to download and install the CMake, SWIG, and Python tools.

- `cmake` : CMake build system
- `pcre` : PCRE - Perl Compatible Regular Expressions
- `swig` : Simplified Wrapper and Interface Generator (SWIG)
- `python` : Python 2

After downloading, you can install these simply with:

```
./install.sh cmake
./install.sh pcre
./install.sh swig
./install.sh python
```

6.3.3 Build BLAS/LAPACK from source

Most systems include a compiled linear algebra library. (On Titan, these are embedded in the Cray scientific libraries; on a Mac and most Linux systems, they're located in `/usr/lib`.) If your machine does not, or you wish to have an autotuned library for your specific machine, you'll need to install:

- `atlas` : Automatically Tuned Linear Algebra Software (ATLAS)
- `lapack` : LAPACK linear algebra library

Place the compressed LAPACK source `.tgz` alongside the ATLAS source in your source directory, and execute:

```
./install.sh ATLAS
```

This will install ATLAS BLAS functions and supplement them with the LAPACK routines.

6.3.4 Build public TPLs from source

Several third party libraries (see *Third party libraries*) are necessary or recommended to build and run Exnihilo. These can be downloaded with the following links:

- `openmpi` : OpenMPI
- `hdf5` : HDF5 scientific format library
- `silo`: Silo data output library
- `numpy` : numeric python library (optional but strongly recommended)

If installing with SCALE enabled (not needed for ADVANTG), QT is required:

- `qt`: large application and UI framework

To install these, simply execute:

```
./install.sh openmpi
./install.sh hdf5
./install.sh silo
./install.sh numpy
```

6.3.5 Build toolchain with MacPorts

MacPorts allows almost the entire toolchain to be downloaded and built on a Mac. You must first download the [MacPorts](#) disk image and run the included installer to get MacPorts (a package manager for the Mac platform).

To install a recent version of GCC and most of the required TPLs and tools, we have provided a script:

```
sudo xcodebuild -license
sudo ./setup_macports
```

You may want to modify/trim the list of ports in that script: it installs the Macports X11 and several other components that may be redundant for your system configuration. It also installs [iPython notebook](#) and other tools.

6.3.6 Build Exnihilo from source

The Exnihilo source requires three different repositories (SCALE, TriBITS, Trilinos) to build: Exnihilo, TriBITS, and Trilinos must be symlinked or moved inside the SCALE base directory. See [Getting the code](#) for detailed directions on obtaining and assembling these source codes.

Once the source is in place in your source directory, Exnihilo can be installed:

```
./install.sh Exnihilo {variant}
```

Here, [variant] is the specific set of cmake options passed to the configure. h{are:

- empty, for a typical server installation meant for users of Shift or Denovo;
- scale-debug, for development with SCALE support;
- for-advantg, if installing just for use in ADVANTG;

etc. (See the `install/codes/Exnihilo/*.cmake` and other `cmake` files in its subdirectories.)

6.4 The `install.sh` script

This script automates the process of “sourcing” an rc file for a particular system, setting up installation directories, and running cmake/build/install. It will execute a combination of system (with optional “system variant”) and code (with optional “code variant”). By default it will look for `rc/${hostname}`, or if unavailable `rc/${uname}`. It looks for install scripts or CMake configurations inside the `codes` directory, first under `codes/${code}/${hostname}` and next under `codes/${code}`. The ability to find options under a hostname allows individual build configurations for each user of a code.

6.4.1 Install environment description

Inside the `config.sh` files are a number of environment variables used by the scripts and installers that determine the installer paths, compilers, etc.

source_base This should be the root directory in which all the source directories are stored. For example, on most systems I use /usr/local/src. Inside the src directory I have Exnihilo, openmpi-x.x.x, hdf5-x.x.x, etc.

prefix_base This is the default parent directory of all the installed TPLs and Exnihilo. For example, if your prefix is set to /usr/local/denovo, you'll end up with .../local/denovo/openmpi, .../local/denovo/Exnihilo, etc.

build_base This is the parent directory for the temporary builds. These files don't need to be retained after Exnihilo is installed, so it's often a good idea to set a directory in /tmp as the build root, because disk access speeds are often much faster (being on a local drive rather than a network mount for some clusters).

CC, CXX, F90 These are the default compilers.

mcnp_src Deprecated: older versions of Lava required the MCNP source code to build.

mcnp_exe This is the actual MCNP executable, used for generating runtpe files.

anisn_path This is the data path to the ANISN-formatted files used by ADVANTG. It's not necessary if SCALE is all that's being installed.

scale_data_path This is the path to the SCALE data directory.

6.4.2 Example of using install.sh

Example on host sierrajuliet:

```
install.sh lava
```

will source rc/sierrajuliet/base.sh, which is a symbolic link to the Darwin/macports.sh resource. It will then build from the /usr/local/src/lava directory (the source base directory is specified in the RC file), create a build directory, and run cmake using the options specified in codes/lava/base.cmake. It will then start building and try to install.

Another example of installing a “code variant” (e.g. a custom debug build):

```
install.sh -t dev Exnihilo shift-release
```

will source rc/MACHINE/dev.sh rather than rc/MACHINE/base.sh; it will then run CMake based on codes/Exnihilo/shift-release.cmake.

There are also command line options for explicitly specifying the source, build, and install directories.

If you want to create custom source/install paths for your particular machine, simply create an rc/HOSTNAME directory and add scripts to it.

If the software package you're installing does not support CMake, you can create an install_CODE.sh file inside codes.

Tip: Unique builds can be specified by adding a machine variant. An example is the variant for machine *mirkwood*. Looking in the rc/mirkwood directory we see the system variant debug. To do a debug build on mirkwood in /home/me/build and install in /home/me/debug from source in /home/me/Exnihilo:

```
./install.sh -p /home/me/debug -b /home/me/build \
             -c /home/me/Exnihilo -t debug Exnihilo
```

Note: It is not necessary to specify the system unless you want to override the default. For example, you could use the mirkwood variant on another Mac OS X system by running:

```
./install.sh -p /home/me/debug -b /home/me/build \
    -c /home/me/Exnihilo -s mirkwood -t debug Exnihilo
```

This will use the Exnihilo base options with overrides defined in `rc/mirkwood/debug.cmake`.

6.4.3 Power user explanation

Having a script that initializes a CMake build to create a Makefile to build a piece of software is necessarily confusing. The `install.sh` script is at its essence the following script:

```
scripts_dir=$(pwd)
source rc/Darwin/macports.sh
export prefix_dir=${prefix_base}/Exnihilo
cd ${build_base}/Exnihilo
cmake -C ${scripts_dir}/codes/Exnihilo/base.sh \
    ${source_base}/Exnihilo
```

Instead of using the `-C` option to read the CMake configure file, you may define options on the command line:

```
cmake \
    -D Exnihilo_DISABLE_SCALE:BOOL=ON \
    -D SCALE_ENABLE_GeometriaLava:BOOL=ON \
    -D SCALE_ENABLE_DenovoPyKBA:BOOL=ON \
    -D SCALE_ENABLE_DenovoPySPN:BOOL=ON \
    -D CMAKE_BUILD_TYPE:STRING="Release" \
    -D ENABLE_PYTHON_WRAPPERS:BOOL=ON \
    -D ENABLE_DOCUMENTATION:BOOL=ON \
    -D SCALE_ENABLE_SWIG_EXCEPTIONS:BOOL=ON \
    -D PyKBA_ENABLE_ALL_EQUATIONS:BOOL=ON \
    -D SCALE_ENABLE_ALL_FORWARD_DEP_PACKAGES:BOOL=OFF \
    -D SCALE_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=ON \
    -D SCALE_ENABLE_SECONDARY_TESTED_CODE:BOOL=ON \
    -D BUILD_SHARED_LIBS:BOOL=ON \
    -D SCALE_ENABLE_CXX11:BOOL=ON \
    -D CMAKE_INSTALL_PREFIX:PATH="/usr/local/exnihilo" \
    Exnihilo
```

6.4.4 Command line options

- c** dir
Manually specify the source directory name.
- b** dir
Manually specify the build directory name.
- p** dir
Manually specify the install directory (i.e. prefix) name.
- s** sysname
Override the system name to load a custom RC file and cmake script.
- t** variant
Specify a system variant, loading `rc/${sys}/${variant}.sh` as the resource file.
- m**
Reuse make file if possible.

-n

Do not build or install. This option only generates the cmake configuration.

-u

Use the ctest driver to upload and run unit tests. This option is only valid when building Exnihilo or SCALE.

WORKFLOWS

7.1 Configuration workflows

Exnihilo uses `git` for configuration management. Feature development is performed on *topic* branches. After review branches are merged into the master versions. We follow the basic tenets described in the `cmake-git` workflow strategy. In short, make a local *tracking* branch in your repo that diverges from the master branch:

```
$ git branch
* master
$ git checkout -b my-branch
```

Do your work and only merge back to the master when **all** unit tests pass. Feel free to commit often on the topic branch (including broken code). The objective is to only merge **correct** code back to the master. (Code on branches in a developmental state is allowed to be temporarily broken.)

Once the topic branch is completed, and all unit tests pass, do a quick code review with an Exnihilo team member. To merge the topic branch into the master, do the following:

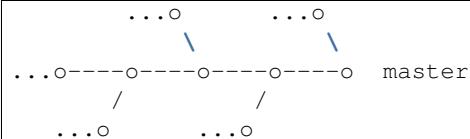
```
$ git checkout master
$ git pull
$ git merge --no-commit my-branch
```

For details and recommendations on merging and conflict resolution, it is highly recommended that the *Merging and conflict resolution* section be consulted.

Upon successfully completing the merge, you can push to the origin:

```
$ git push origin
```

Using this basic workflow will create a clean repository in which commits are descriptively labeled, and it will be easy to walk backwards through the history:



7.2 Design and implementation workflows

7.2.1 Design workflow

Exnihilo software design and implementation is performed using a 3-step process:

1. Documentation of numerical methods and/or algorithms to be implemented in a RNSD Technical Note:

```
nemesis-note -b title
```

this will generate a LaTeX technical note template.

2. Design model sketching using either UML or descriptive text; these are not formally archived.
3. Implementation with inline documentation using **doxygen**, which serves as a formal documentation of the design.

This process is highly agile and iterative. In some cases, steps 1 & 2 may be highly contracted or skipped altogether depending on the task. Step 3 can be highly iterative; thus we have found repeated updating of preliminary design sketching to be very disruptive. Informal reviews are performed after each step (see [Reviews](#) for details on reviews). Depending upon the impact/scope of the implementation, a formal review may be performed at the conclusion of the activity. The documented code is the ultimate, persistent design. All code should be self-documenting according to the coding standards described in [Coding Standards](#).

7.2.2 Development workflows

Exnihilo is developed using an *agile* configure/edit/build/test cycle, often using paired-programming techniques. The *Trilinos Build System* **cmake** extensions are used to control code builds, testing, and deployment. A standard development workflow is:

1. Create topic branch
2. Configure the code using cmake
3. Edit the code (perhaps as part of pair-programming or multi-programming) and corresponding unit tests
4. Build code and tests
5. Run tests Repeat until full feature is implemented
6. Review topic branch
7. Merge topic branch onto master
8. Full build + downstream tests
9. Push to server

7.3 Deployment workflow

Exnihilo is developed using a test-driven, continuous-integration process. Thus, each version of master can be successfully deployed as a release because it successfully runs all unit and acceptance tests.

7.4 Reviews

The decision upon whether to perform a *formal* or *informal* review is up to the Exnihilo team. Generally, factors such as complexity, importance, and difficulty determine whether a formal or informal review needs to be performed. The objective of reviews is to identify issues as close to implementation as possible; this dramatically reduces defect resolution times and minimizes adverse impacts across the project. The majority of reviews in Exnihilo are *informal*.

7.4.1 Formal reviews

Formal reviews are conducted by a review panel. The size, scope, and members of the panel are determined by the item being reviewed. The following table gives recommended numbers and team composition for different types of reviews.

Reviewed Item	Panel Size	Team Composition
Code walk-through	1–2	team members only
Code review	2–3	may include an external member
Design review	2–3	should include an external member
Requirements review	2–3	should include an external member, preferably a project stakeholder
Methods review	2–3	should include an appropriate subject matter expert
User and developer manuals	2–3	should include an external member, preferably a user

This is not an exhaustive list as many activities outside of these areas may require review. Use this information as a guide for such activities.

The requirements for formal review are:

- review minutes should be documented using **iPython Notebook** and archived in the Exnihilo documents repository (`ssh://angband/repos/git/documents.git`) under `reviews`.
- reviews must be moderated and have a time-limit
- issues should be discussed, not solutions

Review minute documentation need not be exhaustive or comprehensive. Important issues and outstanding action items should be noted.

7.4.2 Informal reviews

Informal reviews take nearly any form. The most effective format is a *walk-through*. This type of review has the creator of an artifact review the item with another person. The other person is usually a team member. No formal documentation or archiving is required for informal reviews.

CODING STANDARDS

This chapter gives the coding standards used in Exnihilo. Most of the software in Exnihilo is be written in C++. The power of C++ can be abused easily, resulting in code that is difficult to understand and maintain. This document gives the practices that must be followed on the Exnihilo project for all new code that is written. The intent is not to be onerous but to ensure that the code is solid and maintainable. As of June 1, 2014, the Exnihilo code base will be integrating C++11 constructs into the code base. Accepted C++11 practices are described in [C++11 Usage](#).

External libraries that Exnihilo uses do not have to meet these requirements, although we encourage external developers to follow these practices. For any code that Exnihilo takes ownership, the project will decide on a case-by-case basis on any changes.

8.1 Organizing and writing code

This section deals mainly with how code should be organized and written in terms of files, file names, code formatting, and documentation.

8.1.1 File names

Each class is defined within its own set of files, as summarized in the following table:

Table 8.1: Exnihilo TPLs

File	Description
A.hh	<i>Header file.</i> Contains definition of class A. It may also contain member function definitions, although preferably, function definitions should be in one or more of the files A.cc, A.i.hh, and A.t.hh.
A.cc	<i>Implementation file.</i> If A is non-templated, then contains non-templated member function definitions of A. If A is templated, contains member function definitions of specializations of A.
A.t.hh	<i>Template implementation file.</i> May be used if A is a templated class, or if A contains templated member functions. In these cases, contains the corresponding function definitions which will be explicitly instantiated by A.t.cc.
A.i.hh	<i>Implementation file for member functions.</i> This file should always be included at the bottom of A.hh. If A is templated, or has templated member functions, then this file can be used for an automatic instantiation model. For non-templated entities, this file may contain inline function definitions.
A.pt.cc	<i>Instantiation file.</i> Used for explicit template instantiation of the definitions in A.t.hh. For specific template arguments, contains instantiations of A or its templated member functions.
test/tstA.cc	<i>Unit test file.</i> The unit test for A. Other files may also be used by the unit test.

Note: Multiple classes should not be defined in a single set of files, except in very special circumstances where the

classes are closely related (for example, nested classes, or an iterator class for a container).

8.1.2 Generating files

The Exnihilo *Development Environment* installs a `template-gen` tool that creates templates for various file types. To generate files for a new templated class called `Foo`, you might do the following:

```
$ template-gen Foo.{hh,t.hh}
>>> Sucessfully created Foo.hh
>>> Sucessfully created Foo.t.hh
$ template-gen test/tstFoo.cc
>>> Sucessfully created test/tstFoo.cc
```

This will create the class header, the template definition file, and a unit test template.

8.1.3 Template model

In Exnihilo, there are two models that are used for template instantiation, based on usage:

Automatic This is for templated classes for which it is unlikely that the template arguments are known beforehand. Examples are containers and smart pointers (which can contain or point to any data type). These classes include their function definitions within their header file (`*.hh`, either explicitly or via a `*.i.hh` file. Hopefully, to avoid code bloat and excessive compile times, these classes are small.

Explicit This is for templated classes for which the range of template arguments is known. An example is a finite-volume class that is templated on the mesh type (the number of mesh types one typically requires is known and fairly small). Here, the function definitions (`*.t.hh`) are included by the template instantiation file (`*.t.cc`). The instantiation file instantiates the class for each template argument that is desired.

Note that the STL follows the automatic instantiation model.

8.1.4 Write unit tests

The author of a class must write a unit test for that class. The unit test not only tests the functionality of the class, but also helps serve as an example for how to use the class. Some authors actually prefer to write the test before the class. See [File Names](#) for where the unit test should reside.

8.1.5 Syntax names and code formatting

There are only two hard things in Computer Science: cache invalidation and naming things.

—Phil Karlton

Rules for names and code formatting can be onerous. We believe many such rules are based more on personal preference and do not significantly add value to a code's readability. If too many rules are specified, teams tend to ignore all the rules, or become unhappy with enforcement. The intent here is to have code that is quickly understandable by anyone on the Exnihilo team. But individual team members may not find each other's style "pretty."

Hint: With regards to code style, the most important thing is to be neat and consistent. Your code will be read by other team members, so readability is important. Code that is sloppy will not be accepted into the head version of the code. Try to follow existing source code conventions when possible.

Important: The most important stylistic guideline imposed by the Exnihilo code team (without exception) is an 80 column limit on statement lines. Please limit all code statements to no more than 80 columns.

Consequently, we have narrowed down the list of rules to those we believe are the most important (in addition to the inviolate 80 column limit mentioned above):

1. Indent your code as follows:
 - Do not indent curly braces relative to their control statements.
 - Do not indent namespace blocks.
 - Indent `public`, `private`, `protected` statements two spaces relative to their class' braces.
 - For other code blocks, indent four spaces relative to their enclosing braces.
 - Comments on their own line should be indented to the same level as the code they are commenting.
2. Be consistent in spacing, bracket placement, formatting, etc. If modifying someone else's code, respect and attempt to follow their style (otherwise, you are introducing inconsistency).
3. Separate words and acronyms within a name with an underscore (if you **must** use CamelCase, be consistent throughout the class/scope).
4. Prefer complete words or acronyms in names. For example, use `is_anal_retentive` instead of `is_anl_ret`.
5. Distinguish variable and function names from user-defined type names as follows: Begin all words in type names with a capital letter. Begin all words in variable and function names with a lowercase letter.
6. One should not have to search outside of a file, or parse an entire function definition, to determine the origin of a name used within that file (ideally, apply this down to function definitions). This rule implies the following:
 - Use `d_` for members of a class. Use `b_` for members of a base class. Define the private `typename Base` in a derived class as a reference for the `Base` class, ie:

```

class A
{
  public:
    A(int a, int b);

  private:
    // Base class members that do something important (notice that
    // these members are commented?).
    int b_a, b_b;
};

class B : public A
{
  typedef A Base;
  public:

    explicit B(int x);

  private:
    // This is an important datum.
    int d_important_datum;
};

```

This allows constructors and virtual functions to clearly and easily reference the parent class:

```
B::B(int x)
    : Base(x + 1, x + 2)
    , d_important_datum(x)
{
    /* */
}
```

- Even within an implementation file, avoid using namespace. One exception here is using namespace std, as long as you are using very common stuff from std(cout, endl, vector,...). It is still preferable that you explicitly specify which names you're using (e.g., using std::cout).
7. Limit the body of a function definition to approximately one page in length.
 8. Declare only one variable per line, including within function definitions:

```
void blorp(int i,
           int j)
{
    // preparing to blorp
    // ...
}
```

9. In general, do only one operation per line. For example, avoid:

```
a = b = c = d = 0;
```

however, this is acceptable:

```
int a = 0, b = 0, c = 0, d = 0;
```

8.1.6 Code comments and documentation

Code must be reasonably commented and documented using **Doxxygen**. The Exnihilo templates provide code-blocks for **Doxxygen** comments. Also, the Exnihilo development environment provides **emacs** and **vim** define macros and menus for documenting the code as described in *Development Environment*. Our preference is to put one-line comments for each member function in the header file and its documentation in the implementation file.

8.2 Use of language features

This section gives rules on language features to use and to avoid. Items specifically related to C++11 can be found in *C++11 Usage*.

8.2.1 Use namespaces

All code should be contained within a **namespace**. The name of the **namespace** is generally (with exceptions) taken from the package containing the code; although, it is acceptable to use subpackage namespaces when these make logical design sense.

8.2.2 Enforce const-correctness

When declaring a variable or function, use the `const` qualifier whenever possible. If unsure when declaring the variable, then go ahead and add the `const` qualifier. You can always remove the qualifier later. We discourage strongly the practice of adding const-correctness after major features have been implemented.

Denovo may have to use external libraries that do not enforce const-correctness. Such libraries must be wrapped in the equivalent functionality that enforces const-correctness. Otherwise, if wrapping is not done, the external library potentially could force all of Denovo to abandon const-correctness. Wrapping has other benefits, such as that the external library can be swapped out.

8.2.3 Design-by-Contract (TM)

The Design-by-Contract (DBC) macros defined in are defined in `harness/DBC.hh`. The following examples shows how DBC can be used:

```
#include "harness/DBC.hh" // defines Require, Check, Ensure, ...

double pressure(double temperature,
                double density)
{
    Require (temperature >= 0.0); // use Require() to check input values
    Require (density >= 0.0);

    double p; // pressure that is returned

    // ... code that computes initial guess for p ...

    Check (p >= 0.0); // use Check() for intermediate calculations

    // ... code that computes final value of p ...

    Ensure (p >= 0.0); // use Ensure() for final values

    return p;
}
```

8.2.4 Data Hiding

Aside from pure data structures (i.e., a `struct`), class member data should be accessed only through member functions. There are several suggestions for accessors. Consider the following example with accessors to large data structures; presumably, the `CCF` class contains a large amount of data. Care must be taken that `Solution` is not destroyed while handles it has returned are still available.:

```
class Solution
{
    // >>> DATA

    typedef std::vector<double> CCF; // cell-centered field
    CCF d_pressure;
    CCF d_density;

    public:

    // >>> ACCESSORS

    CCF &pressure() { return d_pressure; }
    const CCF &pressure() const { return d_pressure; }

    CCF &density() { return d_density; }
    const CCF &density() const { return d_density; }
```

```
// >>> ITERATORS

CCF::iterator begin_pressure() { return d_pressure.begin(); }
CCF::iterator end_pressure() { return d_pressure.end(); }

// ... etc ...
};
```

A memory-safe solution is the use the RCF (Reference Counted Field) class; however, this is may not be optimal for all use cases.:

```
#include "utils/RCF.hh"
class Solution
{
    // >>> DATA

    typedef denovo::RCF< std::vector<double> > CCF; // cell-centered field
    CCF d_pressure;
    CCF d_density;

public:

    // >>> ACCESSORS

    CCF pressure() { return d_pressure; }

    CCF density() { return d_density; }

    // >>> ITERATORS

    CCF::iterator begin_pressure() { return d_pressure.begin(); }
    CCF::iterator end_pressure() { return d_pressure.end(); }

    // ... etc ...
};
```

The data pointed to by any RCF will not go out of scope until the last one is destroyed. Also, copies are cheap because only the underlying reference is copied, not the data itself. Of course, this allows any owners of these fields to modify the underlying fields.

With regards to data hiding, optimization, ease of use, and safety must dictate design. These constraints are often conflicting. The following guidelines are helpful to keep in mind:

- If the class is a data container (for example, a container of cell-centered fluxes on a mesh), then data container semantics may be used to access the underlying data. These semantics include `operator[]`, `operator()`, and iterator access.
- If the member data is a large data structure, then for efficiency, handles to that data may be returned. The handles may be in the form of iterators or references, as in the first example above.
- Otherwise, `get/set` semantics should be used. The `@code{get}` function should not return a non-const handle to the data.

8.2.5 The `std` namespace

For example, use `#include <cmath>` instead of `#include <math.h>`. In general, do not use the `.h` system header files, which pollutes the global namespace. The resulting code will have the following constructions:

```
#include <cmath>
#include <iostream>

// ... stuff
x = std::sqrt(y);
std::cout << x << std::endl;
```

8.2.6 using statements

Do not place using statements where they might pollute the global namespace. Unless within the scope of an inline function, using statements should not be placed within header files (*.hh). Even within implementation files, preferably using statements should appear only within function scope.

Consider a A.hh file that contains the following:

```
#include <iostream>

namespace using_abuse
{
    using namespace std; // Not here!!!

    class A
    {
        public:
            void print_something() { cout << "I am lazy.\n"; }
    };
}

// end of namespace using_abuse
```

Now, consider a translation unit that uses A.hh:

```
#include "A.hh"

int main()
{
    // The following using statement is OK, because it's in an implementation
    // file. Unfortunately, it asks for using_abuse, but got std too!!!
    using namespace using_abuse;

    A a; // using_abuse::A is OK too
    a.print_something();

    cout << "Where did cout come from???\n"; // Answer: via using_abuse.
}
```

You might argue that namespace using_abuse is “yours,” and you’re free to pollute it all you like. However, someone else might have to maintain your code in the future. Also, placing using statements with header files may affect those who want to use your class, as illustrated above.

8.3 Things to avoid

The following is a list of things to avoid. It is not comprehensive, and like everything, there are times when “rules” need to be broken.

8.3.1 Circular dependencies

Circular dependencies arise from two-way associations. Denovo subscribes to the principles of *levelized design* (sometimes referred to as *acyclic* design). This is critical to the unit-testing and continuous-integration lifecycle models in Exnihilo. Thus, circular dependencies are **not** allowed. Additionally, there is never a reason to use circular dependencies (an *association* class can be used to decouple two-way associations). A simple example of a circular dependency is shown below:

```
class B; // forward declaration

class A
{
    int do_b(const B &b); // refers to class B
};

class B
{
    int do_a(const A &a); // refers to class A
};
```

Because both classes A and B refer to one another, they cannot be tested independently. Circular dependencies also create build-system nightmares.

8.3.2 Friendship

There are specialized cases where `friend` is useful (such as an iterator class for a container class), but generally, the use of `friend` is strongly discouraged. The use of `friend` violates data hiding, which was covered in the previous section. Remember, “in C++ friendship, much as in life, is often more trouble than its worth.”

8.3.3 Macro functions

Macro functions are not type-safe and should generally be avoided. This is not to imply that macros should **never** be used. We use them for our DBC implementation and conditional compiling. Simply be judicious when using macros.

8.3.4 Raw pointers

The use of raw pointers (for example, `double *x`) can be a major source of bugs. Often, the use of raw pointers can be avoided by substituting one of following techniques:

- Use a *shared pointer* instead (either `std::shared_ptr` or `std::unique_ptr`).
- Use a reference.

There are situations where using a raw pointer cannot be avoided. For example, raw pointers cannot be avoided when communicating with other languages, such as Fortran or C. In other cases, their use should be encapsulated. For example, container classes often use a pointer for their underlying data storage and may define its `iterator` type as pointer via a `typedef`. However, the pointer implementation in this case is encapsulated from the user of the container class. It is very important to avoid the following constructs:

```
class Foo_Factory
{
    private:
        Foo *d_foo;

    public:
```

```

void build()
{
    // build foo
    d_foo = new Foo();
    // ...
}

Foo* get_Foo() const { return d_foo; }
};

```

The issue is, “who ultimately deletes the Foo here?”. By returning a `SP<Foo>` this question is null. The last remaining owner will delete the Foo.

Finally, pointers-to-functions are a relic of C and can be avoided through the use of virtual functions.

8.4 C++11 usage

C++11 has many useful features; however, in Exnihilo we recommend a subset of features to use. The best place to discover the accepted use-patterns of C++11 features is to peruse the examples in `Nemesis/cxx11`. One thing to be wary of is that many compilers outside of `gcc` do not have good support for many C++11 features. And, `gcc` before 4.7.1 has some limitations as well. The following is a basic list of C++11 best practices:

- Keep the `auto` keyword usage *local* (ie. at function-scope). Avoid using `auto` in argument lists.
- Use `nullptr` to initialize raw pointers, *when raw pointers are absolutely necessary*.
- Use `unordered_map` and other new standard library containers; however, beware that the `emplace` is not supported by several compilers (see `tstCXX11_Std.cc`).

Note on Smart Pointers We will be making the conversion from native Exnihilo smart pointers (`SP` class) to the new C++11 standard smart pointer, `shared_ptr`. This is a transition, and `shared_ptr` should be avoided except inside of local class/scope internals at this point.

- Respect the [Rule of 5](#) (formerly known as the *Rule of 3*).

TESTING

Exnihilo contains two primary types of unit tests based on the two coding languages used.

9.1 C++ unit tests

Exnihilo uses the [Google Test](#) framework for most of its unit tests. (Some tests in the repository use an obsolete test harness.)

9.1.1 Creating a unit test

The Nemesis environment (*Development Environment*) installs a handy script for generating a template for new tests inside a test directory:

```
$ template-gen -n shift tstBlah.cc  
Successfully created tstBlah.cc
```

Here, the `-n` option is followed by the namespace. This creates a new google test file. If you don't have the environment scripts installed, simply create a new `.cc` file with the following line among the includes:

```
#include "Nemesis/gtest/nemesis/gtest.hh"
```

This includes the Google Test header file, declares additional Nemesis macros, and injects the custom `main()` function needed to run all the tests.

9.1.2 Adding the test

To build and automatically run the test, you will need to modify the `CMakeLists.txt` file inside the test directory. This file must include the following setup line:

```
INCLUDE(NemesisTest)
```

To build and run the test, add the simple macro directive:

```
ADD_NEMESIS_TEST(tstBlah.cc NP 1 2)
```

The `NP 1 2` option indicates the number of processors to allow to run the test. The full list of options (including setting environmental variables, linking in libraries, adjusting the timeout, and suppressing the running of the test through CTest) can be viewed inside the `Exnihilo/packages/Nemesis/cmake/NemesisTest.cmake` file.

9.1.3 Testing functionality

If you've generated the test file from the Nemesis environment utility, you'll see a file header, the command to include the gtest harness, and an empty "test fixture", along with two example testing blocks. Each TEST_F or TEST command generates a distinct subtest that appears in the output. This is a good way of grouping related tests. Each test contains testing macros that will print nothing when they succeed but will provide detailed error descriptions when they fail.

```
TEST(Math, simple)
{
    int a = 2;
    EXPECT_EQ(4, 2 + 2);
    EXPECT_GT(5, 2 * 2);
    EXPECT_TRUE(is_integer(a));
    EXPECT_FALSE(is_float(a));
}
```

A complete list of the test macros is available in the Google Test [primer](#) and the [advanced guide](#).

9.1.4 Additional functionality

The test harness code defined in `nemesis_gtest.hh` includes additional functionality not written by Google. It has a custom harness that handles MPI initializing and finalizing (using the Nemesis comm package), and it uses the Nemesis release metadata to print the Exnihilo and Scale git repository versions.

Additional macros that we define are mostly for comparing floating point values or containers of values:

EXPECT_SOFT_EQ (expected, actual, rel_error)

Check for "soft equivalence" (relative error except for numbers near zero) between two values.

Parameters

- **expected** – expected value
- **actual** – actual value
- **rel_error** – numerical tolerance

EXPECT_SOFT_EQ (expected, actual)

Check for "soft equivalence" with a relative error of 1.e-12.

EXPECT_VEC_EQ (expected, actual)

Check for equality between two contiguous containers of simple types (int, unsigned int, float, double, char, const char*, std::string). Each container can either be a fixed array such as:

```
int reference[] = {1, 2, 3};
```

or a contiguous container (std::vector, Nemesis view field, etc).

If the two container sizes are unequal, the test prints their sizes and returns a failure message. If the sizes are equal, it will do an element-by-element comparison. Any values (up to the first 40) that do not compare equal are printed along with their index:

```
tstVecEq.cc:87: Failure
Values in: actual
Expected: expected
2 of 5 elements differ:
i      expected          actual
0            1              2
3            4              5
```

Parameters

- **expected** – expected value
- **actual** – actual value

EXPECT_VEC_SOFTEQ(expected, actual, rel_error)

Check for “soft equivalence” (relative error except for numbers near zero) between two vectors of floating point values.

Parameters

- **expected** – expected value
- **actual** – actual value
- **rel_error** – numerical tolerance

EXPECT_VEC_SOFTEQ(expected, actual)

Check for vector soft equivalence with a relative error of 1.e-12.

We also provide a function that’s useful for printing “regression” values of integers or floats:

PRINT_EXPECTED(data)

Print a formatted array and initializer list for the given data. For example:

```
std::vector<int> vec = {1, 2, 3, 4, 5};  
  
PRINT_EXPECTED(vec);
```

will print:

```
const int expected_vec[] = {1, 2, 3, 4, 5, };
```

The resulting values can then be copy-pasted to the unit test and used in the expression:

```
EXPECT_VEC_EQ(expected_vec, vec);
```

9.1.5 Command line options

The Google test harness provides a number of useful command line options that can be viewed by passing the `--help` argument to the command line:

Each individual test block can be executed separately by running the test with the `--gtest_filter=TestCase.testname` flag.

9.1.6 Other features worth looking into

- Disabling performance tests by prepending `DISABLED_` to a test name.
- Using the `ASSERT_` macro to exit a test function when all further tests are known to fail (or cause crashes) if a condition is not met.
- Adding `EXPECT_THROW({func}, nemesis::assertion)` to ensure that DBC checks work.

9.2 Python unit tests

Exnihilo installs the `exnihilotest` package and the `denovo_unittest` module, which both wrap and extend the Python `unittest` package. Combined, they give the ability to do more advanced testing, they integrate into TriBITS, and they can run correctly and automatically with MPI.

9.2.1 Creating a unit test

Create a new `.py` file, either with the Nemesis `nemesis-python` command or with the function from `srjutils`, `template-gen tstWhatever.py`.

```
$ template-gen test_mymodule.py
Successfully created test_mymodule.py
```

The file (with dividers omitted for clarity) should look like:

```
1 from __future__ import (division, absolute_import, print_function, )
2 import omnibus.testing as unittest
3
4 if __name__ == '__main__':
5     unittest.main()
```

Line 1 contains a standard directive to make Python 2.7 behave more like Python 3: dividing integers yields a floating point number (use the `//` operator for C-like integer division); module imports are by default absolute rather than relative; and the `print` command acts like a function.

The second line acts as a stand-in for the standard Python `unittest` module. If testing a standalone Python package like `swordproc`, replace `omnibus.testing` with `exnihilotest`; if testing SWIG-wrapped code that might run in parallel (e.g. in the `pykba` or `pyshift` packages), then replace `omnibus.testing` with `denovo_unittest`.

Lines 4 and 5 will execute all the unit tests in the file when the script is run.

9.2.2 Adding to a unit test

Unit tests inside the file are created by adding a new class that begins with `Test`, inherits from `unittest.TestCase`, and has at least one method that begins with `test_`:

```
class TestSomething(unittest.TestCase):
    def test_blaah(self):
        self.assertEqual(4, 2+2)
```

Inside the methods are assertions that check for equality between expected and actual results. Each test function is run independently, so if one assertion fails, separate tests may still be run. For an extensive look at the capabilities of the package, see the full `unittest` documentation.

9.2.3 Additional functionality

The `exnihilotest` package defines several useful macros not available in the standard `unittest` package.

`TestCase.assertDataEqual(self, expected, actual[, eps=1.e-14])`

Check all elements recursively in a container. This works on dictionaries, numpy arrays, lists of lists, named tuples, etc.

Parameters `eps` (`float`) – “Soft equivalence” tolerance parameter.

`TestCase.assertTextEqual(self, expected, actual[, eps=1.e-14])`

Compare blocks of text which may have numbers inside it.

Parameters `eps` (`float`) – “Soft equivalence” tolerance parameter.

`TestCase.assertSoftEquiv(self, expected, actual[, eps=1.e-14])`

Check soft equivalence on two floats.

Parameters `eps` (`float`) – “Soft equivalence” tolerance parameter.

`TestCase.assertXmlFilesEqual(self, expected_f, actual_f, eps=1.e-14)`

Compare the structure and contents of two XML files for equality.

Parameters

- `expected_f` (`str`) – Expected file object or file path.
- `actual_f` (`str`) – Actual file object or file path.
- `eps` (`float`) – “Soft equivalence” tolerance parameter.

The `denovo_unittest` package prints warnings emitted by `NEMESIS_WARN` at the end of every test that it runs. When loaded, it prints the current version of Exnihilo and SCALE.

DEVELOPMENT ENVIRONMENT

Exnihilo provides templates and functions for developing code in Exnihilo/environment.

10.1 Setting up the Exnihilo development environment

We provide **emacs** and **vim** environments for editing source code. Also, a set of code preparation templates, along with LaTeX document styles are provided. To install the environment simply run the following script in the Exnihilo/environment directory:

```
sh ./install.sh /data/env
```

Here, the /data/env path should be where you want the environment to be installed. The following output is produced:

```
>>> Making /data/env
>>> Installing bibtex in /data/env/bibtex
>>> Installing emacs in /data/env/emacs
>>> Installing latex in /data/env/latex
>>> Installing etc/templates in /data/env/etc/templates
>>> Installing tools in /data/env/tools
>>> Installing visit in /data/env/visit
>>> Installing python in /data/env/python
>>> Installing bin in /data/env/bin

=====
Congratulations, you're almost finished the nemesis environment install!
```

For complete operation you should set the following paths and environment variables (depending upon your shell) :

- Add /data/env/bin to PATH
- Add /data/env/latex to TEXINPUTS
 - e.g. echo \$TEXINPUTS
 .:./data/env/latex:
- Add /data/env/bibtex to BSTINPUTS
 - e.g. echo \$BSTINPUTS
 .:./data/env/bibtex:
- Copy /data/env/visit/* to ~/.visit

To use the nemesis GNU Emacs or VIM environments see
- environment/README.rst

The script copies several recommended VisIT color palettes; to use them, copy `install/visit/*` into `~/.visit`.

10.2 Command line tools

Several useful development tools are installed into `environment/tools`. A few of the more useful ones are described here. Many of these tools use a small Python framework that allows files to be recursively modified based on their extensions.

10.2.1 auto-annex

Intelligently add or git-annex untracked files.

```
usage: auto-annex [path [path ...]]
```

Each path (either a file or directory) will be examined. If it has not yet been added to the Git repository, a heuristic algorithm will decide whether to annex it or to add it as a regular git file:

- If a binary file ¹ has a size greater than 5 KiB, it is annexed.
- If an image file ² has a size greater than 25 KiB, it is annexed.
- If an output file ³ has a size greater than 100 KiB, it is annexed.
- If any other file has a size greater than 2 MB, it is annexed.
- Any file less than these size thresholds is added as a regular git file.

10.2.2 check-cols

Determine whether source files are violating the Nemesis 80-column guideline.

```
usage: check-cols [-h] [-r] [-x EXTENSIONS] path [path ...]

Check for satisfying the 80-column limit. If violating files were found, a
file "violators" will be written containing their paths.

positional arguments:
  path                  Files/dirs to process

optional arguments:
  -h, --help            show this help message and exit
  -r                   Recursive
  -x EXTENSIONS, --extensions EXTENSIONS
                      Comma-separated extentions to use when doing recursive
```

¹ Binary files are determined by checking the extension against h5 sh5 silo ampx dat bin bmg h5m sat cub dnv 37 bin docx pptx xlsm ppt doc xls session gui.

² Image files have en extension png pdf jpg psd svg.

³ Output file names either have extensions of mcnpo out log plt, have paths that end with out.xml or output.xml, or look like MCNP mctal or meshtal or outp files.

10.2.3 fix-comment-paths

Modify the comment blocks at the beginning and end of Nemesis-style files to ensure that the declared file path is consistent with the actual file path.

```
usage: fix-comment-paths [-h] [-r] [-x EXTENSIONS] path [path ...]

Update file paths in the comment headers and footers of Nemesis-style files.

positional arguments:
  path                  Files/dirs to process

optional arguments:
  -h, --help            show this help message and exit
  -r                   Recursive
  -x EXTENSIONS, --extensions EXTENSIONS
                      Comma-separated extentions to use when doing recursive
```

10.2.4 make-skeleton-scale

Create a “skeleton SCALE” source directory used to build Exnihilo without any of the SCALE source code. This is primarily used for distributions of ADVANTG, where the SCALE cross section processing code is not used.

```
usage: make-skeleton-scale SCALEDIR DEST.tgz
```

10.2.5 update-scale-data

Update the SCALE data directory to the latest version of SCALE data. Your \${DATA} environment variable must be set for this to work. The first time you run the tool, you must run `mkdir ${DATA}` to create the initial empty directory on your machine.

```
usage: update-scale-data

Update the SCALE data directory (using the DATA environment variable)
from the /projects/scale/scale_dev_data directory on remus.ornl.gov.
```

Note: In addition to copying the SCALE data itself, this tool saves a copy of the Subversion repository data (version number and date of change), which is processed by Exnihilo and saved into Omnibus output files for later verification.

10.3 Text editor environments

As previously mentioned, both **emacs** and **vim** editing environments are provided that correspond to the Exnihilo indentation style, along with many other useful editing features.

10.3.1 EMACS Development Environment

To use the nemesis GNU **emacs** environment add the following to your `.emacs` or `.emacs.d/init.el` file:

```
(setq nemesis-dir "/data/env/emacs")
(add-to-list 'load-path nemesis-dir)
(load-library "nemesis")
```

Other useful **emacs** packages/modes include

- **doxymacs**: for **doxygen** markup in source code
- **cmake**: for **cmake** editing
- **auctex**: for LaTeX editing
- **ecb**: the **emacs** code-browser extensions
- **auto-complete**: auto-completion of symbols (code)

10.3.2 VIM Development Environment

These are **vim** format commands for compliance with the Exnihilo coding guide. Simply add the `ftplugin` files to your `~/.vim/ftplugin`, and append `vimrc` to your existing `~/.vimrc`.

USEFUL TOOLS

The Exnihilo developers tend to use a common group of tools, including git for versioning, Emacs and MacVim for file editing, etc. This document provides more details on both the standard tool set as well as additional tools in use.

11.1 Git

Git at first is a daunting tool, but a little experience and guidance makes it a powerful ally in the quest for a maintainable code system.

11.1.1 Git configuration

Git supports a configuration file at `~/.gitconfig`. A recommended default is:

```
[core]
    excludesfile = ~/.gitignore
    pager = less -r
    autocrlf = input
    whitespace = trailing-space,space-before-tab
[apply]
    whitespace = fix
[color]
    ui = auto
[alias]
    autoannex = !auto-annex
    st = status -s
    co = checkout
    dc = diff --cached --ignore-space-change
    mnc = merge --no-commit --no-ff -s recursive -Xignore-space-change
    comall = commit --all
[branch]
    autosetuprebase = always
[rebase]
    autosquash = true
[push]
    default = upstream
[rerere]
    enabled = true
[merge]
    autosetuprebase = always
[user]
    email = johnsonsr@ornl.gov
    name = Seth R Johnson
```

Don't forget to change the email and name!

The init file points to a global `.gitignore` file that hides temporary files from Git, preventing them from accidentally being included in a commit. A recommended default is:

```
*.aux  
*.pyc  
*.out  
*.log  
*.bb1  
*.blg  
*.synctex.gz  
*.nav  
*.snm  
*.toc  
*.swp  
*.swo  
.DS_Store  
visitlog.py  
*~  
runtpl*  
srctp*  
.ipynb_checkpoints  
.cecache*
```

11.1.2 Merging and conflict resolution

To view all the changes your branch has made since splitting off from master, use the command:

```
$ git diff $(git merge-base origin/master HEAD)
```

To merge the topic branch into the master, do the following:

```
$ git checkout master  
$ git reset --hard origin/master  
$ git merge --no-commit my-branch
```

Resolve any conflicts and run all (downstream) tests to ensure that the merged branch does not cause any failures. Conflicts may be either *explicit*, where the master and the topic have both modified a line of code, or they may be *implicit*, where a new file on the master relies on old behavior that the topic branch changes.

Caution: Implicit conflicts are subtle and cannot be caught by Git's merge resolution mechanisms. Consider the following case:

```
...N----o (other developer)  
     \  
...o----o----o----o---M **master**  
     \| /  
      o----R----o * (your topic) *
```

Commit R in your topic branch renames a class `MyClass` to `MyRenamedClass`, and some other developer on their branch creates a new class at commit N that references `MyClass`. The other developer merges into the master branch and everything works. You finish up your topic branch, and everything works. Because your topic branch and the other developer's branch didn't touch the same files, merging into master at M compile individually, and no merge conflicts will be generated. However, because the new class the other developer created references `MyClass` instead of `MyRenamedClass`, the merge will fail to compile! This is why it's important to always build and test before pushing.

Upon successfully completing the merge, you can push to the origin:

```
$ git push origin
```

Using this basic workflow will create a clean repository in which commits are descriptively labeled, and it will be easy to walk backwards through the history.

11.1.3 Advanced merging

When resolving a long-running branch, where significant restructuring to code has been performed, it's often easier to complete the conflict resolution in multiple stages.

Tip: When merging a topic branch that contains a significant code refactor, it takes time to resolve conflicts. The more time it takes, the more likely it is that another developer pushes changes onto the master branch, and the more frustrating it will be to redo the merge.

To mitigate this frustration, our git server supports the ability to “lock” the repository while the merge is taking place; talk to one of the Exnihilo developers for details on how and when to do this, and it will reduce the likelihood of your having to use the advanced techniques laid out in this section.

The first step is to make sure that the “rerere” option given in [Git configuration](#) is set; this allows Git to remember how you resolved your merge in case you have to perform the merge again. Then, run `git merge --no-commit my-branch`, resolving explicit merge conflicts as needed. Run `git diff` to make sure that no conflict resolution markers are left in the code, then run `git commit --all`.

The next major step is to resolve any *implicit* conflicts. Start building the merged code. If it builds and the tests pass, you can push. If there are build errors or test failures from the merge, fix them, mark the changes to be added with `git add -A :/`, and make a commit with `git commit -m "Merge fixes"`. This creates a checkpoint of all the merge conflicts that Git won't be able to automatically resolve.

If the master has *not* been updated since you started your merge (run `git fetch` to check), you can run the following code to collapse your rolls both the explicit and implicit conflicts into the single merge commit:

```
$ git reset --soft HEAD^
$ git commit --amend
```

Then `git push` and you're done.

However, if someone *has* pushed to master while you were doing your merge, you have a problem:

```
...o--o *(unsuspecting developer's topic)*
 \
...o---o---o--o **origin/master**
 \
      M---N **master** (your original merge in progress)
 /
 ...-o--o *(your topic)*
```

In this case your original merge resolutions are in M, the additional merge fixes are in N, and your `master` has diverged from `origin/master`. If you try to push, you'll get a message saying your push was rejected.

Warning: If this happens, *do not* merge `origin/master` into your merge commit, and also *do not* merge M' into `origin/master`!

But fear not! There is an easy solution. You can recreate your merge on top of the updated `origin/master` because you separated “implicit” merge conflicts into a separate commit and because you have the “rerere” option enabled. First, copy the git hash so that you can later refer to your original merge attempt:

```
$ git log -1 --oneline  
abc123 Merge fixes
```

To rebase your merge, use the `--preserve-merges` option:

```
$ git fetch origin  
$ git rebase -p origin/master
```

which rebases your merge commit. You'll get a message about "recorded resolutions" being replayed; you should have to commit to acknowledge that they were applied correctly, and then the rebase should complete. You should end up with:

```
...o---o * (unsuspecting developer's topic)*  
 \  
...o---o---o---o---M'---N' **new master**  
 /  
...o---o * (your topic)*
```

Then you can "roll" your `N'` into `M'`. (Double-check that no new conflicts have been created, of course.) To do a final check that you didn't lose anything in this rebase, you can refer to the earlier git hash you saved and run a git diff:

```
$ git diff --stat abc123
```

The only differences should be those the unsuspecting developer introduced in their branch.

Tip: If you didn't save your git hash earlier, you can recover it by looking through the results of `git reflog` and finding the commit named "Merge fixes".

Once everything is checked, you are of course free to push.

11.2 Git annex

Git-annex is an add-on to Git designed for safely syncing large files between repositories. Many codes output binary files that measure in GB or more. Git is slow to handle such files, and every user who pulls a copy of a research repository must also pull all those large files. Because of Git's nature, it's also nearly impossible to delete one of those files when it's superseded by a newly regenerated version.

Git-annex solves these problems by moving large files into a hidden directory inside your git repository and using `rsync` under the hood to copy the files between computers. It maintains an awareness of where your files are located (through a hidden git branch) and will prevent them from being accidentally deleted.

For example, when git-annex is configured locally and on the server, to add some large research result alongside some smaller text documents, you might do the following:

```
# First, add large files to the annex. This moves them to  
# .git/annex/objects and creates symlinks in your directory.  
git annex add *.h5  
git add .  
git commit -m "Added some files"
```

Now these large files are stored securely on your computer. Calling `git rm` will not delete the original files; you'd have to run `git annex drop` to remove them.

11.2.1 Installation

On the Mac, the easiest way to install is to [download the OS X binary](#). After you copy the app to /Applications, you can symbolically link the following important files to /usr/local/bin or some other convenient location:

```
cd /Applications/git-annex.app/Contents/MacOS/bundle/
ln -s git-annex git-annex-shell gsha256sum /usr/local/bin
```

You should of course ensure that /usr/local/bin is included in your \$PATH environment variable.

11.2.2 Creating a research repository

On the server system, you'll want to initialize a bare (i.e., only git files and no working copies of the files) repository accessible and writable by your group.

```
$ git init --bare --shared srj_annex
Initialized empty shared Git repository in /repos/git/srj_annex/
```

If you have git-annex set up correctly with your paths, you can then initialize git-annex with a unique name for this repository:

```
$ git annex init origin
init origin ok
(Recording state in git...)
```

Now, clone your repository on your personal computer:

```
$ git clone ssh://server/repos/git/srj_annex
Cloning into 'srj_annex'...
```

That's it: you've now got git-annex-compatible repositories on the server and your computer.

You can also `git annex init` on existing repositories, and add files to those, but because they might already have obnoxiously large files in them it may be better to simply create a new repository. If you're a power user and want to preserve your history but annex old files, you may be able to adapt use [this advanced git-filter technique](#).

11.2.3 Adding files

In the Nemesis environment, we provide a tool `auto-annex` that assists in determining whether files should be added to the repository or added to the git-annex store. Instead of calling:

```
$ git add .
```

after creating new files, you should call:

```
$ auto-annex .
```

(assuming the Nemesis environment tools are in your path.)

11.2.4 Adding files manually

Instead of using the `auto-annex` tool, files can be added manually.

When you have a commit to make, it's better to add large files with git-annex before adding them with git. This is because git has to do extensive processing of the file's entire contents, whereas git-annex has a much shorter task to perform.

```
# (move files in or run your code)
$ git annex add *.h5
(Recording state in git...)
$ git add .
$ git commit -m "Initial commit"
[master (root-commit) ac4ed4c] Initial commit
```

Notice that after the first `git annex` command, your HDF5 files have become symlinks that point to a hidden directory stored only on your computer (at the moment). Pushing to the origin only pushes the symbolic link, not the actual data. (Thus, for very large data files that you probably won't ever need to share, you can limit their duplication.) To actually push the annexed files, you should not only push your master branch...:

```
$ git push -u origin HEAD
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 364 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To ssh://server/repos/git/srj_annex
 * [new branch] HEAD -> master
```

...but also copy the annexed files:

```
$ git annex copy --to origin .
(merging origin/git-annex into git-annex...)
(Recording state in git...)
copy celltally.h5 (checking origin...) (to origin...)
....
ok
(Recording state in git...)
```

Now `git-annex` will tell you that you've safely stored copies of the large file both here and on your remote server:

```
$ git annex whereis celltally.h5
whereis celltally.h5 (2 copies)
  68214887-..... -- [origin]
  9e9236ec-..... -- s3j@local~/srj_annex [here]
```

If more than one person is syncing with your repository, it might be necessary to run `git annex sync` to see all the remote locations where your file is stored.

11.2.5 Managing files

At this point, since `git-annex` knows that your file is safely stored elsewhere, you can tell it to “drop” the file, or remove the local copy:

```
$ git annex drop celltally.h5
drop celltally.h5 (checking origin...) ok
(Recording state in git...)
```

And now of course, you can no longer access the file on your computer because it's been deleted:

```
$ h5dump celltally.h5
h5dump error: unable to open file "celltally.h5"
```

To get it back by pulling it from the server, you use `git annex get`:

```
$ git annex get celltally.h5
get celltally.h5 (from origin...)
SHA256E-.....h5
    714736 100% 40.72kB/s 0:00:17 (xfer#1, to-check=0/1)
...
ok
(Recording state in git...)
```

By the way, all these transfers are rsync over ssh, so it's efficient, secure, and fast.

If you need to modify an annexed file, which is read-only by design, you'll have to "unlock" it:

```
git annex edit celltally.h5
unlock celltally.h5 (copying...) ok
```

Then you can overwrite it, and the next time you commit, git annex will re-annex the modified file (under a new unique file name):

```
$ git commit -m "Modified cell tally file"
add celltally.h5 ok
ok
(Recording state in git...)
[master 2fb8463] Modified cell tally file
```

Thus both the original file and the modified copy are preserved.

Finally, if you have multiple computers referencing your git annex-ed files, you can run `git annex sync` to ensure that your computer's knowledge of what files are safely annexed is in sync with the server's knowledge.

11.2.6 Cautions

- I've had weird behavior when rebasing with an annexed repository. I'd recommend not rebasing, and just merging your remote and local branches.
- Because annexed files are stored in `.git/objects` rather than the git repository itself, it is *imperative* that you call `git annex copy` in addition to `git push` if you plan on deleting and re-pulling the working copy of your repository.

11.3 iPython notebook

iPython notebook is a Python-base interactive analysis environment that we frequently use. We have example files and analysis notebooks in various directories, including `examples/`, `packages/Physica/sce/nb`, and `packages/Omnibus/frontend/nb`.

iPython integrates into Exnihilo via both the Python wrappers (which expose compiled C++ code) and the Omnibus postprocessing suite. The Python wrappers enable users to explore SCALE multigroup and CE data, to build multi-group cross sections, and even to drive SCALE sequences to generate multigroup data.

11.3.1 General conventions

We typically begin each notebook with some boilerplate code that enables various useful packages and default settings:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from omnibus.parser.validator import (MTValidator, NuclideValidator,
                                       to_nuclide)
name_to_mt = MTValidator.to_mt

from srjutils.matplotlib import (article_style, screen_style, poster_style,
                                   grid)
%matplotlib inline
screen_style()

pd.set_option('display.float_format', '{:.3e}'.format)
pd.set_option('display.max_rows', 25)
```

To save a figure for inclusion in a technical article, you should modify the plot settings to use sans-serif fonts and so forth using the `article_style` function:

```
article_style()
plt.plot(x, y)
plt.savefig("everythingisawesome.pdf")
```

11.3.2 Configuration

To create and edit the default iPython notebook:

```
$ ipython profile create
```

If you have a Mac with a “Retina” display or some other high-DPI monitor, you can set iPython to default to producing 2x-resolution images for inline display:

```
$ open ~/.ipython/profile_default/ipython_notebook_config.py
```

and set

```
c.InlineBackend.figure_format = 'retina'
```

Part III

User Guide: Omnibus

Omnibus is the general-purpose front end to Exnihilo. It provides an easy and “traditional” ASCII interface to the deterministic and Monte Carlo solvers in Exnihilo.

FRONT END INTERFACE

Your Exnihilo installation contains not only an `omnibus` executable (which runs Denovo and Shift) but also additional Python scripts that preprocess user input and program output.

12.1 Running Omnibus

The `omnibus-run` script is able to create an internal Omnibus input file, drive and monitor the `omnibus` executable as it's being run, and postprocess the output.

Note: Unlike most code drivers, `omnibus-run` is meant to be executed **on the head node** of a cluster rather than on a compute node. Using a machinefile (if the `[RUN=mpi]` option is being used) or pbs submission (for `[RUN=pbs]`), it is able to submit the job to other nodes and monitor the application process.

Tip: For systems such as Titan that have special filesystems (Lustre) that from which the code is executed, the easiest way to ensure that all Omnibus I/O remains on that system is to leave the `.omn` input file on Lustre and call `omnibus-run` from that directory.

12.1.1 Example on a local machine

Suppose you have an input file `batman.omn` on your local machine:

```
[PROBLEM]
name Batman
description "I'm the Batman."

! -- snip -- !

[RUN=mpi]
np 4
```

If Omnibus is installed with MPI, all you need to do is to open a terminal and call:

```
$ omnibus-run batman.omn
```

This runs the following sequence:

1. The preprocessor will validate your input.
2. After successful validation, the preprocessor will write an `xml` intermediate file read in by the `omnibus` binary executable. It will also, if necessary, generate other input files (such as the run tape file used for running on MCNP geometry).

3. The script will launch a local process with arguments like `mpirun -np 4 omnibus batman.xml`, then save the output to the current directory and echo it to the screen.
4. When the `omnibus` binary is complete, it will write out tally data and other program output to several different files.
5. A Python postprocessor reads the output and converts it to a human-readable format, leaving the original output files for later postprocessing.

12.1.2 Example on a cluster using PBS/Torque

In this example, we copy the local machine problem and change the run block (see *[RUN=pbs]*):

```
[PROBLEM]
name Batman
description "I'm the Batman."
! -- snip -- !

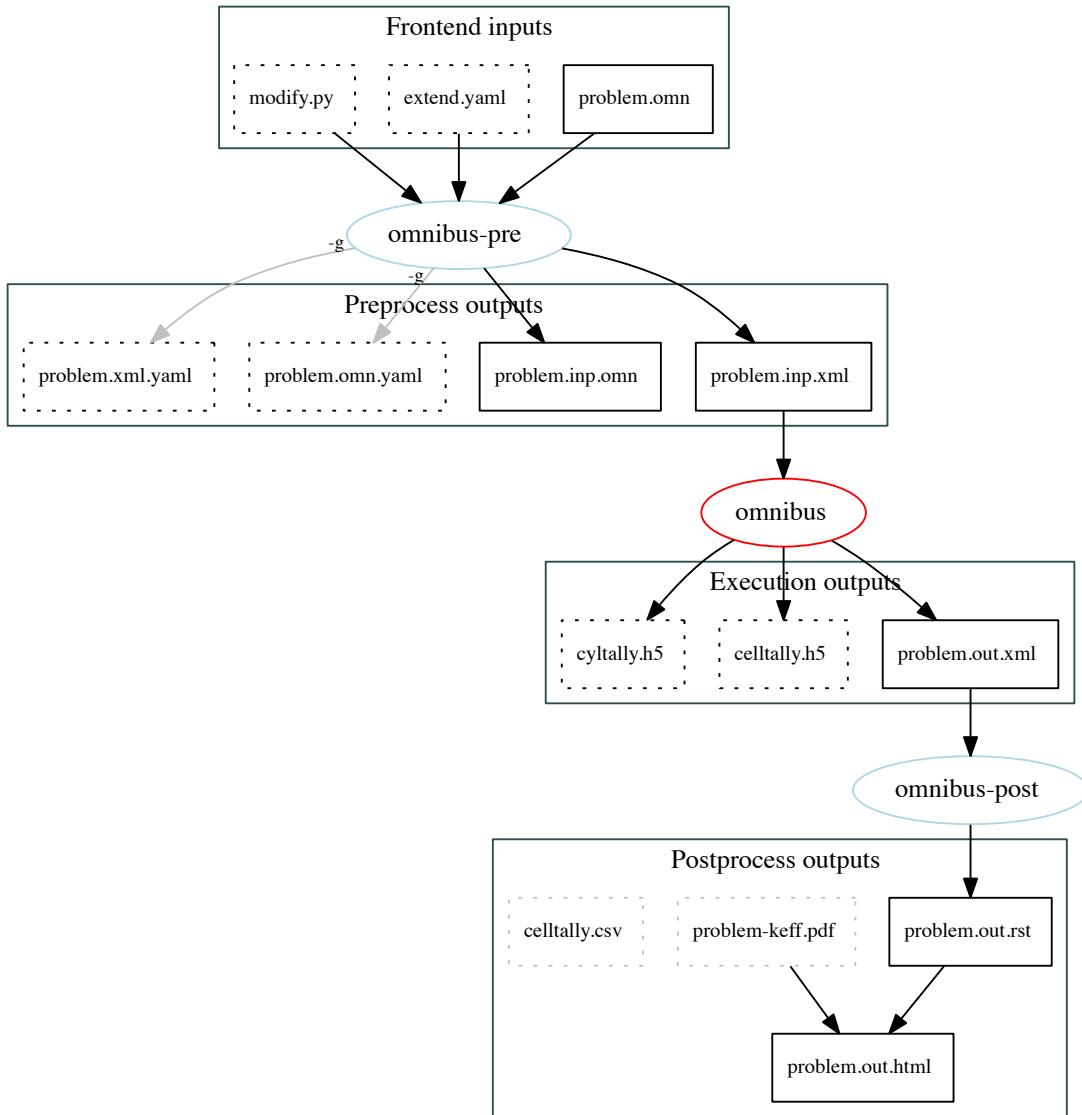
[RUN=pbs]
nodes      1
ppn        32
walltime  "1:00:00"
```

When `omnibus-run` is executed on the head node, it will launch `qsub`, monitor the job ID until it begins running on the compute node, and echo output to the screen over an ssh connection. The process on the head node remains almost entirely idle during the problem run, so the user need not worry about incurring the wrath of the system's administrator.

12.2 Omnibus input and output files

Omnibus accepts multiple input formats, writes (possibly multiple) intermediate files, and processes the output into more useful formats.

The following image describes how files are generated and used:



Here, the small black boxes are the typical input/output files, blue circles are scripts, the red circle is the Omnibus executable, and dotted lines are optional files (e.g. having multiple input files or using the `-g` option when calling `omnibus-run` or `omnibus-pre`).

12.2.1 Input files

Omnibus ASCII input files (with the ‘.omn’ extension) are described in [ASCII input](#):

```
[PROBLEM]
name "CE pin cell lava_scempp kcode problem"
mode kcode
xml_output kcode.xml
```

```
[GEOMETRY=mcnp]
mcnp mcnp_godiva.mcnp
```

We also support YAML and JSON heirarchical databases, which may appeal more to power users:

```
{
  "problem": {
    "name": "CE pin cell lava_scempp kcode problem",
    "mode": "kcode",
    "xml_output": "kcode.xml"
  },
  "geometry": {
    "_type": "mcnp",
    "input": "mcnp_godiva.mcnp",
  }
}
```

Additionally supported are Python files that can modify an existing (e.g. ASCII-created) database. This last method is extremely powerful for automating repetitious tallies, as demonstrated in this example that creates five similar cylindrical mesh tallies that share an energy grid:

```
import numpy as np

new_tallies = []

neutron_bins = [2e7, 1e5, 1e3, 10, 1, 1e-5]
photon_bins = np.linspace(0, 1e6, 11)[::-1] # 10 linear bins to 1 MeV
reactions   = ["flux"]

targets = [
    # area,      loc,      x,      y,      r
    ('PTP', 'FT-A1', -4.66117, -2.69113, 0.929640),
    ('SVXF', 'VXF-1', 3.07648, 39.09038, 2.011680),
    ('SVXF', 'VXF-2', -3.45642, 43.91796, 2.011680),
    ('SVXF', 'VXF-3', -9.15368, 38.12784, 2.011680),
    ('PTP', 'FT-A1', -4.66117, -2.69113, 0.929640),
]

# Add each tally to the list
for (area, loc, x, y, r) in targets:
    tal = {
        'name': "%s:%s" % (area, loc),
        'description': "flux in %s target location %s" % (area, loc),
        'reactions': reactions,
        'r': [0.0, r],
        'theta': [0.0, 1.0], # divided by 2pi
        'translate': [x, y, 0],
        'z': [-25.4, 25.4],
        'neutron_bins': neutron_bins,
        'photon_bins': photon_bins,
    }
    new_tallies.append(tal)

# Set all cylindrical tallies
assert 'tally' in db
assert 'cylmesh' not in db['tally']
db['tally']['cylmesh'] = new_tallies
```

12.2.2 Preprocessing output files

The preprocessing step will typically create several files for an input `problem.omn`:

`problem.json` If using the `omnibus-run` front end, this will be created: it is a fully processed and reformatted version of the problem input.

`problem.inp.omn` If using the `omnibus-run` front end, this will be created: it is a fully processed and reformatted version of the problem input.

`problem.inp.xml` The Teuchos ParameterList XML file is read by the `omnibus` executable.

`problem.inp.xml.json` This file is simply a more-readable version of the XML file, produced when the `-g` option is enabled for debugging.

Additionally, if MCNP geometry, physics, or sources are being used, *run tape* files will be generated. Finally, if a [RUN=pbs] block is present, a `problem.pbs` submission script will be generated.

12.2.3 Executable output files

All output file names are generated from user input (although the XML output name defaults to `problem.out.xml`). The post-processed `problem.html` file will contain a list of the created files and their descriptions.

12.3 Running Omnibus manually

The relationship between the various input files and executables is necessarily complicated, which is why the `omnibus-run` command is preferred. To generate the Omnibus XML file from an ASCII input, call:

```
$ omnibus-pre my_problem.omn
```

This will create a Teuchos ParameterList XML input file `unikitty.xml`. This parameter list is then run with the Omnibus driver:

```
mpirun -np 16 omnibus my_problem.inp.xml
```

Postprocessing (including plotting keff and Shannon entropy convergence, as well as rendering the XML output into a more human-readable format) is done with the command:

```
omnibus-post my_problem.out.xml
```

12.4 Command line tools

12.4.1 omnibus-run

Run the Omnibus preprocessor, run Omnibus, and run the postprocessor.

Run Omnibus from start to finish.

```
usage: omnibus-run [-h] [--version] [-g] [-c] [-v] [-q] [--very-quiet]
                   [--silent]
                   [--log {None,DEBUG,STATUS,INFO,WARNING,ERROR,CRITICAL}]
                   [--doc]
                   [inp [inp ...]]
```

Positional arguments:

inp Input file names (omnibus, yaml, and/or python).

Options:

--version show program's version number and exit
-g=False, --debug=False Enable extended debug assertions
-c=False, --clobber=False Overwrite exiting output files rather than renaming them.
-v=STATUS, --verbose=STATUS Print all debug messages
-q, --quiet Only print informational and warning messages
--very-quiet Only print warning messages
--silent Print messages only on failure
--log Create a log file with the given verbosity
Possible choices: None, DEBUG, STATUS, INFO, WARNING, ERROR, CRITICAL
--doc=False Print documentation and exit.

Exnihilo version (UNKNOWN)

12.4.2 omnibus-pre

Generate an XML input file for Omnibus, validating input along the way.

Preprocess Omnibus input files.

```
usage: omnibus-pre [-h] [--version] [-g] [-c] [-v] [-q] [--very-quiet]
                   [--silent]
                   [--log {None,DEBUG,STATUS,INFO,WARNING,ERROR,CRITICAL}]
                   [-o OUTPUT] [--doc]
                   [inp [inp ...]]
```

Positional arguments:

inp Input file names (omnibus, json, yaml, and/or python).

Options:

--version show program's version number and exit
-g=False, --debug=False Enable extended debug assertions
-c=False, --clobber=False Overwrite exiting output files rather than renaming them.
-v=STATUS, --verbose=STATUS Print all debug messages
-q, --quiet Only print informational and warning messages
--very-quiet Only print warning messages
--silent Print messages only on failure
--log Create a log file with the given verbosity
Possible choices: None, DEBUG, STATUS, INFO, WARNING, ERROR, CRITICAL
-o, --output Output filename (xml, json, omn, yaml)

--doc=False Print documentation and exit.
 Exnihilo version (UNKNOWN)

12.4.3 omnibus

The actual Omnibus binary executable.

```
usage: omnibus [--version] xml_input
```

Positional arguments:

xml_input	Path to the XML parameter input file.
-----------	---------------------------------------

Options:

--version	Show usage information and exit.
-----------	----------------------------------

12.4.4 omnibus-post

Postprocess tally output.

Post-process Omnibus output.

```
usage: omnibus-post [-h] [--version] [-g] [-c] [-v] [-q] [--very-quiet]
                     [--silent]
                     [--log {None,DEBUG,STATUS,INFO,WARNING,ERROR,CRITICAL}]
                     [--fast]
                     outp
```

Positional arguments:

outp	Omnibus XML output file names
-------------	-------------------------------

Options:

--version	show program's version number and exit
-g=False, --debug=False	Enable extended debug assertions
-c=False, --clobber=False	Overwrite exiting output files rather than renaming them.
-v=STATUS, --verbose=STATUS	Print all debug messages
-q, --quiet	Only print informational and warning messages
--very-quiet	Only print warning messages
--silent	Print messages only on failure
--log	Create a log file with the given verbosity
	Possible choices: None, DEBUG, STATUS, INFO, WARNING, ERROR, CRITICAL
--fast=False	Undocumented

Exnihilo version (UNKNOWN)

12.5 Advanced execution through Python

Because the Omnibus front end is simply a Python script, it's possible to drive it through another Python script. For example, the shell command

```
$ omnibus-run inyoka.omn
```

can be written as the following python script:

```
from omnibus.scripts import omnibus_run

omnibus_run.run(["inyoka.omn"])
```

The first argument to `run` takes a list, because multiple files can be passed to the input just like with the scripted front end. An additional and very nifty feature is that a python *function* can also be passed to modify the database, just like a Python script can be passed through the command line:

```
from omnibus.scripts import omnibus_run

def change_name(db):
    db['problem']['name'] = "Something else"

omnibus_run.run(["inyoka.omn", change_name])
```

This opens up exciting possibilities of automating scaling studies and parameter studies. For example, this script performs a weak scaling run:

```
from collections import defaultdict
import json
import os

from omnibus.scripts import omnibus_run
from omnibus.utils import working_dir

# Number of histories
histories_per_core = 1e4

data = defaultdict(list)

# We run in a subdirectory, so save the absolute path to our input
omn_input = os.path.abspath("omn_input.omn")

for num_procs in range(1, 8+1):
    def np.setter(db):
        # Adjust number of cores
        db['run'] = {
            '_type': "mpi",
            'np': num_procs,
        }

        # Adjust number of histories
        db['shift']['np'] = num_procs * histories_per_core

    # Execute in a subdirectory and extract the postprocessed out.xml file
    with working_dir("np-%01d" % (num_procs)):
        manager = omnibus_run.run_or_pp([omn_input, np.setter])

    data['cores'].append(manager.num_procs)
```

```

    data['histories'].append(manager.num_histories)
    data['solve_time'].append(manager.get_timing("shift",
                                                "shift::Fixed_Source_Solver.solve"))

# Save timing results
with open('results.json', 'w') as f:
    json.dump(dict(data), f)

```

The special `run_or_pp` command will reuse any existing `.out.xml` file if available, so this script can be resumed if canceled partway. The result is a lovely file of solution times:

```
{
    "cores": [1, 2, 3, 4, 5, 6, 7, 8],
    "histories": [10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000],
    "solve_time": [66.6750500202179, 71.05331420898438, 73.68022584915161,
                   74.62563920021057, 75.13657093048096, 75.84799098968506,
                   76.0614001750946, 76.82996106147766]
}
```

which can then be plotted or analyzed:

```

import json
import pandas as pd
with open("results.json") as f:
    data = json.load(f)

results = pd.DataFrame.from_dict(data)
results.index = results.pop("cores")

results['solve_time'].plot()

```

12.6 Parameter list explanation

The omnibus executable runs using a heirarchical XML parameter list. The necessary complexity of the parameter list makes it nearly prohibitive for a user to write it from scratch. However, some knowledge of the underlying XML parameter list structure is useful in understanding the design choices made for the Omnibus frontend.

The parameter list tree built by Omnibus consists of three general types of elements:

- Parameters: simple elements with a name and a particular range of allowed values (e.g. a string, a real number between zero and one, or a list of integers);
- Databases: containers of parameters, sub-databases, and sublists; and
- Sublists: a list of congruent databases (e.g. materials in a problem).

Each database has to conform to a specific “class”, such as the PROBLEM database or the SHIFT database. Some databases have different “types”: these are viewed as interchangeable by the enclosing database, but the different types may have different parameters associated with them. For example, the “SHAPE” sub-database of the SOURCE database could be a “box” type or a “cylinder” type. The SOURCE database requires a single shape sub-database, but each of those two types has a different set of required parameters (the dimensions of the box versus the width and height of the cylinder).

12.7 Developer notes

Exnihilo developers will need to modify the Omnibus input description to expose new transport capabilities to the front end. The input description is contained in the Python files at `Omnibus/frontend/omnibus/omn`. The root-level database is located in `root.py`, and sub-databases are located alongside it.

To update the ASCII input regression problems when the input database changes, run `make regression_update` from `Exnihilo/packages/Omnibus/frontend` inside the build directory.

12.7.1 Updating documentation

Additional documentation (besides the one-line parameter descriptions) can be added by expanding the files inside `Omnibus/frontend/doc/detail`. All of the `.rst` files will be read and treated equally; the multiple `.rst` files are just for convenience in editing and organizing the files.

The pseudo-directive `.. OMNIDOC {location}` is used to insert the following documentation (until the next `.. OMNIDOC` directive or the end of the file). The `{location}` is a simple syntax that is parsed to determine where in the Omnibus input documentation the contents should be inserted:

- Use `.` to separate subdatabases.
- Use `=` to specify a particular subtype for a database.
- The final entry can be a parameter or command, or it can be `_intro` to be printed at the top of a database description, or `_concl` to be added to the bottom.

Example locations are:

```
problem.mode
geometry=rtk._intro
source.shape=box.box
shift.decomposition.overlap
```

To regenerate the Omnibus input documentation, run `make omnidoc` from `Exnihilo/packages/Omnibus/frontend` inside the build directory.

CHAPTER
THIRTEEN

OMNIBUS ASCII INPUT FORMAT

The Omnibus ASCII input is a human-readable, minimal input syntax for Omnibus. The underlying Omnibus data structure is hierarchical, and the ASCII input is designed to flatten the hierarchy. The input consists of “blocks” of input data, each of which represents a database, and cards, which consist of parameters and “commands”, which generate parameters or perform other functions.

13.1 Blocks

Block titles have the formats

```
[CLASS]
[CLASS=type]
[CLASS name]
[PARENT] [CLASS=type name]
[GRANDPARENT] [PARENT] [CLASS=type name]
```

which embeds the location in the hierarchy, the database class, the database type, and the name of this particular instance of the database class. The “name” (which requires a value with only letters, numbers, and the underscore) is simply a shorthand for declaring the block and adding a “name” parameter. The class type is required for databases that have multiple allowed types (e.g. geometry and physics) but disallowed for types that do not. The block will be inserted inside the last instance of the [GRANDPARENT][PARENT] block.

Whitespace in block titles, as well as capitalization for the class and type attributes, is ignored.

Note: The exact regular expression used to match titles is:

```
^(?:\[\s*(\w+)(?:\s*=\s*(\w+))?\s*(\w+)?\]\s*)+$
```

13.2 Cards

Cards are started on a new line; an indentation of **four or more spaces** is treated as a continuation of the previous card. Spaces separate values in a parameter list or arguments in a command. For strings, quotation marks can be used to treat whitespace as standard characters. The backslash can be used to escape quotation marks inside a quoted string.

For example, these two parameters demonstrate the correct usage of whitespace:

```
param This is a list of seven parameters.
param "This is a single parameter with an \" embedded quotation mark."
```

Tip: One common input error is to mistake a small indentation on the next line for a continuation. This statement

declares three parameters inside a block:

```
[WAYNE]
something value value
    business business numbers
        is this working
```

whereas this is one parameter with multiple values:

```
[WAYNE]
something value value
    business business numbers
        is this working
```

Using the syntax highlighting files for Vim and Emacs provided in the Exnihilo/environment directory (see [Development Environment](#)) will make such errors very obvious.

13.3 Other features

Whitespace (outside of the initial line indentation and when within quotation marks) is generally flexible and ignored. Inside numerical lists, standard MCNP interpolation/repetition shorthands “I”, “ILOG”, “M”, and “R” are implemented:

```
x_coordinates 1 2I 4
```

is interpolated to form:

```
x_coordinates 1 2 3 4
```

Tip: The `vacuum_omnibus_input` script will read your Omnibus input file, reformat it, and rewrite it. If the input and output are not logically the same, there is a subtle syntax error in the input file (e.g. not indenting when continuing). This tool only parses the input file; it does no expansion, validation, or defaulting.

Tip: To view a validated and reformatted ASCII version of your input, you can explicitly tell the preprocessor to save an `.omn` file:

```
$ omnibus-pre problem.omn -o problem.validated.omn
```

OMNIBUS INPUT DATABASE SPECIFICATION

version 5.4 (r21399: #fbb1e564 on 2015JUL07)

date 2015-07-08 14:44:05

The Omnibus input is split into a hierarchy of blocks. Each of the first-level blocks (and the overall problem input file) are described in the following section:

14.1 Omnibus input file

14.1.1 Sub-databases

Name and type	Frequency
<i>problem</i>	Exactly once
<i>response</i> = { <i>histogram</i> , <i>interpolated</i> }	Zero or more
<i>tally</i>	Optional
<i>source</i> = { <i>separable</i> , <i>fissionmesh</i> , <i>mesh</i> , <i>mcnp</i> }	At least one
<i>geometry</i> = { <i>mcnp</i> , <i>scale</i> , <i>rtk</i> , <i>mesh</i> , <i>sword</i> }	Exactly once
<i>comp</i>	Optional
<i>physics</i> = { <i>smg</i> , <i>sce</i> , <i>void</i> }	At least one
<i>depletion</i>	Optional
<i>shift</i>	Optional
<i>denovo</i>	Optional
<i>manualww</i>	Optional
<i>run</i> = { <i>serial</i> , <i>mpi</i> , <i>pbs</i> , <i>titan</i> }	Optional

14.1.2 Additional defaults

- If running a kcode problem with only Denovo, no source needs to be defined.

- If running in mode `raytrace`, physics will be auto-set to void..

14.1.3 Additional restrictions

- The ‘denovo’ and ‘manualww’ databases are mutually exclusive..
- No transport will be run if [SHIFT] and [DENOVO] databases are both missing..
- Raytrace mode requires a [DENOVO] entry and no [SHIFT]..

14.2 [PROBLEM]

The problem database specifies top-level information about the problem being run. It includes the output file name, a unique problem identifier, and overall solution technique.

14.2.1 Parameters

name

Descriptive problem name.

Default ‘Untitled’

Type string

mode

Problem mode.

Valid modes are:

kcode Solve the k -eigenvalue problem for criticality safety or reactor physics analysis.

forward Solve a fixed-source problem for shielding calculations etc.

adjoint Solve an adjoint fixed-source problem with the [SOURCE] block interpreted as adjoint sources.

raytrace Use the Denovo ray tracer to generate voxelized materials for the problem. No transport will be performed.

Type ‘kcode’, ‘forward’, ‘adjoint’, or ‘raytrace’

screen_verbosity

Minimum level of output to print to screen.

Default ‘diagnostic’

Type ‘debug’, ‘diagnostic’, ‘status’, ‘info’, ‘warning’, ‘error’, or ‘critical’

log_verbosity

Minimum level of output to save to log file.

Default ‘info’

Type ‘debug’, ‘diagnostic’, ‘status’, ‘info’, ‘warning’, ‘error’, or ‘critical’

14.2.2 Advanced parameters

These parameters are not meant for typical use.

`xml_output / xml`

Destination path for the XML output file.

Type file path to write (extension ‘.xml’)

`pid`

Unique identifier automatically set for this problem run.

The problem identifier (pid) is a unique string generated by the Omnibus preprocessor to ensure that input and output files are properly correlated. The problem ID value added to the XML input file is copied to the XML output file as well as all relevant HDF5 output files. It’s comprised of the problem execution date and a randomly generated unique identifier string (UUID).

Type string

`exnihilo_rev`

Exnihilo revision used to generate this file..

Type integer

14.2.3 Additional defaults

- Default the XML output path to `input.out.xml`.
- Set the unique problem identifier for this run.

14.3 [RESPONSE]

The RESPONSE block is for defining energy- and particle-dependent responses such as dose conversion factors. Each response can be used multiple times in the tallies.

14.4 [RESPONSE=histogram]

14.4.1 Parameters

`name`

Short title or label for response.

Type string

`description`

Optional longer descriptive string.

Default

“ ”

Type string

`energy / e`

Lowest bound plus upper bounds for the histograms.

Type list of monotonically increasing floats

Units eV

response / r

Histogram response values.

Type list of floats

particle_type / pt

Particle type to apply this response.

Type particle type ('n', 'neutron', 'p', 'photon')

14.4.2 Additional restrictions

- Size of energy bounds must be one greater than the number of response values.

14.5 [RESPONSE=interpolated]

14.5.1 Parameters

name

Short title or label for response.

Type string

description

Optional longer descriptive string.

Default

“”

Type string

interpolation_type / interp

Type of interpolated response.

Type 'linear', 'log_lin', 'lin_log', or 'log_log'

energy / e

Energy points.

Type list of monotonically increasing floats

Units eV

response / r

Response values at energy points.

Type list of floats

particle_type / pt

Particle type to apply this response.

Type particle type ('n', 'neutron', 'p', 'photon')

14.5.2 Additional restrictions

- Response energies must be list of positive floats for `log_log` and `log_lin` response types.
- Response energies must be list of positive floats for `log_log` and `lin_log` response types.
- The parameters `energy`, `response` must have the same length..

14.6 [TALLY]

The tally database specifies tallies and problem diagnostics.

General tallies support a common set of attributes, including a name and optional description, a list of reactions to tally, a list of [\[RESPONSE\]](#) objects, and neutron and photon energy bin boundaries.

Note: Unlike other Monte Carlo transport codes, the bin boundaries in Omnibus are truly boundaries, not upper or lower energies. Thus the number of energy bins will be one less than the number of bounds. To reproduce the behavior of MCNP and Monaco, which tally from the cutoff energy to the lowest given energy, you must explicitly add the cutoff energy as the lowest energy bound.

Because multiple tallies can write to the same files, file names and prefixes are saved in the main [\[TALLY\]](#) block.

14.6.1 Sub-databases

Name and type	Frequency
<code>mesh</code>	Zero or more
<code>cylmesh</code>	Zero or more
<code>cell</code>	Zero or more
<code>diagnostic = { pathlength, collision, source, history, debug, fission_site }</code>	Zero or more
<code>sensitivity</code>	Optional

14.6.2 Parameters

`output`

Name of tally hdf5 output file.

Default ‘tallies.h5’

Type file path to write (extension ‘.h5’) (empty value allowed)

Applicability when at least one tally is present.

`mesh_silo_output`

Prefix name for mesh tally silo output files.

Default ‘meshtally’

Type string

Applicability when at least one MESH tally is present.

14.6.3 Additional restrictions

- [TALLY] block must contain at least one sublist (MESH, CELL, etc.).

14.6.4 [TALLY][MESH]

Shift supports multiple structured Cartesian path length mesh tallies. The mesh can be defined over all or part of the physical problem geometry. The Cartesian mesh tally is output at the end of a problem in Silo format for easy analysis using VisIt, and in HDF5 format for analysis.

Parameters

name

Short title or label for the tally.

Type string

description / desc

Optional longer descriptive string.

Default

“”

Type string

reactions / rxn

Reactions to calculate for this tally.

Default ['flux']

Type list of 'flux', 'total', 'absorption', 'scattering', 'fission', 'nu_fission', 'kappa_sigma', or
'kerma'

responses / resp

Responses for this tally.

Default []

Type list of strings

normalization

Constant multiplicative factor to apply to tally results.

Default 1.0

Type positive real number

neutron_bins / nbins

Energy bin boundaries for neutrons.

Default []

Type nonnegative floats in decreasing order

Units eV

photon_bins / pbins

Energy bin boundaries for photons.

Default []

Type nonnegative floats in decreasing order

Units eV

cycles

During what phase of a kcode problem this should be tallying.

Default ‘active’

Type ‘active’, or ‘inactive’

Applicability when problem mode is ‘kcode’.

x

Mesh tally coordinates along the X axis.

Type list of two or more monotonically increasing floats

Units cm

y

Mesh tally coordinates along the Y axis.

Type list of two or more monotonically increasing floats

Units cm

z

Mesh tally coordinates along the Z axis.

Type list of two or more monotonically increasing floats

Units cm

Additional restrictions

- Response names are validated against [RESPONSE] blocks.

14.6.5 [TALLY][CYLMESH]

Particles can be tracked on a translated, rotated cylinder broken into (r, z, θ) mesh cells.

Parameters**name**

Short title or label for the tally.

Type string

description / desc

Optional longer descriptive string.

Default

“

Type string

reactions / rxn

Reactions to calculate for this tally.

Default ['flux']

Type list of 'flux', 'total', 'absorption', 'scattering', 'fission', 'nu_fission', 'kappa_sigma', or
'kerma'

responses / resp

Responses for this tally.

Default []

Type list of strings

normalization

Constant multiplicative factor to apply to tally results.

Default 1.0

Type positive real number

neutron_bins / nbins

Energy bin boundaries for neutrons.

Default []

Type nonnegative floats in decreasing order

Units eV

photon_bins / pbins

Energy bin boundaries for photons.

Default []

Type nonnegative floats in decreasing order

Units eV

cycles

During what phase of a kcode problem this should be tallying.

Default 'active'

Type 'active', or 'inactive'

Applicability when problem mode is 'kcode'.

r

Radial mesh coordinates.

Type list of two or more monotonically increasing floats

Units cm

theta

Theta mesh coordinates.

Default [0.0, 1.0]

Type list of two or more monotonically increasing floats

Units revolutions

z

Mesh coordinates along the Z axis.

Type list of two or more monotonically increasing floats

Units cm

rotate / rot

Rotation matrix.

Default [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]

Type row-major flattened matrix

translate / trans

Translation vector (applied after rotation).

Default [0.0, 0.0, 0.0]

Type xyz coordinates

Additional restrictions

- Response names are validated against [RESPONSE] blocks.

14.6.6 [TALLY][CELL]

Geometry cells and unions of cells are tallied using a hash table, allowing constant-time scaling with respect to the number of total cells being tallied.

The recommended way to tally multiple particle spectra in the same cell is to use a single tally.

```
[TALLY] [CELL energybinned]
description "5n3g energy-binned tally."
reactions flux
cells 1 100 4:5:6
neutron_bins 1e7 1e6 1e3 10 1 1e-3
photon_bins 2e7 1e6 1e5 1e3
```

Parameters

name

Short title or label for the tally.

Type string

description / desc

Optional longer descriptive string.

Default

“”

Type string

reactions / rxn

Reactions to calculate for this tally.

Default ['flux']

Type list of ‘flux’, ‘total’, ‘absorption’, ‘scattering’, ‘fission’, ‘nu_fission’, ‘kappa_sigma’, or
‘kerma’

responses / resp

Responses for this tally.

Default []

Type list of strings

normalization

Constant multiplicative factor to apply to tally results.

Default 1.0

Type positive real number

neutron_bins / nbins

Energy bin boundaries for neutrons.

Default []

Type nonnegative floats in decreasing order

Units eV

photon_bins / pbins

Energy bin boundaries for photons.

Default []

Type nonnegative floats in decreasing order

Units eV

cycles

During what phase of a kcode problem this should be tallying.

Default ‘active’

Type ‘active’, or ‘inactive’

Applicability when problem mode is ‘kcode’.

Advanced parameters

These parameters are not meant for typical use.

union_cells

Flattened list of cells in each union.

Type list of integers

union_lengths

Number of cells per union in the above list.

Type list of integers

Commands

These commands generate one or more parameters.

cells

Generate ‘union_cells’ and ‘union_lengths’ from colon-separated unions.

Additional restrictions

- Response names are validated against [RESPONSE] blocks.

14.6.7 [TALLY][DIAGNOSTIC=pathlength]

The path length diagnostic produces a distribution of the path length traveled by a particle between events. Note that this is not the same as the true path length distribution (distance between collisions), as the events considered by Shift include material boundary crossings, problem boundary crossings, etc.

Parameters

p1_bins

Lower bin boundaries for the traversed path lengths in each event.

Type list of positive floats

event_bins

Lower bin boundaries for the number of events per history.

Type list of monotonically increasing nonnegative integers

14.6.8 [TALLY][DIAGNOSTIC=collision]

The collision diagnostic tallies the number of collisions per history as a function of material, nuclide, and reaction ID.

Parameters

output

Path to collision diagnostic hdf5 output file.

Default ‘collisions.h5’

Type file path to write (extension ‘.h5’) (empty value allowed)

Additional restrictions

- This tally is only compatible with SCE physics.

14.6.9 [TALLY][DIAGNOSTIC=source]

This diagnostic tally is provided to calculate the source density binned into spatial cells. It also calculates the *particle* source density (i.e., binning $n(\vec{r})$ rather than $wn(\vec{r})$) to assist in the construction of biased sources.

Parameters

x

Mesh tally coordinates along the X axis.

Type list of two or more monotonically increasing floats

Units cm

y

Mesh tally coordinates along the Y axis.

Type list of two or more monotonically increasing floats

Units cm

z

Mesh tally coordinates along the Z axis.

Type list of two or more monotonically increasing floats

Units cm

silo_output

Path to Silo output file (no extension).

Default ‘source’

Type string

14.6.10 [TALLY][DIAGNOSTIC=history]

The history diagnostic tallies every event in a particle’s lifetime. Currently, all particle histories are tallied, and only one processor writes the histories.

Parameters

history_hdf5_output / output

Destination file for history events.

Default ‘history.h5’

Type file path to write (extension ‘.h5’) (empty value allowed)

domain

Domain on which history events will be saved.

Default 0

Type non-negative integer

begin_history

First history for which events will be saved.

Default 0

Type non-negative integer

end_history

Last + 1 history for which events will be saved.

Default 0

Type non-negative integer

14.6.11 [TALLY][DIAGNOSTIC=debug]

The debug diagnostic currently records the number of events per particle history, but it will be extended to provide other useful high-level debug information. The output data is written to the omnibus output XML file and converted in the postprocessor.

14.6.12 [TALLY][DIAGNOSTIC=fission_site]

Parameters

write_cycles / wc

List of cycles in which to write the fission source.

Default []

Type list of nonnegative integers

output

Filename for cycle-by-cycle fission source output.

Default ‘fission_site.h5’

Type string

Additional restrictions

- This tally is only applicable to mode kcode.

14.6.13 [TALLY][SENSITIVITY]

The sensitivity tally allows sensitivities to cross sections and other input to be calculated using advanced methods.

Note: Currently sensitivity tallies can only be run in serial mode.

Parameters

neutron_bins / nbins

Energy bin boundaries for neutrons.

Default []

Type nonnegative floats in decreasing order

Units eV

method / cet

Sensitivity coefficient calculation mode.

Default ‘ce_tsunami’

Type ‘ce_tsunami’, ‘clutch’, or ‘ifp’

latent_generations / cfp

Number of latent generations for IFP/F*(r) calculation.

Default 1

Type positive integer

Applicability when S/U method uses an F* mesh.

Advanced parameters

These parameters are not meant for typical use.

sensitivity_output

Destination path for the sensitivity tally output file (.sdf).

Type file path to write (extension ‘.sdf’)

Additional defaults

- Default the SDF output path to INPUT.sdf.

Additional restrictions

- This tally is only compatible with SCE physics.

14.7 [SOURCE]

The source database specifies the source particle distribution for a fixed-source problem, as well as the starting source for an eigenvalue problem. Any combination of types is allowed.

The total strength of all sources is used as the typical particle starting weight.

Warning: Adjusting the total source strength means adjusting the average particle weight. One implication of this is that, because variance reduction cutoffs are not normalized to the source strength, roulette will allow particles to live for longer if the source strength is higher. Correspondingly, if the total source strength is less than 0.25, the default value for the `weight_cutoff` parameter of russia roulette, the average particle will be rouletted immediately upon birth.

It is therefore very important to adjust your [\[SHIFT\]/\[VR\]](#) options if you change the source strength. (If using CADIS or FW-CADIS, the weight windows are automatically normalized based on the source strength.)

An alternative to modifying the total source strength is to modify the `normalization` property of the tallies.

14.8 [SOURCE=separable]

14.8.1 Sub-databases

Name and type	Frequency
<code>shape = { box, cylinder, cylindershell, sphere, sphereshell, point, global }</code>	Exactly once
<code>energy = { histogram, mono, lines, watt, origin }</code>	Exactly once
<code>angle = { isotropic, mono }</code>	Exactly once

14.8.2 Parameters

name

Short title or label for the source.

Default ‘source’

Type string

description / desc
Optional longer descriptive string.

Default
“”

Type string

strength / q
Source strength.

Default 1.0

Type positive real number

Units particles/s

particle_type / pt
Particle type to emit.

Type particle type ('n', 'neutron', 'p', 'photon')

fissionable_only / fis
Whether particles are only emitted in fissionable regions.

Type boolean

denovo_source_type
Type of Denovo source discretization.

Default ‘default’

Type ‘default’, ‘volumetric’, ‘uncollided_mc’, or ‘uncollided_analytic’

14.8.3 Additional defaults

- When running an eigenvalue problem, Default to an isotropic distribution and a Watt (U-235) energy spectrum, and only emit particles inside of fissionable cells.
- Default the denovo source discretization scheme based on the separable source shape.

14.8.4 [SOURCE][SHAPE=box]

Parameters

xmin
Minimum x-coordinate of box source.

Type real number

xmax
Maximum x-coordinate of box source.

Type real number

ymin
Minimum y-coordinate of box source.

Type real number

y_{max}

Maximum y-coordinate of box source.

Type real number

z_{min}

Minimum z-coordinate of box source.

Type real number

z_{max}

Maximum z-coordinate of box source.

Type real number

Commands

These commands generate one or more parameters.

box

Expand into parameters x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}.

14.8.5 [SOURCE][SHAPE=cylinder]

The cylinder shape is defined along one of the three cartesian axes. Its origin (base) is defined as the center point of the lower circular face. The height extends in the positive direction from the origin along the given axis.

Parameters

axis

Axis along regular cylinder source.

Type axis ('x','y','z')

origin_x / xo

X-coordinate of regular cylinder source origin.

Type real number

origin_y / yo

Y-coordinate of regular cylinder source origin.

Type real number

origin_z / zo

Z-coordinate of regular cylinder source origin.

Type real number

radius / r

Radius of regular cylinder source.

Type real number

height / h

Height of regular cylinder source.

Type real number

Commands

These commands generate one or more parameters.

origin

Expand into parameters `origin_x`, `origin_y`, `origin_z`.

14.8.6 [SOURCE][SHAPE=cylindershell]

Parameters

axis

Axis along regular cylinder shell source.

Type axis ('x','y','z')

origin_x / xo

X-coordinate of regular cylinder shell source origin.

Type real number

origin_y / yo

Y-coordinate of regular cylinder shell source origin.

Type real number

origin_z / zo

Z-coordinate of regular cylinder shell source origin.

Type real number

inner_radius / ir

Inner radius of regular cylinder shell source.

Type real number

outer_radius / or

Outer radius of regular cylinder shell source.

Type real number

height / h

Height of regular cylinder shell source.

Type real number

Commands

These commands generate one or more parameters.

origin

Expand into parameters `origin_x`, `origin_y`, `origin_z`.

14.8.7 [SOURCE][SHAPE=sphere]

Parameters

origin_x / xo

X-coordinate of sphere source origin.

Type real number

origin_y / yo

Y-coordinate of sphere source origin.

Type real number

origin_z / zo

Z-coordinate of sphere source origin.

Type real number

radius / r

Radius of sphere source.

Type real number

Commands

These commands generate one or more parameters.

origin

Expand into parameters `origin_x`, `origin_y`, `origin_z`.

14.8.8 [SOURCE][SHAPE=sphereshell]

Parameters

origin_x / xo

X-coordinate of sphere shell source origin.

Type real number

origin_y / yo

Y-coordinate of sphere shell source origin.

Type real number

origin_z / zo

Z-coordinate of sphere shell source origin.

Type real number

inner_radius / ir

Inner radius of sphere shell source.

Type real number

outer_radius / or

Outer radius of sphere shell source.

Type real number

Commands

These commands generate one or more parameters.

origin

Expand into parameters `origin_x`, `origin_y`, `origin_z`.

14.8.9 [SOURCE][SHAPE=point]

Parameters

x

X position of point source.

Type real number

y

Y position of point source.

Type real number

z

Z position of point source.

Type real number

Commands

These commands generate one or more parameters.

point

Expand into parameters x, y, z.

14.8.10 [SOURCE][SHAPE=global]

The “global” source is a labor-unintensive way to attempt to sample particles inside the geometry. It works by querying the geometry for a bounding box (currently, only RTK and SCALE geometries are supported), then sampling points uniformly inside that box. Any point outside the geometry will be rejected. The `sample_attempts` parameter determines how many samples inside the bounding box to attempt before a warning is emitted.

Warning: If the bounding box of the geometry is much different than the underlying geometry, the rejection fraction may be high, and the source sampling may be very slow. This is especially true in the case of fissionable-only sources, where an additional rejection step is overlaid.

Parameters

sample_attempts

Number of geometry positions to sample per source particle.

Default 1000

Type positive integer

14.8.11 [SOURCE][ENERGY=histogram]

Parameters

energy / e

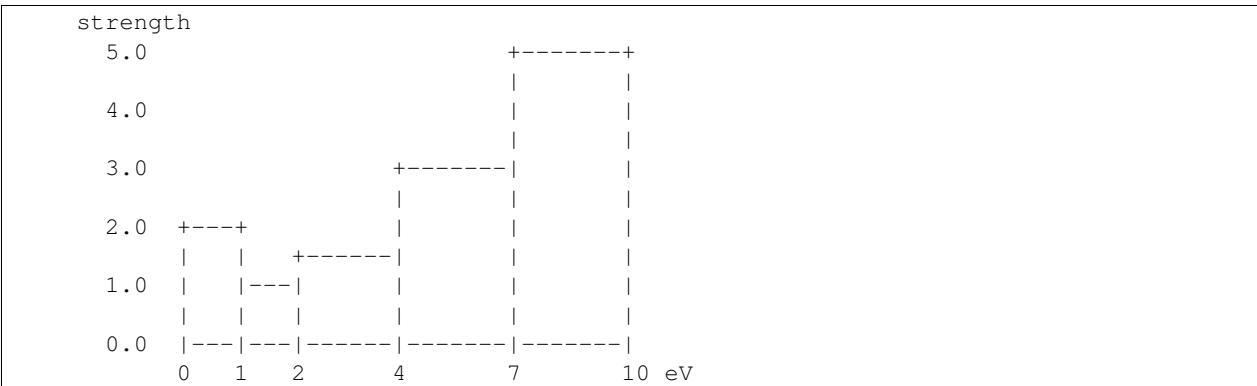
Histogram lower energy bin bounds.

Type list of non-negative monotonically increasing floats

Units eV**probability / p**

Probability of an energy bin being selected.

For the discrete source distribution that looks like:



The Omnibus input and integrated probability for each bin will be:

energy:probability	! strength
0 0.067	! p * 30 / 1 = 2
1 0.033	! p * 30 / 1 = 1
2 0.1	! p * 30 / 2 = 1.5
4 0.3	! p * 30 / 3 = 3.
7 0.4	! p * 30 / 3 = 4.
10 0.5	! p * 30 / 3 = 5.

Type list of non-negative floats

Additional restrictions

- Histogram source must have one more energy value than probability.

14.8.12 [SOURCE][ENERGY=mono]

Parameters

energy / e

Source energy.

Type positive real number

14.8.13 [SOURCE][ENERGY=lines]

Parameters

energy / e

Individual line energies.

Type list of positive floats

Units eV

probability / p

Probability of each line being selected.

Type list of non-negative floats

Additional restrictions

- The parameters `energy`, `probability` must have the same length..

14.8.14 [SOURCE][ENERGY=watt]

The Watt distribution samples a fission spectrum:

$$f(E) = c \exp(-E/a) \sinh(\sqrt{bE})$$

where c is a normalization constant.

Parameters**a**

Value for the ‘a’ constant in Watt equation.

Type positive real number

Units MeV

b

Value for the ‘b’ constant in Watt equation.

Type positive real number

Units 1/MeV

Additional defaults

- Watt spectrum defaults to a=0.965 and b=2.29 (U-235 fission).

14.8.15 [SOURCE][ENERGY=origen]**Parameters****filename**

File (ft71) containing the ORIGEN-generated spectrum.

Type file path for reading

statepoint

Statepoint in the file from which to read the source.

Type non-negative integer

override_strength

Indicates whether to let the ORIGEN source strength override user-specified strength..

Default True

Type boolean

14.8.16 [SOURCE][ANGLE=isotropic]

14.8.17 [SOURCE][ANGLE=mono]

Parameters

direction / dir

Direction source particles are emitted.

Type xyz coordinates

14.9 [SOURCE=fissionmesh]

The fission mesh source allows a fission neutron production mesh tally from a prior run (kcode or fixed source) to be used as a source. This source could either be a starting source in a kcode calculation (to reduce the number of inactive cycles needed) or a fixed source (so that a coupled neutron-gamma problem can be run separately from the kcode calculation). The `input` parameter points to the name of the `tallies.h5` file; and the `mesh_tally_name` parameter should be the name of the fission mesh tally.

By default, this source uses a U-235 energy spectrum.

14.9.1 Sub-databases

Name and type	Frequency
<code>energy = { histogram, mono, lines, watt, origen }</code>	Exactly once

14.9.2 Parameters

name

Short title or label for the source.

Default ‘source’

Type string

description / desc

Optional longer descriptive string.

Default

‘’

Type string

strength / q

Source strength.

Default 1.0

Type positive real number

Units particles/s

particle_type / pt

Particle type to emit.

Type particle type (‘n’, ‘neutron’, ‘p’, ‘photon’)

fissionable_only / fis

Whether particles are only emitted in fissionable regions.

Type boolean

denovo_source_type

Type of Denovo source discretization.

Default ‘default’

Type ‘default’, ‘volumetric’, ‘uncollided_mc’, or ‘uncollided_analytic’

mesh_tally_input / input

Name of file containing fission source distribution.

Type file path for reading (extension ‘.h5’)

mesh_tally_name

Name of fission source tally in file.

Type string

14.9.3 Additional defaults

- Default to an isotropic distribution and a Watt (U-235) energy spectrum, and only emit particles inside of fissionable cells.

14.9.4 [SOURCE][ENERGY=histogram]

Parameters

energy / e

Histogram lower energy bin bounds.

Type list of non-negative monotonically increasing floats

Units eV

probability / p

Probability of an energy bin being selected.

Type list of non-negative floats

Additional restrictions

- Histogram source must have one more energy value than probability.

14.9.5 [SOURCE][ENERGY=mono]

Parameters

energy / e

Source energy.

Type positive real number

14.9.6 [SOURCE][ENERGY=lines]

Parameters

energy / e

Individual line energies.

Type list of positive floats

Units eV

probability / p

Probability of each line being selected.

Type list of non-negative floats

Additional restrictions

- The parameters `energy`, `probability` must have the same length..

14.9.7 [SOURCE][ENERGY=watt]

Parameters

a

Value for the ‘a’ constant in Watt equation.

Type positive real number

Units MeV

b

Value for the ‘b’ constant in Watt equation.

Type positive real number

Units 1/MeV

Additional defaults

- Watt spectrum defaults to $a=0.965$ and $b=2.29$ (U-235 fission).

14.9.8 [SOURCE][ENERGY=origen]

Parameters

filename

File (ft71) containing the ORIGEN-generated spectrum.

Type file path for reading

statepoint

Statepoint in the file from which to read the source.

Type non-negative integer

override_strength

Indicates whether to let the ORIGEN source strength override user-specified strength..

Default True

Type boolean

14.10 [SOURCE=mesh]

14.10.1 Parameters

name

Short title or label for the source.

Default ‘mesh_source’

Type string

mesh_source_input / input

Name of file containing mesh source distribution.

Type file path for reading (extension ‘.h5’)

14.11 [SOURCE=mcnp]

The MCNP source uses the Lava library to extract and use the source definition from an SDEF card. It supports automatic interpretation of the particle type (defaulting to the problem mode) and the strength (from the WGT parameter) as well as the actual distributions themselves.

Note: Currently, the MCNP source is only available if the MCNP geometry is being used.

14.11.1 Parameters

name

Short title or label for the source.

Default ‘mcnp_source’

Type string

description / desc

Optional longer descriptive string.

Default

“

Type string

14.11.2 Additional restrictions

- An MCNP source can currently only be used with an MCNP geometry..

14.12 [GEOMETRY]

Shift provides adapters for several geometry types.

Note: For most geometry types, Shift allows cell volumes to be manually input with the “volumes” and “volume_cells” keywords:

volumes	1.0	2.34	0.5
volume_cells	1	200	15

Here the cells are the “cell labels” (e.g. the cell card IDs in MCNP) and the volumes are the corresponding volumes in cm³.

14.13 [GEOMETRY=mcnp]

MCNP support is provided through the Lava plugin, developed at ORNL by Scott Mosher. Lava is a C interface to MCNP routines. It requires MCNP to generate a run tape from an input file. Omnibus will automatically execute MCNP and generate the run tape.

Note: If using Shift cell tallies with MCNP geometry, the user must include a cell tally or input volumes for desired cells *in the MCNP input* to for volumes to automatically be propagated into Shift.

14.13.1 Sub-databases

Name and type	Frequency
movable = { <i>surfaces</i> }	Zero or more

14.13.2 Parameters

cell_raytrace

Transform cell labels into material IDs for raytracing.

Default False

Type boolean

Applicability when problem mode is ‘raytrace’.

mat_names

Override names for materials in the geometry.

Custom names can be added to the geometry that will be reflected in material plots inside VisIT and inside postprocessing data elements. MCNP material card numbers are input along with new names to use:

mat_name_mno : mat_names
1 "sodium iodide"
2 "carbon steel"
13 polyethylene
14 iron
105 air
106 concrete

Default []**Type** list of strings**mat_name_mno**

MCNP material names corresponding to the given mat name overrides.

Default []**Type** list of MCNP material numbers**mat_colors**

Add colors for visualization of materials in the geometry.

VisIT supports custom colors to be associated with each material in its plotting routines. Non-specified materials will be filled with black. Specified non-materials will generate an error:

```
mat_color_mno : mat_colors
 1 "light blue"
 2 "dark grey"
 13 "#aaeeff"
 14 "DarkSlateGray"
 105 "#5555FF"
 106 "pale goldenrod"
```

Valid color names are either HTML-style hexadecimal RGB tuples, or *X11 colors*. .. _X11 colors: https://en.wikipedia.org/wiki/X11_color_namesFor complicated inputs with variable materials, it may be wise to generate the colors using a python script and the `matplotlib.colors` module.**Default** []**Type** list of X11 color, HTML color like #FF00ee, or empty**mat_color_mno**

MCNP material names corresponding to the given mat colors.

Default []**Type** list of MCNP material numbers**volumes**

Provide or override volumes for cells in the geometry.

Default []**Type** list of positive floats**Units** cc**volume_cells**

Cell labels corresponding to the given volume overrides.

Default []**Type** list of MCNP cell numbers

14.13.3 Advanced parameters

These parameters are not meant for typical use.

runtpe_path

Path to the MCNP runtpe file.

Type file path for reading

14.13.4 Commands

These commands generate one or more parameters.

input

Generate an MCNP runtpe file and set runtpe_path.

14.13.5 Additional restrictions

- The parameters volumes, volume_cells must have the same length..
- The parameters mat_names, mat_name_mno must have the same length..
- The parameters mat_colors, mat_color_mno must have the same length..

14.13.6 [GEOMETRY][MOVABLE=surfaces]

Apply translation to multiple surfaces simultaneously in the MCNP geometry. Currently this works only for simple axis-aligned planes.

Warning: Because translations and surface deduplication are applied while generating the runtpe file, moving one surface label may end up affecting other coincident surfaces.

Parameters

name

Name of the surface group.

Type string

surfaces

List of MCNP surface labels to move over time.

Type list of positive integers

initial

Movement to apply before the first transport.

Default 0.0

Type real number

Units cm

14.14 [GEOMETRY=scale]

The SCALE geometry supports reading directly from a SCALE input file that includes a KENO VI geometry. Contact Rob Lefebvre <lefebvrera@ornl.gov> for the current status of the Atlas geometry package that underlies KENO support. As of December 2014, the following capabilities are not supported:

- Reflecting boundaries

- Hexagonal arrays

14.14.1 Parameters

input

Path to the KENO VI input file.

Type file path for reading (extension ‘.inp’)

volumes

Provide or override volumes for cells in the geometry.

Default []

Type list of positive floats

Units cc

volume_cells

Cell labels corresponding to the given volume overrides.

Default []

Type list of MCNP cell numbers

14.14.2 Additional restrictions

- The parameters `volumes`, `volume_cells` must have the same length..

14.15 [GEOMETRY=rtk]

The RTK geometry implemented by Omnibus reads a geometry from an XML input file. In general, the Insilico front-end should be used for creating reactors with the RTK geometry.

14.15.1 Parameters

input

Path to the RTK geometry xml file.

Type file path for reading (extension ‘.xml’)

14.16 [GEOMETRY=mesh]

14.16.1 Parameters

input_type

Where the mesh geometry is defined.

Default ‘hdf5’

Type ‘hdf5’, or ‘manual’

input

Path to the mesh geometry hdf5 file.

Type file path for reading (extension ‘.h5’)

Applicability when ‘input_type’ is ‘hdf5’.

dimension

Dimension of mesh.

Default 3

Type integer 2 or 3

Applicability when mesh input is manual.

matids

List of material IDs for each mesh cell.

Type list of nonnegative integers

Applicability when mesh input is manual.

x_planes

Mesh coordinates along the X axis.

Type list of two or more monotonically increasing floats

Units cm

Applicability when mesh input is manual.

y_planes

Mesh coordinates along the Y axis.

Type list of two or more monotonically increasing floats

Units cm

Applicability when mesh input is manual.

z_planes

Mesh coordinates along the Z axis.

Type list of two or more monotonically increasing floats

Units cm

Applicability when mesh input is manual; and mesh dimension is 3.

14.16.2 Additional restrictions

- If more than 1000 mesh cells are present, then HDF5 input must be used.

14.17 [GEOMETRY=sword]

14.17.1 Advanced parameters

These parameters are not meant for typical use.

xdr_path

Path to the SWORD binary input file.

Type file path for reading (extension ‘.xdr’)

14.17.2 Commands

These commands generate one or more parameters.

input

Generate a binary SWORD representation and set `xdr_path`.

14.18 [COMP]

The `[COMP]` block allows custom definition of compositions. The matids in each block must correspond to the matids in the problem geometry.

14.18.1 Sub-databases

Name and type	Frequency
<i>material</i>	At least one

14.18.2 Additional restrictions

- The parameters `nd`, `zaid` must have the same length..

14.18.3 [COMP][MATERIAL]

Parameters

name

Label for the material.

Type string

matid

Internal matid number.

The `matid` is a [0,N)-indexed internal numbering system. The matids used by Shift typically differ from the material names used in the problem physics input (for example, `m10` in an MCNP input deck might correspond to `matid=1`). Currently, the only way to guarantee this corresponds to a particular material in the geometry input is to use an input that specifies matids explicitly (RTK or mesh geometry). However, by viewing the matid-to-label mapping given in an Omnibus postprocess output for a SCALE or MCNP input geometry, it is possible to figure out for a particular problem what matid corresponds to what material.

Type non-negative integer

temperature / tmp

Material temperature.

Type non-negative real number

Units K

deplete / depel

Whether the material is depletable.

Default False

Type boolean

fission / fiss

Whether the material is fissionable.

Default False

Type boolean

zaid

Element IDs (MZZZAAA) in this material.

Type list of positive integers

nd

Number densities of each nuclide.

Type list of positive floats

Units atoms/(b-cm)

color

Color for the material.

Default

“”

Type X11 color, HTML color like #FF00ee, or empty

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
nd		num_densities
zaid		elements

14.19 [PHYSICS]

Omnibus currently supports two coupled-physics packages: SCALE multigroup (SMG) and SCALE continuous energy (SCE). The SMG physics also supports manually input cross sections, either through an input database in the .omn file, through an XML input file, or through AMPX/GIP working library files.

14.20 [PHYSICS=smg]

The SCALE MG physics package by default uses SCALE to calculate infinite homogeneous medium cross sections for all materials. No self-shielding is used, so results will generally not have good accuracy. The default cross section processing is primarily intended for generating deterministic solutions for hybrid calculations.

When used with Monte Carlo physics, SMG replicates the multigroup physics used in KENO, including replicating its idiosyncrasies. For example, do not expect P1 to work correctly: when collisions happen, the exiting cosine is *set* to the mean cosine rather than sampled from it. Thus an isotropically scattering medium in a P1 problem will have all collisions happen perpendicular to their original direction.

14.20.1 Sub-databases

Name and type	Frequency
<i>xs</i>	Zero or more
<i>gip</i>	Optional

14.20.2 Parameters

name

Label for the physics.

Default ‘mg’

Type string

mg_lib_path

MG library name or path to MG library file.

Type file path for reading

Applicability when ‘xsgen’ is ‘processed’.

gip_filepath

Path to a GIP cross section input file.

Type file path for reading (extension ‘.ampx’)

Applicability when ‘xsgen’ is ‘gip’.

ampx_filepath

Path to an AMPX cross section input file.

Type file path for reading (extension ‘.ampx’)

Applicability when ‘xsgen’ is ‘ampx’.

xml_filepath

Path to an XML cross section input file.

Type file path for reading (extension ‘.xml’)

Applicability when ‘xsgen’ is ‘xml’.

implicit_capture

Enable implicit capture for Monte Carlo transport.

Default True

Type boolean

disable_upscattering / noup

Disable upscattering: default adds upscatter to within-group.

Default False

Type boolean

subtract_upscattering / subup

Disable upscatter by subtracting it from the total.

Default False

Type boolean

Applicability when ‘disable_upscattering’ is ‘True’.

pn_order

Scattering order of problem.

Default 0

Type zero or positive odd integer

comp_hdf5_output / comp_out

Write compositions to the given HDF5 file, or skip if empty.

Default ‘compositions.h5’

Type file path to write (extension ‘.h5’) (empty value allowed)

num_groups

Number of energy groups.

Type non-negative integer

Applicability when ‘xsgen’ is ‘xml’ or ‘inline’ or ‘ampx’ or ‘gip’.

neutron_bnd

Neutron group boundaries.

Default []

Type positive floats in decreasing order

Applicability when ‘xsgen’ is ‘xml’ or ‘inline’ or ‘ampx’ or ‘gip’.

photon_bnd

Photon group boundaries.

Default []

Type positive floats in decreasing order

Applicability when ‘xsgen’ is ‘xml’ or ‘inline’ or ‘ampx’ or ‘gip’.

14.20.3 Advanced parameters

These parameters are not meant for typical use.

xsgen

Type of cross section data input.

The `xsgen` parameter specifies in what format the cross sections are defined for multigroup physics. The options are:

Option	Description
inline	The <code>[XS]</code> subdatabase is used to define cross sections for every material.
ampx	The cross sections for every material are defined in AMPX format. This file is specified using the <code>ampx_filepath</code> parameter.
gip	The cross sections for every material are defined in GIP format. This file is specified using the <code>gip_filepath</code> parameter. Other required parameters describing the file must be in the [GIP] sub-database.
process	The cross sections use the SCALE multigroup libraries. The processed library is specified by the <code>mg_lib</code> parameter.
xml	The cross sections for every material are defined in a separate XML file format. This file is specified using the <code>xml_filepath</code> parameter.

Based on which filepath parameters and databases are present, the front end will attempt to determine the correct `xsgen` option.

Type ‘ampx’, ‘gip’, ‘inline’, ‘processed’, or ‘xml’

14.20.4 Commands

These commands generate one or more parameters.

`mg_lib`

Set `mg_lib` to the given value using SCALE DATA resolution.

14.20.5 Additional defaults

- Automatically set `xsgen` based on given parameters..
- Automatically set `num_groups` based on given group bounds..

14.20.6 Additional restrictions

- [XS] sublists are only for `xsgen inline`, and [GIP] database is only for `xsgen gip`..
- Inline and GIP format XS definitions require group bounds via the ‘neutron_bnd’ and/or ‘photon_bnd’ parameters.
- XS or COMP blocks must be defined if using ‘rtk’, ‘mesh’ geometry.

14.20.7 Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
<code>disable_upscattering</code>	db	<code>downscatter</code>
<code>mg_lib_path</code>	db	<code>xs_library</code>
<code>neutron_bnd</code>	db	<code>neutron_bnd</code>
<code>num_groups</code>	db	<code>num_groups</code>
<code>photon_bnd</code>	db	<code>photon_bnd</code>
<code>pn_order</code>	db	<code>Pn_order</code>
<code>subtract_upscattering</code>	db	<code>subtract_upscattering</code>

14.20.8 [PHYSICS][XS]

This subdatabase specifies all cross sections for a single material. If the material is fissionable, all of `fission`, `nu`, and `chi` must be present. If not fissionable, none of them can be present.

Parameters

`name`

Label for the material.

Default ‘untitled’

Type string

`matid`

Material ID.

Type non-negative integer

`total`

Total cross section by group.

Type list of non-negative floats

`s0`

Isotropic scattering cross section.

Type list of non-negative floats

`fission`

Fission cross section.

Default []

Type list of non-negative floats

`nu`

Neutron production.

Default []

Type list of non-negative floats

`chi`

Fission spectrum.

Default []

Type list of non-negative floats

Additional restrictions

- The parameters `total`, `fission`, `nu`, `chi` must have the same length. Empty lists are ignored..

14.20.9 [PHYSICS][GIP]

Parameters

`num_materials`

Number of materials in GIP xs file.

Type non-negative integer

iht
Position of total in GIP xs file.

Type non-negative integer

ihs
Position of within-group scattering in GIP xs file.

Type non-negative integer

ihm
Position of ending downscattering groups in GIP xs file.

Type non-negative integer

num_thermal
Number of column width of thermal groups in GIP xs file.

Type non-negative integer

n
Pn order in GIP xs file.

Type non-negative integer

14.21 [PHYSICS=sce]

SCE is the new interface to SCALE continuous energy physics.

14.21.1 Sub-databases

Name and type	Frequency
splice = { <i>ampx</i> }	Zero or more

14.21.2 Parameters

name
Label for the physics.

Default ‘sce’

Type string

ce_lib_path
Path to SCALE CE Library XML or HDF5 file.

Type file path for reading (extension ‘.xml’ or ‘.h5’)

mode
Particle transport mode.

Type particle transport mode (‘n’, ‘neutron’, ‘np’, ‘p’, ‘photon’, ‘pn’)

comp_hdf5_output / comp_out
Write compositions to the given HDF5 file, or skip if empty.

Default ‘compositions.h5’

Type file path to write (extension ‘.h5’) (empty value allowed)

probability_tables / ptab

Use probability table method for unresolved resonances.

Default True

Type boolean

preserve_bands

Preserve probability table bands for nuclides across materials.

This option changes the behavior of probability tables when streaming through multiple materials. In some codes such as KENO, streaming from one material to another resets the probability band values. As a consequence, if a particle streams through two instances of the same material by traveling through a void, its probability table band will be resampled in the second material.

When enabled, this option builds a list of nuclides with probability table data (and in which the particle’s energy is inside the URR) encountered since the last collision. If the same nuclide is encountered, the band index is preserved. This is more physical but might have a small performance impact.

Note that this method does not consider temperature dependence; if a neutron is in a particular band for one temperature of a nuclide, it will remain at that temperature in another.

Default True

Type boolean

Applicability when ‘probability_tables’ is ‘True’.

otf_elastic_scattering

Use on-the-fly elastic scattering rather than table lookups.

Default False

Type boolean

xs_temp_correction_mode

Method of temperature correction for cross-section data.

•none = No correction.

•oned = Correct 1D cross-sections, collision probabilities, and probability tables only.

•all = Correct 2D scattering data in addition to above.

Default ‘none’

Type ‘none’, ‘oned’, or ‘all’

n_energy_min / n_emin

Minimum global neutron energy cutoff for cross sections.

Default 1e-05

Type positive real number

Units eV

Applicability when ‘mode’ is ‘np’ or ‘n’.

n_energy_max / n_emax

Maximum global neutron_energy cutoff for cross sections.

Default 20000000.0

Type positive real number

Units eV**Applicability** when ‘mode’ is ‘np’ or ‘n’.**p_energy_min / p_emin**

Minimum global photon energy cutoff for cross sections.

Default 10000.0**Type** positive real number**Units** eV**Applicability** when ‘mode’ is ‘np’ or ‘p’.**p_energy_max / p_emax**

Maximum global photon energy cutoff for cross sections.

Default 25000000.0**Type** positive real number**Units** eV**Applicability** when ‘mode’ is ‘np’ or ‘p’.**thermal_energy_cutoff**

Thermalization energy cutoff for scattering kernels.

Default 10.0**Type** positive real number**Units** eV**orig_zaid_n**

Replace CE data for these nuclides in problem materials.

The ZAID remapping options allow a ZAID present in the problem input to be replaced with a different ZAID. These options are most useful when entered in column input form:

orig_zaid_n : subs_zaid_n
1001 8001001
11022 11023

This is usually necessary when nuclide metadata is not available in the SCALE standard composition library.

Default []**Type** list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)**subs_zaid_n**

Substitute ZAID corresponding to ‘orig_zaid_n’.

Default []**Type** list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)**orig_zaid_p**

Replace photon CE data for these nuclides in problem materials.

Default []**Type** list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)**subs_zaid_p**

Substitute ZAID corresponding to ‘orig_zaid_p’.

Default []

Type list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)

14.21.3 Advanced parameters

These parameters are not meant for typical use.

use_unionized_energy_grid

Create unionized energy grid.

Default False

Type boolean

load_gamma_production_data

Load the gamma production data.

Type boolean

load_nubar

Load average neutron production data.

Default True

Type boolean

reactions

Reactions to load (AMPX_MT values).

Default []

Type list of MT number or name (e.g. N_GAMMA, 102)

14.21.4 Commands

These commands generate one or more parameters.

energy_limits

Expand into parameters n_energy_min, n_energy_max.

ampx_kerma

Load KERMA factors from an AMPX library.

ce_lib

Set ce_lib to the CE library path.

The SCALE FileNamesAliases.txt file is used to resolve the data files. Current options are:

Path	Description
ce_v7.0_endf.xml	ENDF/B-VII.0
ce_v7.xml	“
ce_v7_endf.xml	“
ce_v7.1_endf.xml	ENDF/B-VII.1
ce.xml	“

If your build has the CMake variable SCALE_HPC_DATA_DIR defined, Shift will preferentially load HDF5 data from that.

14.21.5 Additional defaults

- Default mode to ‘n’ for kcode, or based on sources if present. .
- For particle mode np, ‘load_gamma_production_data’ defaults to True

For particle mode _default, ‘load_gamma_production_data’ defaults to False.

14.21.6 Additional restrictions

- Kcode problems must be run in mode ‘n’.
- The parameters orig_zaid_n, subs_zaid_n must have the same length..
- The parameters orig_zaid_p, subs_zaid_p must have the same length..

14.21.7 Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
ce_lib_path	db:ce	ce_lib_path
load_gamma_production_data	db:ce	gamma_production
load_nubar	db:ce	nubar
mode	db	mode
n_energy_max	db:ce	nemax
n_energy_min	db:ce	nemin
orig_zaid_n	db:ce	orig_zaid_n
orig_zaid_p	db:ce	orig_zaid_p
p_energy_max	db:ce	pemax
p_energy_min	db:ce	pemin
preserve_bands	db:ce	preserve_bands
probability_tables	db:ce	probability_tables
reactions	db:ce	reactions
subs_zaid_n	db:ce	subs_zaid_n
subs_zaid_p	db:ce	subs_zaid_p
thermal_energy_cutoff	db:ce	ethermal
use_unionized_energy_grid	db:ce	unionize_energy
xs_temp_correction_mode	db:ce	temp_correction

14.21.8 [PHYSICS][SPLICE=ampx]

The AMPX data splicing option allows multigroup data from an AMPX library to be spliced into the CE transport. This feature allows specialized reactions to be calculated during transport (e.g. KERMA tallies).

Currently, only multigroup reaction rates can be inserted. No probability tables or collision data can be inserted.

If a CE reaction is already present, the AMPX splicer will not override it. To get around this, the user can manually specify the list of reactions to load in the PHYSICS block, omitting the reactions to be loaded as multigroup data.

Parameters

mg_lib_path

Path to AMPX library to load data.

Type file path for reading

orig_zaid_n

Replace MG data for these nuclides in problem materials.

Default []

Type list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)

subs_zaid_n

Substitute ZAID corresponding to ‘orig_zaid_n’.

Default []

Type list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)

orig_zaid_p

Replace photon MG data for these nuclides in problem materials.

Default []

Type list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)

subs_zaid_p

Substitute ZAID corresponding to ‘orig_zaid_p’.

Default []

Type list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)

Advanced parameters

These parameters are not meant for typical use.

reactions

Multigroup reactions to splice into the CE data (AMPX_MT values).

Default []

Type list of MT number or name (e.g. N_GAMMA, 102)

Commands

These commands generate one or more parameters.

mg_lib

Set mg_lib to the given value using SCALE DATA resolution.

Additional restrictions

- The parameters orig_zaid_n, subs_zaid_n must have the same length..
- The parameters orig_zaid_p, subs_zaid_p must have the same length..

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
mg_lib_path		xs_library

14.22 [PHYSICS=void]

The special `void` physics type creates a one-group transport problem with void cross sections for all materials. It is used primarily in conjunction with the `raytrace` visualization mode.

14.22.1 Parameters

`name`

Label for the physics.

Default ‘void’

Type string

14.22.2 Advanced parameters

These parameters are not meant for typical use.

`disable_upscattering`

Lump upscattering into self-scattering.

Default True

Type boolean

`xsgen`

XS generation (always ‘void’).

Default ‘void’

Type string

`pn_order`

PN order for void (always zero).

Default 0

Type non-negative integer

`num_groups`

Number of energy groups.

Default 1

Type non-negative integer

`neutron_bnd`

Neutron group boundaries.

Default [20000000.0, 1e-05]

Type positive floats in decreasing order

14.22.3 Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
disable_upscattering	db	downscatter
neutron_bnd	db	neutron_bnd
num_groups	db	num_groups
pn_order	db	Pn_order

14.23 [DEPLETION]

Omnibus couples Shift and ORIGEN, a depletion/transmutation analysis code, using ORIGEN's new C++ API. This new in-memory API avoids the prohibitive cost (on HPC systems) of writing to disk between transport and depletion steps and also enables each depletion step to be performed on each depletable region in parallel. Shift uses the same approach as VESTA to obtain microscopic per-nuclide reaction rates. This approach tallies ultra-fine-group fluxes in each depletion region and then uses these fluxes to collapse the microscopic CE cross sections into one-group reaction rates. Shift then sends these reaction rates to ORIGEN for a depletion calculation.

Currently, Shift uses constant-power depletion with several coupling schemes to choose from. In order to optimize the parallel efficiency of the depletion calculation, the depletable regions on each block are distributed amongst the available processors in order to minimize the number of depletion solves performed on each core. After every core has calculated the new concentrations for its depletion regions, the results are broadcast to every other core on the block.

14.23.1 Sub-databases

Name and type	Frequency
<i>move</i>	Zero or more

14.23.2 Parameters

deplete_cells / cells

Labels of cells to deplete instead of all fissionable cells.

By default, when depletion is enabled, all “depletable” cells will be tracked and depleted. (Depletable cells correspond to materials with fissionable materials, or if a [COMP] block is used, materials with the depletable flag set.) This parameter overrides this default: cells with the listed labels will have their materials depleted.

Default []

Type list of integers

boron_level

Boron level for a given time interval.

If given, the boron_level parameter must have the same length as burn_time: each value corresponds to the boron level for a given time step.

Default []

Type list of positive floats

borated_cells / mod_cells

Cells containing borated moderator.

Default []

Type list of integers

group_bounds

Energy bin boundaries for depletion tally.

Default []

Type nonnegative floats in decreasing order

Units eV

tracking_nuclides

Nuclides that will be tracked for depletion.

Default []

Type list of nuclides

hdf5_output / depl_out

Name of the output file to write, or blank to disable output.

Default ‘depletion.h5’

Type file path to write (extension ‘.h5’) (empty value allowed)

coupling_method / method

Method used to solve the coupled depletion-transport problem..

The coupling_method parameter specifies which method to use for selecting the cross sections to use over each depletion step or predicting their behavior over the step (when not assumed constant). The same treatment is used for the spatial distribution of the neutron flux. The behavior of the methods is further controlled by the predictor_substeps and corrector_substeps parameters. The options (methods) and their descriptions are the following:

fully_explicit On the predictor, transport is solved at the beginning of step (BOS) and the BOS cross-sections are used for the entire step. No corrector.

middlestep Two-transport-solutions-per-step version of the middlestep method. Predictor uses BOS cross-sections and flux to get middle of step compositions. Corrector uses MOS cross- sections and flux to deplete for the entire step.

ce Same as fully_explicit (newer implementation).

le On the predictor, transport is solved at BOS and the cross sections are linearly interpolated through the previous step values and the BOS values. No corrector.

ce/li Predictor works as ce. On the corrector, transport is solved at the end of step (EOS) and the cross-sections are interpolated linearly through the BOS and EOS values.

le/li Predictor works as le. On the corrector, transport is solved at the end of step (EOS) and the cross-sections are interpolated linearly through the BOS and EOS values.

le/qi Predictor works as le. On the corrector, transport is solved at the end of step (EOS) and the cross-sections are interpolated quadratically through the previous step, BOS, and EOS values.

The labels ce, le, li, and qi come from constant extrapolation, linear extrapolation, linear interpolation, and quadratic interpolation, respectively. They stand for the approximation used for the time development of the cross-sections and flux on the predictor or corrector.

The methods le, le/li, and le/qi are more accurate than the rest, but also sensitive to large (roughly factor of 5 or more) changes in step lengths. These methods will default to lower order methods when previous step data is not available, or is not applicable due to changed normalization. For a more detailed description see the methodology manual (TODO).

Pure decay steps with zero flux will always be solved with ce regardless of the choice of coupling_method as it is the simplest method and still exact in the absence of neutron flux.

Default ‘fully_explicit’

Type ‘fully_explicit’, ‘middlestep’, ‘ce’, ‘le’, ‘ce/li’, ‘le/li’, or ‘le/qi’

write_predictor_data / write_p

Flag to write predictor data to an HDF5-formatted output file.

Default False

Type boolean

Applicability when writing HDF5 output; and ‘coupling_method’ is ‘ce/li’ or ‘le/li’ or ‘le/qi’ or ‘middlestep’.

write_xs

Whether to write the collapsed origin XS to the HDF5 file.

Default False

Type boolean

Applicability when writing HDF5 output.

write_filtering

Whether to save data about what nuclides were filtered.

Default False

Type boolean

Applicability when writing HDF5 output.

max_step

Maximum time before automatically increasing the number of steps.

Default 400.0

Type non-negative real number

Units days

burn_length / burn

Burnup lengths for each input step.

Type list of non-negative floats

Units days

decay_length / decay

Decay lengths for each input step.

Default []

Type list of non-negative floats

Units days

num_burn_steps

Number of steps to take for each burn length entry.

The num_burn_steps parameter must have the same length as burn_length, and decay_length. It is the number of constant-flux calculations per entry. Increasing the number will increase the accuracy of the answer (by having better approximations of the depleted concentrations during the transport step) but will increase the computational cost (because more transport calculations must be performed).

Default []

Type list of nonnegative integers

num_decay_steps

Number of steps to take for each decay length entry.

Default []

Type list of nonnegative integers

power

Constant power to be applied per burnup step.

Default []

Type list of non-negative floats

Units MW

constant_flux_per_step / flux

Constant flux to be applied per burnup step.

Default []

Type list of non-negative floats

Units n/cm²

origen_library

Filepath to an ORIGEN library file.

Default u'/usr/local/scale/data/origen_library/pwr.rev02.orglib'

Type library path

yield_library

Filepath to an ORIGEN fission yields library file.

Default u'/usr/local/scale/data/origen_data/origen.rev04.yields.data'

Type library path

jeff_library

Filepath to an ORIGEN JEFF multigroup file.

Default u'/usr/local/scale/data/origen.rev01.jeff252g'

Type library path

max_burn_substep_size

Maximum substep size for an ORIGEN time step of non-zero power/flux.

Default 40.0

Type non-negative real number

Units days

max_decay_substep_size

Maximum substep size for an ORIGEN time step of zero power/flux.

Default 75.0

Type non-negative real number

Units days

nuclide_filter_type

How to filter nuclides for transport calculations.

Default 'none'

Type ‘none’, ‘number_density’, ‘absorption’, or ‘total’

nuclide_filter_threshold

Threshold at which nuclides below are removed from transport.

Type non-negative real number

Applicability when filtering nuclides.

always_transport

Always run transport for each timestep, even during decay.

Default True

Type boolean

predictor_substeps

Number of substeps on the predictor.

The `predictor_substeps` parameter specifies the number of depletion substeps to use on the predictor. Substeps are always equidistant, except for long zero-flux steps with MATREX, where each substep will be thrice as long as the previous.

Substeps serve two roles:

1. The depletion for each substep uses cross-sections and flux averaged over that substeps from the prediction made in the coupling scheme.
2. The neutron flux is renormalized at each substep using the cross-sections and flux predicted for that sub-step.

In addition, the number of substeps affects the depletion solver accuracy. The MATREX depletion solver requires a sufficient number of substeps that are not too long. The number of substeps is automatically incremented on a step-by-step basis to meet these requirements. Decay steps with the CRAM solver always use one only substep. See the parameter `cram_internal_substeps`.

The default value of this parameter depends on the depletion solver and coupling scheme. If MATREX is used for depletion, the default is always 8. Else if `coupling_scheme` is `le`, the default is 5. Else if `coupling_scheme` is `le/li` or `le/qi` the default is 2. Else the default is 1.

Type positive integer

corrector_substeps

Number of substeps on the corrector.

The `corrector_substeps` parameter specifies the number of depletion substeps to use on the corrector (if applicable). See `predictor_substeps` for more details.

The default value of this parameter depends on the depletion solver and coupling scheme. If MATREX is used for depletion, the default is always 8. Else if `coupling_scheme` is `ce/li`, `le/li`, or `le/qi`, the default is 5. Else the default is 1.

Type positive integer

depletion_solver

Depletion solver type.

Default ‘cram’

Type ‘cram’, or ‘matrex’

cram_order

Order of the CRAM depletion solver.

Default 16

Type positive integer

Applicability when ‘depletion_solver’ is ‘cram’.

cram_internal_substeps

Number of internal substeps in the CRAM depletion solver. Only applied when there are fewer coupling substeps..

The `cram_internal_substeps` parameter specifies the minimum number of substeps that CRAM depletion solver should use on each step for depletion accuracy purposes. If this amount is larger than the number of regular substeps, the CRAM solver uses a feature called internal substeps to meet the amount specified here.

Internal substeps do not involve different cross sections or renormalization, but are much faster than regular substeps. The number of internal substeps to apply on each regular substep is selected so that the total number of internal substeps over all substeps adds up to at least `cram_internal_substeps`.

The default value is 2.

Default 2

Type positive integer

Applicability when ‘depletion_solver’ is ‘cram’.

14.23.3 Commands

These commands generate one or more parameters.

tracking_set

append a set of TRITON nuclides to tracking_nuclides to track (none, addnux1, addnux-2, addnux2, addnux3, addnux4, all).

The `tracking_set` command exposes the TRITON `addnux` option to the user.

`tracking_set none` adds no extra nuclides to track (the default).

`tracking_set addnux1` corresponds to `addnux=1` and adds:

U-234, U-235, U-236, U-238, Np-237, Pu-238, Pu-239, Pu-240, Pu-241, Pu-242, Am-241, Am-242, Am-243, Cm-242, Cm-243

`tracking_set addnux-2` corresponds to `addnux=-2` and adds the above nuclides as well as:

H-1, B-10, B-11, N-14, O-16, Kr-83, Zr-94, Nb-93, Mo-95, Tc-99, Ru-106, Rh-103, Rh-105, Ag-109, Sn-126, I-135, Xe-131, Xe-135, Cs-133, Cs-134, Cs-135, Cs-137, Ce-144, Pr-143, Nd-143, Nd-145, Nd-146, Nd-147, Nd-148, Pm-147, Pm-148, Pm-149, Sm-147, Sm-149, Sm-150, Sm-151, Sm-152, Eu-151, Eu-153, Eu-154, Eu-155, Gd-152, Gd-154, Gd-155, Gd-156, Gd-157, Gd-158, Gd-160, Cm-244

`tracking_set addnux2` corresponds to `addnux=2` and adds the above nuclides as well as:

Zr-91, Zr-93, Zr-95, Zr-96, Nb-95, Mo-97, Mo-98, Mo-99, Mo-100, Ru-101, Ru-102, Ru-103, Ru-104, Pd-105, Pd-107, Pd-108, Cd-113, In-115, I-127, I-129, Xe-133, Ba-140, La-139, Ce-141, Ce-142, Ce-143, Pr-141, Nd-144, Sm-153, Eu-156

`tracking_set addnux3` corresponds to `addnux=3` and adds the above nuclides as well as:

Ge-72, Ge-73, Ge-74, Ge-76, As-75, Se-77, Se-78, Se-80, Se-82, Br-79, Br-81, Kr-80, Kr-82, Kr-84, Kr-85, Kr-86, Rb-85, Rb-86, Rb-87, Sr-84, Sr-86, Sr-87, Sr-88, Sr-89, Sr-90, Y-89, Y-90, Y-91, Zr-90, Zr-92, Nb-94, Mo-92, Mo-94, Mo-96, Ru-96, Ru-98, Ru-99, Ru-100, Ru-105, Pd-102, Pd-104, Pd-106, Pd-110, Ag-107, Ag-111, Cd-106, Cd-108, Cd-110, Cd-111, Cd-112, Cd-114, Cd-115m, Cd-116, In-113, Sn-112, Sn-114, Sn-115, Sn-116, Sn-117, Sn-118, Sn-119, Sn-120, Sn-122, Sn-123, Sn-124, Sn-125, Sb-121, Sb-123, Sb-124, Sb-125, Sb-126, Te-120, Te-122, Te-123, Te-124,

Te-125, Te-126, Te-127m, Te-128, Te-129m, Te-130, Te-132, I-130, I-131, Xe-124, Xe-126, Xe-128, Xe-129, Xe-130, Xe-132, Xe-134, Xe-136, Cs-136, Ba-134, Ba-135, Ba-136, Ba-137, Ba-138, La-140, Ce-140, Pr-142, Nd-142, Nd-150, Pm-151, Sm-144, Sm-148, Sm-154, Eu-152, Eu-157, Tb-159, Tb-160, Dy-160, Dy-161, Dy-162, Dy-163, Dy-164, Ho-165, Er-166, Er-167, Lu-175, Lu-176, Ta-181, W-182, W-183, W-184, W-186, Re-185, Re-187, Au-197, Th-230, Th-232, Pa-231, Pa-233, U-232, U-233

`tracking_set addnux4` corresponds to `addnux=4` and adds the above nuclides as well as:

H-2, H-3, He-3, He-4, Li-6, Li-7, Be-7, Be-9, N-15, O-17, F-19, Na-23, Mg-24, Mg-25, Mg-26, Al-27, Si-28, Si-29, Si-30, P-31, S-32, S-33, S-34, S-36, Cl-35, Cl-37, Ar-36, Ar-38, Ar-40, Ka-39, Ka-40, Ka-41, Ca-40, Ca-42, Ca-43, Ca-44, Ca-46, Ca-48, Sc-45, Ti-46, Ti-47, Ti-48, Ti-49, Ti-50, Cr-50, Cr-52, Cr-53, Cr-54, Mn-55, Fe-54, Fe-56, Fe-57, Fe-58, Co-58m, Co-58, Co-59, Ni-58, Ni-59, Ni-60, Ni-61, Ni-62, Ni-64, Cu-63, Cu-65, Ga-69, Ga-71, Ge-70, As-74, Se-74, Se-79, Kr-78, Ag-110m, Sn-113, Xe-123, Ba-130, Ba-132, Ba-133, La-138, Ce-136, Ce-138, Ce-139, Pm-148m, Gd-153, Dy-156, Dy-158, Ho-166m, Er-162, Er-164, Er-168, Er-170, Hf-174, Hf-176, Hf-177, Hf-178, Hf-179, Hf-180, Ta-182, Ir-191, Ir-193, Hg-196, Hg-198, Hg-199, Hg-200, Hg-201, Hg-202, Hg-204, Pb-204, Pb-206, Pb-207, Pb-208, Bi-209, Ra-223, Ra-224, Ra-225, Ra-226, Ac-225, Ac-226, Ac-227, Th-227, Th-228, Th-229, Th-233, Th-234, Pa-232, U-237, U-239, U-240, U-241, Np-235, Np-236, Np-238, Np-239, Pu-236, Pu-237, Pu-243, Pu-244, Pu-246, Am-242m, Am-244, Am-244m, Cm-241, Cm-245, Cm-246, Cm-247, Cm-248, Cm-249, Cm-250, Bk-249, Bk-250, Cf-249, Cf-250, Cf-251, Cf-252, Cf-253, Cf-254, Es-253, Es-254, Es-255

`tracking_set all` contains all the nuclides available in ORIGEN, which are the above nuclides as well as:

H-4, He-5, He-6, He-8, Li-8, Li-9, Be-8, Be-10, Be-11, Be-12, B-12, C-0, C-12, N-13, N-16, O-18, O-19, F-20, Ne-20, Ne-21, Ne-22, Ne-23, Na-22, Na-24, Na-24m, Na-25, Mg-27, Mg-28, Al-26, Al-28, Al-29, Al-30, Si-31, Si-32, P-32, P-33, P-34, S-25, S-35, S-37, Cl-36, Cl-38, Cl-38m, Ar-37, Ar-39, Ar-41, Ar-42, Ka-42, Ka-43, Ka-44, Ca-41, Ca-45, Ca-47, Ca-49, Sc-44, Sc-44m, Sc-45m, Sc-46, Sc-46m, Sc-47, Sc-48, Sc-49, Sc-50, Ti-44, Ti-45, Ti-51, V-48, V-49, V-50, V-51, V-52, V-53, V-54, Cr-48, Cr-49, Cr-51, Cr-55, Cr-66, Cr-67, Mn-52, Mn-53, Mn-54, Mn-56, Mn-57, Mn-58, Mn-66, Mn-67, Mn-68, Mn-69, Fe-55, Fe-59, Fe-60, Fe-65, Fe-66, Fe-67, Fe-68, Fe-69, Fe-70, Fe-71, Fe-72, Co-55, Co-56, Co-57, Co-60, Co-60m, Co-61, Co-62, Co-65, Co-66, Co-67, Co-68, Co-69, Co-70, Co-71, Co-72, Co-73, Co-74, Co-75, Ni-56, Ni-57, Ni-63, Ni-65, Ni-66, Ni-67, Ni-68, Ni-69, Ni-70, Ni-71, Ni-72, Ni-73, Ni-74, Ni-75, Ni-76, Ni-77, Ni-78, Cu-62, Cu-64, Cu-66, Cu-67, Cu-68, Cu-68m, Cu-69, Cu-70, Cu-70m, Cu-71, Cu-72, Cu-73, Cu-74, Cu-75, Cu-76, Cu-77, Cu-78, Cu-79, Cu-80, Cu-81, Zn-63, Zn-64, Zn-65, Zn-66, Zn-67, Zn-68, Zn-69, Zn-69m, Zn-70, Zn-71, Zn-71m, Zn-72, Zn-73, Zn-74, Zn-75, Zn-76, Zn-77, Zn-78, Zn-79, Zn-80, Zn-81, Zn-82, Zn-83, Ga-66, Ga-67, Ga-68, Ga-70, Ga-72, Ga-72m, Ga-73, Ga-74, Ga-74m, Ga-75, Ga-76, Ga-77, Ga-78, Ga-79, Ga-80, Ga-81, Ga-82, Ga-83, Ga-84, Ga-85, Ga-86, Ge-66, Ge-67, Ge-68, Ge-69, Ge-71, Ge-71m, Ge-73m, Ge-75, Ge-75m, Ge-77, Ge-77m, Ge-78, Ge-79, Ge-79m, Ge-80, Ge-81m, Ge-81, Ge-82, Ge-83, Ge-84, Ge-85, Ge-86, Ge-87, Ge-88, Ge-89, As-69, As-71, As-72, As-73, As-75m, As-76, As-77, As-78, As-79, As-80, As-81, As-82, As-82m, As-83, As-84, As-84m, As-85, As-86, As-87, As-88, As-89, As-90, As-91, As-92, Se-72, Se-73, Se-73m, Se-75, Se-77m, Se-79m, Se-81, Se-81m, Se-83m, Se-83, Se-84, Se-85, Se-86, Se-87, Se-88, Se-89, Se-90, Se-91, Se-92, Se-93, Se-94, Br-75, Br-76, Br-77, Br-77m, Br-78, Br-79m, Br-80, Br-80m, Br-82, Br-82m, Br-83, Br-84, Br-84m, Br-85, Br-86, Br-87, Br-88, Br-89, Br-90, Br-91, Br-92, Br-93, Br-94, Br-95, Br-96, Br-97, Kr-76, Kr-77, Kr-79m, Kr-79, Kr-81, Kr-81m, Kr-83m, Kr-85m, Kr-87, Kr-88, Kr-89, Kr-90, Kr-91, Kr-92, Kr-93, Kr-94, Kr-95, Kr-96, Kr-97, Kr-98, Kr-99, Kr-100, Rb-79, Rb-81, Rb-82, Rb-83, Rb-84, Rb-86m, Rb-88, Rb-89, Rb-90, Rb-90m, Rb-91, Rb-92, Rb-93, Rb-94, Rb-95, Rb-96, Rb-97, Rb-98, Rb-99, Rb-100, Rb-101, Rb-102, Sr-82, Sr-83, Sr-85, Sr-85m, Sr-87m, Sr-91, Sr-92, Sr-93, Sr-94, Sr-95, Sr-96, Sr-97, Sr-98, Sr-99, Sr-100, Sr-101, Sr-102, Sr-103, Sr-104, Sr-105, Y-85, Y-86, Y-87, Y-87m, Y-88, Y-89m, Y-90m, Y-91m, Y-92, Y-93, Y-93m, Y-94, Y-95, Y-96, Y-96m, Y-97, Y-97m, Y-98, Y-98m, Y-99, Y-100, Y-101, Y-102, Y-103, Y-104, Y-105, Y-106, Y-107, Y-108, Zr-86, Zr-87, Zr-88, Zr-89, Zr-89m, Zr-90m, Zr-97, Zr-98, Zr-99, Zr-100, Zr-101, Zr-102, Zr-103, Zr-104, Zr-105, Zr-106, Zr-107, Zr-108, Zr-109, Zr-110, Nb-89, Nb-90, Nb-91, Nb-91m, Nb-92, Nb-92m, Nb-93m,

Nb-94m, Nb-95m, Nb-96, Nb-97, Nb-97m, Nb-98, Nb-98m, Nb-99, Nb-99m, Nb-100, Nb-100m, Nb-101, Nb-102, Nb-102m, Nb-103, Nb-104, Nb-104m, Nb-105, Nb-106, Nb-107, Nb-108, Nb-109, Nb-110, Nb-111, Nb-112, Nb-113, Mo-90, Mo-91, Mo-93, Mo-93m, Mo-101, Mo-102, Mo-103, Mo-104, Mo-105, Mo-106, Mo-107, Mo-108, Mo-109, Mo-110, Mo-111, Mo-112, Mo-113, Mo-114, Mo-115, Tc-93, Tc-95, Tc-95m, Tc-96, Tc-97, Tc-97m, Tc-98, Tc-99m, Tc-100, Tc-101, Tc-102, Tc-102m, Tc-103, Tc-104, Tc-105, Tc-106, Tc-107, Tc-108, Tc-109, Tc-110, Tc-111, Tc-112, Tc-113, Tc-114, Tc-115, Tc-116, Tc-117, Tc-118, Ru-95, Ru-97, Ru-107, Ru-108, Ru-109, Ru-109m, Ru-110, Ru-111, Ru-112, Ru-113, Ru-114, Ru-115, Ru-116, Ru-117, Ru-118, Ru-119, Ru-120, Rh-99, Rh-99m, Rh-100, Rh-101, Rh-101m, Rh-102, Rh-102m, Rh-103m, Rh-104, Rh-104m, Rh-105m, Rh-106, Rh-106m, Rh-107, Rh-108, Rh-108m, Rh-109, Rh-109m, Rh-110, Rh-110m, Rh-111, Rh-112, Rh-113, Rh-114, Rh-115, Rh-116, Rh-117, Rh-118, Rh-119, Rh-120, Rh-121, Rh-122, Rh-123, Pd-99, Pd-100, Pd-101, Pd-103, Pd-107m, Pd-109, Pd-109m, Pd-111, Pd-111m, Pd-112, Pd-113, Pd-114, Pd-115, Pd-116, Pd-117, Pd-118, Pd-119, Pd-120, Pd-121, Pd-122, Pd-123, Pd-124, Pd-125, Pd-126, Ag-103, Ag-105, Ag-105m, Ag-106, Ag-106m, Ag-107m, Ag-108, Ag-108m, Ag-109m, Ag-110, Ag-111m, Ag-112, Ag-113, Ag-113m, Ag-114, Ag-115, Ag-115m, Ag-116, Ag-116m, Ag-117, Ag-117m, Ag-118, Ag-118m, Ag-119, Ag-120, Ag-120m, Ag-121, Ag-122, Ag-122m, Ag-123, Ag-124, Ag-125, Ag-126, Ag-127, Ag-128, Ag-129, Ag-130, Cd-105, Cd-107, Cd-109, Cd-111m, Cd-113m, Cd-115, Cd-117, Cd-117m, Cd-118, Cd-119, Cd-119m, Cd-120, Cd-121, Cd-121m, Cd-122, Cd-123, Cd-123m, Cd-124, Cd-125, Cd-126, Cd-127, Cd-128, Cd-129, Cd-130, Cd-131, Cd-132, In-107, In-109, In-111m, In-111, In-112, In-112m, In-113m, In-114m, In-114, In-115m, In-116m, In-116, In-117m, In-117, In-118m, In-118, In-119m, In-119, In-120m, In-120, In-121m, In-121, In-122m, In-122, In-123m, In-123, In-124m, In-124, In-125m, In-125, In-126m, In-126, In-127m, In-127, In-128m, In-128, In-129m, In-129, In-130m, In-130, In-131m, In-131, In-132, In-133, In-134, In-135, Sn-111, Sn-113m, Sn-117m, Sn-119m, Sn-121, Sn-121m, Sn-123m, Sn-125m, Sn-127, Sn-127m, Sn-128, Sn-128m, Sn-129, Sn-129m, Sn-130, Sn-130m, Sn-131, Sn-131m, Sn-132, Sn-133, Sn-134, Sn-135, Sn-136, Sn-137, Sb-113, Sb-115m, Sb-115, Sb-117, Sb-118m, Sb-118, Sb-119, Sb-120m, Sb-120, Sb-122m, Sb-122, Sb-124m, Sb-126m, Sb-127, Sb-128m, Sb-128, Sb-129, Sb-130m, Sb-130, Sb-131, Sb-132m, Sb-132, Sb-133, Sb-134m, Sb-134, Sb-135, Sb-136, Sb-137, Sb-138, Sb-139, Te-115, Te-117, Te-118, Te-119m, Te-119, Te-121, Te-121m, Te-123m, Te-125m, Te-127, Te-129, Te-131, Te-131m, Te-133, Te-134, Te-135, Te-136, Te-137, Te-138, Te-139, Te-140, Te-141, Te-142, I-121, I-122, I-123, I-124, I-125, I-126, I-128, I-130m, I-132, I-132m, I-133, I-133m, I-134m, I-134, I-136m, I-136, I-137, I-138, I-139, I-140, I-141, I-142, I-143, I-144, I-145, Xe-122, Xe-125, Xe-125m, Xe-127m, Xe-127, Xe-129m, Xe-131m, Xe-133m, Xe-134m, Xe-135m, Xe-137, Xe-138, Xe-139, Xe-140, Xe-141, Xe-142, Xe-143, Xe-144, Xe-145, Xe-145m, Xe-146, Xe-147, Cs-127, Cs-128, Cs-129, Cs-130, Cs-131, Cs-132, Cs-134m, Cs-135m, Cs-136m, Cs-138m, Cs-138, Cs-139, Cs-140, Cs-141, Cs-142, Cs-143, Cs-144, Cs-145, Cs-146, Cs-147, Cs-148, Cs-149, Cs-150, Cs-151, Ba-128, Ba-129, Ba-131, Ba-131m, Ba-133m, Ba-135m, Ba-136m, Ba-137m, Ba-139, Ba-141, Ba-142, Ba-143, Ba-144, Ba-145, Ba-146, Ba-147, Ba-148, Ba-149, Ba-150, Ba-151, Ba-152, Ba-153, La-133, La-134, La-135, La-136, La-137, La-141, La-142, La-143, La-144, La-145, La-146m, La-146, La-147, La-148, La-149, La-150, La-151, La-152, La-153, La-154, La-155, Ce-134, Ce-135, Ce-137m, Ce-137, Ce-139m, Ce-145, Ce-146, Ce-147, Ce-148, Ce-149, Ce-150, Ce-151, Ce-152, Ce-153, Ce-154, Ce-155, Ce-156, Ce-157, Pr-139, Pr-140, Pr-142m, Pr-144m, Pr-144, Pr-145, Pr-145, Pr-146, Pr-146, Pr-147, Pr-147, Pr-148m, Pr-148, Pr-149, Pr-149, Pr-150, Pr-151, Pr-152, Pr-153, Pr-154, Pr-155, Pr-155, Pr-156, Pr-157, Pr-158, Pr-159, Nd-140, Nd-141, Nd-149, Nd-151, Nd-152, Nd-153, Nd-154, Nd-155, Nd-156, Nd-157, Nd-158, Nd-159, Nd-160, Nd-161, Pm-141, Pm-143, Pm-144, Pm-145, Pm-146, Pm-150, Pm-152m, Pm-152, Pm-153, Pm-154m, Pm-154, Pm-155, Pm-156, Pm-157, Pm-158, Pm-159, Pm-160, Pm-161, Pm-162, Pm-163, Sm-143, Sm-143m, Sm-145, Sm-146, Sm-155, Sm-156, Sm-157, Sm-158, Sm-159, Sm-160, Sm-161, Sm-162, Sm-163, Sm-164, Sm-165, Eu-145, Eu-146, Eu-147, Eu-148, Eu-149, Eu-150m, Eu-150, Eu-152m, Eu-154m, Eu-158, Eu-159, Eu-160, Eu-161, Eu-162, Eu-163, Eu-164, Eu-165, Eu-166, Eu-167, Gd-146, Gd-147, Gd-148, Gd-149, Gd-150, Gd-151, Gd-153m, Gd-155m, Gd-159, Gd-161, Gd-162, Gd-163, Gd-164, Gd-165, Gd-166, Gd-167, Gd-168, Gd-169, Tb-151, Tb-152, Tb-153, Tb-154m, Tb-154, Tb-155, Tb-156m, Tb-156, Tb-157, Tb-158m, Tb-158, Tb-161, Tb-162, Tb-163, Tb-164, Tb-165, Tb-166, Tb-167, Tb-168, Tb-169, Tb-170, Tb-171, Dy-155, Dy-157, Dy-159, Dy-165, Dy-

165m, Dy-166, Dy-167, Dy-168, Dy-169, Dy-170, Dy-171, Dy-172, Ho-159m, Ho-159, Ho-160m, Ho-160, Ho-161m, Ho-161, Ho-162m, Ho-162, Ho-163m, Ho-163, Ho-164m, Ho-164, Ho-166, Ho-167, Ho-168, Ho-169, Ho-170m, Ho-170, Ho-171, Ho-172, Er-160, Er-161, Er-163, Er-165, Er-167m, Er-169, Er-171, Er-172, Tm-165, Tm-166, Tm-167, Tm-168, Tm-169, Tm-170m, Tm-170, Tm-171, Tm-172, Tm-173, Yb-166, Yb-167, Yb-168, Yb-169m, Yb-169, Yb-170, Yb-171, Yb-172, Yb-173, Yb-174, Yb-175m, Yb-175, Yb-176, Yb-177, Lu-169m, Lu-169, Lu-170, Lu-171m, Lu-171, Lu-172m, Lu-172, Lu-173, Lu-174m, Lu-174, Lu-176m, Lu-177m, Lu-177, Hf-170, Hf-171, Hf-172, Hf-173, Hf-175, Hf-178m, Hf-179m, Hf-180m, Hf-181, Hf-182, Ta-177, Ta-178, Ta-179, Ta-180m, Ta-180, Ta-182m, Ta-183, W-178, W-180, W-181, W-183m, W-185, W-185m, W-187, W-188, W-189, Re-181, Re-182, Re-182m, Re-183, Re-184, Re-184m, Re-186, Re-186m, Re-188, Re-188m, Re-189, Os-182, Os-183, Os-184, Os-185, Os-186, Os-187, Os-188, Os-189, Os-189m, Os-190, Os-190m, Os-191, Os-191m, Os-192, Os-193, Os-194, Ir-185, Ir-186, Ir-188, Ir-189, Ir-189m, Ir-190, Ir-191m, Ir-192, Ir-192m, Ir-193m, Ir-194, Ir-194m, Ir-196, Ir-196m, Pt-188, Pt-189, Pt-190, Pt-191, Pt-192, Pt-193, Pt-193m, Pt-194, Pt-195, Pt-195m, Pt-196, Pt-197, Pt-197m, Pt-198, Pt-199, Pt-199m, Pt-200, Au-193, Au-194, Au-195, Au-195m, Au-196, Au-198, Au-198m, Au-199, Au-199m, Au-200, Au-200m, Hg-193m, Hg-193, Hg-194, Hg-195m, Hg-195, Hg-197, Hg-197m, Hg-199m, Hg-203, Hg-205, Hg-206, Tl-200, Tl-201, Tl-202, Tl-203, Tl-204, Tl-205, Tl-206, Tl-207, Tl-208, Tl-209, Tl-210, Pb-200, Pb-202, Pb-203, Pb-205, Pb-205m, Pb-207m, Pb-209, Pb-210, Pb-211, Pb-212, Pb-214, Bi-205, Bi-206, Bi-207, Bi-208, Bi-210, Bi-210m, Bi-211, Bi-212, Bi-212m, Bi-213, Bi-214, Po-206, Po-207, Po-208, Po-209, Po-210, Po-211, Po-211m, Po-212, Po-213, Po-214, Po-215, Po-216, Po-218, At-216, At-217, At-218, Rn-216, Rn-217, Rn-218, Rn-219, Rn-220, Rn-222, Fr-220, Fr-221, Fr-222, Fr-223, Ra-220, Ra-222, Ra-227, Ra-228, Ac-224, Ac-228, Th-226, Th-231, Pa-228, Pa-229, Pa-230, Pa-234, Pa-234m, Pa-235, U-230, U-231, Np-234, Np-236m, Np-240, Np-240m, Np-241, Pu-237m, Pu-245, Pu-247, Am-239, Am-240, Am-245, Am-246, Am-247, Cm-240, Cm-251, Bk-245, Bk-246, Bk-247, Bk-248, Bk-248m, Bk-251, Cf-246, Cf-248, Cf-255, Es-251, Es-252, Es-254m

These tables are available in the SCALE 6.1 manual.

14.23.4 Additional defaults

- Appropriate number of substeps on predictor based on the coupling scheme and depletion solver.
- Appropriate number of substeps on corrector based on the coupling scheme and depletion solver.

14.23.5 Additional restrictions

- Nuclides in ‘tracking_nuclides’ are validated against the TRITON nuclide list..
- The parameters `burn_length`, `decay_length`, `power`, `num_burn_steps`, `num_decay_steps` must have the same length. Empty lists are ignored..

14.23.6 Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
always_transport	db	always_transport
burn_length	db	burn_lengths
constant_flux_per_step	db	fluxes
corrector_substeps	db	corrector_substeps
coupling_method	db	depletion_method
cram_internal_substeps	db	cramInternalSubsteps
cram_order	db	cramOrder
decay_length	db	decay_lengths
depletion_solver	db	origen_solver
group_bounds	db	group_bounds
hdf5_output	db	hdf5_filename
jeff_library	db	jeff_library
max_burn_substep_size	db	max_burn_step_size
max_decay_substep_size	db	max_decay_step_size
max_step	db	max_burn_length
nuclide_filter_threshold	db	filter_threshold
nuclide_filter_type	db	nuclide_filter
num_burn_steps	db	num_burn_steps
num_decay_steps	db	num_decay_steps
origen_library	db	library
power	db	powers
predictor_substeps	db	predictor_substeps
tracking_nuclides	db	tracking_nuclides
write_predictor_data	db	write_predictor
write_xs	db	write_xs
yield_library	db	yield_library

14.23.7 [DEPLETION][MOVE]

Parameters

name

Name of the surface group.

Type string

delta

Distance to move surfaces at the beginning of each step.

Type list of floats

Additional restrictions

- Number of movement steps should match depletion steps..
- Move name must correspond to a movable geometry name.

14.24 [SHIFT]

The Shift database is for Monte Carlo execution parameters. It is broken into a top-level database and three subsidiary databases. If a KCODE problem is being run (see [\[PROBLEM\]](#)), the [KCODE] subdatabase is required. It specifies

the number of inactive and active cycles, as well as the particles per cycle. If running a fixed-source problem, the `num_histories` parameter in the main [SHIFT] database is required.

Shift supports experimental domain decomposition for very large problems. If desired, the *[DECOMPOSITION]* block should be included to specify the spatial decomposition.

14.24.1 Sub-databases

Name and type	Frequency
<code>decomposition</code>	Optional
<code>kcode</code>	Optional
<code>vr</code>	Exactly once

14.24.2 Parameters

`do_transport`

If false, this disables the actual solve.

Setting `do_transport` to `false` is a way to ensure problem integrity without running an expensive transport calculation. It disables transport itself but allows the rest of the code (including building sources, tallies, and physics) to run. This is similar to `parm=check` in SCALE.

Changed in version 5.3: This replaces the `mode check` problem option.

Default True

Type boolean

`num_histories / np`

Number of histories.

Default 1000

Type positive integer

Applicability when problem mode is ‘forward’ or ‘adjoint’.

`num_dd_samples`

Number of test samples to determine initial particle balance.

With domain decomposed problems, an *a priori* determination of the particle balance across blocks is performed before transport. Each domain randomly samples the source for a total of `num_dd_samples` times; the fraction of source particles found within the “core” (non-overlapping) component of each domain is used to determine the number of particles emitted from the source on that domain.

Type integer

Applicability when Shift spatial partitioning is domain decomposed.

`max_lost_count`

Maximum number of particles that can be lost on a single domain.

Default 10

Type positive integer

output_fraction_completed

Output fraction of completed particles.

Type positive real number

check_frequency / chk_freq

Check frequency for domain decomposed current cycle completion.

Default 1

Type positive integer

Applicability when Shift spatial partitioning is domain decomposed.

particle_buffer_size / bufsize

Size of particle buffer for DD problem.

Default 1000

Type positive integer

Applicability when Shift spatial partitioning is domain decomposed.

14.24.3 Advanced parameters

These parameters are not meant for typical use.

physics

Name of the physics DB to use.

Type string

14.24.4 Additional defaults

- ‘physics’ defaults to the name of the last PHYSICS entered..
- Domain decomposition samples (num_dd_samples) defaults to num_histories / 10.
- Output fraction depends on number of cores and method..
- A [VR] subdatabase is added by default.

14.24.5 Additional restrictions

- A [KCODE] sub-database must be included for eigenvalue problems.

14.24.6 Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
check_frequency		mc_check_freq
num_dd_samples	source_db	dd_test_samples
num_histories	source_db	Np
output_fraction_completed		mc_diag_frac

14.24.7 [SHIFT][DECOMPOSITION]

This database determines how the Shift Monte Carlo problem is spatially decompose. If not present, the problem's spatial extents will be used for the default partition.

Parameters

`x_partition / x`

Boundary mesh along the X axis.

Type list of two or more monotonically increasing floats

`y_partition / y`

Boundary mesh along the Y axis.

Type list of two or more monotonically increasing floats

`z_partition / z`

Boundary mesh along the Z axis.

Type list of two or more monotonically increasing floats

`overlap`

Fraction of DD domain overlap.

Default 0.0

Type real number inclusive [0.0, 1.0]

Applicability when Shift spatial partitioning is domain decomposed.

Advanced parameters

These parameters are not meant for typical use.

`boundary_reflect`

Whether to reflect on each side of the boundary mesh.

Default [0, 0, 0, 0, 0, 0]

Type list of six zero/one integers for -X,+X,-Y,+Y,-Z,+Z

Additional restrictions

- The number of processors must divide evenly into the number of decompositions if a [RUN] block is used.

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
<code>boundary_reflect</code>	<code>boundary_db</code>	<code>reflect_bnd_mesh</code>
<code>overlap</code>	<code>boundary_db</code>	<code>overlap</code>
<code>x_partition</code>	<code>boundary_db</code>	<code>x_bnd_mesh</code>
<code>y_partition</code>	<code>boundary_db</code>	<code>y_bnd_mesh</code>
<code>z_partition</code>	<code>boundary_db</code>	<code>z_bnd_mesh</code>

14.24.8 [SHIFT][KCODE]

The default entropy grid uses the Shift decomposition (which by default is the problem's extents, if using RTK or SCALE geometry) as its outer extents. If it's not specified, it tries to build a mesh with

$$N_{\text{cells}} = \max(N_p/10, 64),$$

where N_p is the number of particles per cycle.

Sub-databases

Name and type	Frequency
acceleration = { kde }	Optional

Parameters

`num_histories_per_cycle / npk`

Number of histories per cycle.

Default 1000

Type positive integer

`x_entropy`

Boundaries in x for the Shannon entropy mesh.

Type list of two or more monotonically increasing floats

Units cm

`y_entropy`

Boundaries in y for the Shannon entropy mesh.

Type list of two or more monotonically increasing floats

Units cm

`z_entropy`

Boundaries in z for the Shannon entropy mesh.

Type list of two or more monotonically increasing floats

Units cm

`initial_keff / initk`

Initial keff value.

Default 1.0

Type positive real number

`num_cycles / nk`

Number of kcode cycles.

Default 50

Type integer

`num_inactive_cycles / nik`

Number of inactive kcode cycles.

Default 10

Type integer

Additional defaults

- Shannon entropy defaults to partition bounds.

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
initial_keff	solver_db	keff_init
num_cycles	solver_db	num_cycles
num_histories_per_cycle	source_db	Np
num_inactive_cycles	solver_db	num_inactive_cycles
x_entropy	solver_db	entropy_x
y_entropy	solver_db	entropy_y
z_entropy	solver_db	entropy_z

[SHIFT][KCODE][ACCELERATION=kde]

Uses a kernel density estimator to sample from the fission sites, thereby providing accelerated fission source convergence.

Parameters

rejection_method / method

KDE kernel rejection method.

Type ‘position_resample’, ‘position_fission_site’, or ‘bandwidth_rejection’

14.24.9 [SHIFT][VR]

Parameters

method / vr

Variance reduction method.

Default ‘roulette’

Type ‘ww’, ‘analog’, or ‘roulette’

weight_cutoff / wc

Particle weight cutoff for roulette.

Default 0.25

Type non-negative real number

Applicability when ‘method’ is ‘roulette’ or ‘ww’.

weight_survival / ws

Particle weight survival for roulette.

Default 0.5

Type non-negative real number

Applicability when ‘method’ is ‘roulette’.

ww_lower_factor / wwlw

Lower weight window ratio.

Default 0.5

Type real number inside (0, 1)

Applicability when ‘method’ is ‘ww’.

ww_upper_factor / wwhigh

Upper weight window ratio.

Default 2.5

Type real number greater than 1

Applicability when ‘method’ is ‘ww’.

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
method	vr_db	method
weight_cutoff	vr_db	weight_cutoff
weight_survival	vr_db	weight_survival
ww_lower_factor	vr_db	ww_lower_factor
ww_upper_factor	vr_db	ww_upper_factor

14.25 [DENOVO]

14.25.1 Sub-databases

Name and type	Frequency
<i>decomposition</i>	Exactly once
<i>solver</i>	Exactly once
<i>quadrature</i>	Exactly once
<i>raytrace</i>	Exactly once

14.25.2 Parameters

do_transport

If false, this disables the actual solve.

Default True

Type boolean

method

Solution method or spatial discretization.

Default ‘sc’

Type ‘spn’, ‘sc’, ‘sc_2d’, ‘bld_2d’, ‘ld’, or ‘tld’

x

Mesh coordinates along the X axis.

Type list of monotonically increasing floats

Applicability when geometry type is ‘rtk’ or ‘mcnp’ or ‘scale’ or ‘sword’.

y

Mesh coordinates along the Y axis.

Type list of monotonically increasing floats

Applicability when geometry type is ‘rtk’ or ‘mcnp’ or ‘scale’ or ‘sword’.

z

Mesh coordinates along the Z axis.

Type list of monotonically increasing floats

Applicability when geometry type is ‘rtk’ or ‘mcnp’ or ‘scale’ or ‘sword’.

boundary

Boundary conditions on the deterministic problem.

Default ‘vacuum’

Type ‘vacuum’, ‘isotropic’, or ‘reflect’

boundary_reflect

Whether to reflect on each side of the problem.

Default [1, 1, 1, 1, 1, 1]

Type list of six zero/one integers for -X,+X,-Y,+Y,-Z,+Z

Applicability when ‘boundary’ is ‘reflect’.

physics

Name of the associated physics database.

Type string

first_group

The first energy group to solve.

Default 0

Type non-negative integer

last_group

The last energy group to solve.

Default 1000000

Type non-negative integer

mc_num_particles

The number of particles to use with Monte Carlo first-collision source.

Type positive integer

Applicability when source denovo discretization type is ‘uncollided_mc’.

mc_invariant

Enable reproducability of Monte Carlo first-collision source.

Default False

Type boolean

Applicability when source denovo discretization type is ‘uncollided_mc’.

mc_num_samples

The number of samples per cell volume to discretize the Shift source onto the mesh.

Default 1000

Type positive integer

Applicability when source denovo discretization type is ‘uncollided_mc’.

do_silo

Do silo output.

Default True

Type boolean

silo_mixing_table

Write mixed materials to Silo output.

Default True

Type boolean

Applicability when ‘do_silo’ is ‘True’.

silo_output

Name of Silo output file (no extension).

Default ‘denovo_output’

Type string

Applicability when ‘do_silo’ is ‘True’.

hdf5_output

Write HDF5 output file.

Default True

Type boolean

hdf5_output_file

Name of HDF5 output file.

Default ‘denovo_output.h5’

Type string

Applicability when ‘hdf5_output’ is ‘True’.

14.25.3 Advanced parameters

These parameters are not meant for typical use.

seed

Fixed RNG seed for ray tracing and MC uncollided flux.

Default 123456

Type non-negative integer

dimension

Spatial dimension of the problem.

Default 3

Type integer 2 or 3

mesh_hdf5_filename

Name of mesh HDF5 file input.

Type file path for reading (extension ‘.h5’)

Applicability when geometry type is ‘mesh’; and mesh geometry input type is ‘hdf5’.

disable_upscattering

Whether to disable upscattering.

Type boolean

pn_order

Order of the Legendre scattering expansion.

Type non-negative integer

use_cuda

Use the CUDA sweeper to solve the KBA equations (experimental).

Default False

Type boolean

Applicability when ‘method’ is ‘sc’ or ‘ld’.

14.25.4 Additional defaults

- ‘physics’ defaults to the name of the last PHYSICS entered..
- Auto-set multigroup physics parameters.
- Auto-set geometry parameters.
- Auto-set reflecting boundary.
- A [DECOMPOSITION] subdatabase is added by default.
- A [SOLVER] subdatabase is added by default.
- A [QUADRATURE] subdatabase is added by default.
- A [RAYTRACE] subdatabase is added by default.

14.25.5 Additional restrictions

- Complex validation of parameters.

14.25.6 Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
boundary_reflect	boundary_db	reflect
disable_upscattering		downscatter
mc_num_particles		np
pn_order		Pn_order
silo_mixing_table	silo_db	mixing_table
silo_output	silo_db	silo_output
x		x_edges
y		y_edges
z		z_edges

14.25.7 [DENOVO][DECOMPOSITION]

Parameters

x_blocks

The number of spatial partitions along the x axis.

Type non-negative integer

y_blocks

The number of spatial partitions along the y axis.

Type non-negative integer

z_blocks

The number of pipelining blocks along the z axis.

Type non-negative integer

energy_sets

The number of energy partitions.

Default 1

Type non-negative integer

partition_upscatter

Partition just the upscatter groups.

Type boolean

Additional defaults

- If a [RUN] block is present, come up with a logically square KBA decomposition.
- Force number of Z blocks in SPN to 1, and default to nx * ny for SN.

- Set `upscatter_partitioning` defaults based on energy set partitioning. `upscatter_partitioning = True` if `energy_sets = 1`, otherwise `upscatter_partitioning = False`.

Additional restrictions

- The number of processors must be equal to the decomposition size ($X * Y * E$) if a [RUN] block is used..

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
<code>energy_sets</code>		<code>num_sets</code>
<code>partition_upscatter</code>		<code>partition_upscatter</code>
<code>x_blocks</code>		<code>num_blocks_i</code>
<code>y_blocks</code>		<code>num_blocks_j</code>
<code>z_blocks</code>		<code>num_z_blocks</code>

14.25.8 [DENOVO][SOLVER]

Parameters

tolerance

Convergence criterion for the deterministic solver.

Default 0.001

Type real number inside (0, 1)

max_iterations

The maximum number of Krylov or source iterations.

Default 100

Type non-negative integer

solver

Solver used in each within-group SN solve.

Default ‘gmres’

Type ‘gmres’, ‘si’, or ‘gmres_r’

krylov_space

The number of Krylov vectors to store when using GMRES.

Default 20

Type integer

Applicability when ‘solver’ is ‘gmres_r’ or ‘gmres’.

multigroup_solver

Solution technique for the multigroup block solve.

Default ‘gauss_seidel’

Type ‘gauss_seidel’, or ‘krylov’

Applicability when upscattering is enabled.

two_grid

Enable two-grid upscattering acceleration.

Default False

Type boolean

Applicability when upscattering is enabled.

upscatter_solver

Solver for upscattering groups.

Type ‘gmres’, ‘si’, or ‘gmres_r’

Applicability when upscattering is enabled.

upscatter_tolerance

Convergence criterion for the upscatter iterations.

Default 0.01

Type real number inside (0, 1)

upscatter_inner_iterations

Maximum number of iterations for the within-group upscatter solve.

Default 10

Type non-negative integer

Applicability when ‘multigroup_solver’ is ‘gauss_seidel’.

upscatter_inner_tolerance

Convergence criterion for the within-group upscattering solve.

Default 0.01

Type real number inside (0, 1)

Applicability when ‘multigroup_solver’ is ‘gauss_seidel’.

eigenvalue_solver

Solver used for eigenvalue iteration.

Default ‘power_iteration’

Type ‘power_iteration’, or ‘arnoldi’

Applicability when problem mode is ‘kcode’.

k_tolerance

Convergence tolerance for keff.

Default 0.0001

Type real number inside (0, 1)

Applicability when problem mode is ‘kcode’.

max_k_tolerance

Maximum solver tolerance for keff.

Default 0.01

Type real number inside (0, 1)

Applicability when problem mode is ‘kcode’; and ‘eigenvalue_solver’ is ‘power_iteration’.

l2norm_tolerance

Tolerance for L2 norm of power iteration.

Default 1.0

Type real number inclusive [0.0, 1.0]

Applicability when problem mode is ‘kcode’; and ‘eigenvalue_solver’ is ‘power_iteration’.

acceleration_method

Type of power iteration acceleration.

Default ‘none’

Type ‘none’, ‘cmfd’, or ‘rebalance’

Applicability when problem mode is ‘kcode’; and ‘eigenvalue_solver’ is ‘power_iteration’.

diagnostic_output

Level of keff diagnostics to output during solve.

Default 0

Type integer

Applicability when problem mode is ‘kcode’; and ‘eigenvalue_solver’ is ‘power_iteration’.

max_arnoldi_restarts

Maximum number of Arnoldi restarts.

Default 100

Type integer

Applicability when problem mode is ‘kcode’; and ‘eigenvalue_solver’ is ‘arnoldi’.

arnoldi_kspace

Size of the Arnoldi solve space.

Default 25

Type integer

Applicability when problem mode is ‘kcode’; and ‘eigenvalue_solver’ is ‘arnoldi’.

calculate_flux_moments

Calculate the flux moments during Arnoldi iteration.

Default False

Type boolean

Applicability when problem mode is ‘kcode’; and ‘eigenvalue_solver’ is ‘arnoldi’.

use_energy_dependent

Use energy-dependent Arnoldi iteration.

Default False

Type boolean

Applicability when problem mode is ‘kcode’; and ‘eigenvalue_solver’ is ‘arnoldi’.

Additional defaults

- Default the upscattering solver to the within-group solver.

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
acceleration_method	eigenvalue_db	acceleration
arnoldi_kspace	eigenvalue_db	arnoldi_kspace
calculate_flux_moments	eigenvalue_db	calculate_moments
diagnostic_output	eigenvalue_db	diagnostic_level
eigenvalue_solver		eigen_solver
k_tolerance	eigenvalue_db	k_tolerance
krylov_space		aztec_kspace
l2norm_tolerance	eigenvalue_db	L2_tolerance
max_arnoldi_restarts	eigenvalue_db	arnoldi_restarts
max_iterations		max_itr
max_k_tolerance	eigenvalue_db	max_tolerance
multigroup_solver		mg_solver
solver		within_group_solver
two_grid	upscatter_db	upscatter_acceleration
upscatter_inner_iterations	upscatter_db	inner_itr
upscatter_inner_tolerance	upscatter_db	inner_tolerance
upscatter_solver	upscatter_db	up_group_solver
upscatter_tolerance	upscatter_db	tolerance
use_energy_dependent	eigenvalue_db	energy_dep_ev

14.25.9 [DENOVO][QUADRATURE]

Parameters

quadrature

Discrete ordinates quadrature set class.

Default ‘qr’

Type ‘levelsym’, ‘glproduct’, ‘qr’, ‘ldfe’, or ‘userdefined’

Applicability when denovo discretization method is ‘sc’ or ‘sc_2d’ or ‘bld_2d’ or ‘ld’ or ‘tld’.

quad_order

Level-symmetric quadrature set order.

Default 10

Type non-negative integer

Applicability when ‘quadrature’ is ‘qr’ or ‘levelsym’.

quad_num_azi

Number of azimuthal angles per level per octant.

Default 4

Type positive integer

Applicability when ‘quadrature’ is ‘ldfe’ or ‘qr’ or ‘glproduct’.

quad_num_azi_vec

List of the number of azimuthal angles per polar angle per octant, ordered from pole to equator.

Default []

Type list of positive integers

Applicability when ‘quadrature’ is ‘qr’.

quad_num_polar

Number of polar angles per level per octant.

Default 4

Type positive integer

Applicability when ‘quadrature’ is ‘ldfe’ or ‘qr’ or ‘glproduct’.

quad_file

User-specified quadrature set file.

Type file path for reading

Applicability when ‘quadrature’ is ‘userdefined’.

ldfe_order

Order for the LDFE quadrature set.

Default 1

Type positive integer

Applicability when ‘quadrature’ is ‘ldfe’.

quad_polar_axis

Axis of rotation for product quadrature sets.

Default ‘z’

Type ‘x’, ‘y’, or ‘z’

Applicability when ‘quadrature’ is ‘qr’ or ‘glproduct’.

do_adjoint

Whether to solve the adjoint.

Default False

Type boolean

Additional defaults

- Set the quadrature to adjoint if running an adjoint problem.

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
do_adjoint	quadrature_db	adjoint
ldfe_order	quadrature_db	order
quad_file	quadrature_db	quadrature_file_name
quad_num_azi	quadrature_db	azimuthals_octant
quad_num_azi_vec	quadrature_db	azimuthals_vector
quad_num_polar	quadrature_db	polars_octant
quad_order	quadrature_db	Sn_order
quad_polar_axis	quadrature_db	polar_axis
quadrature	quadrature_db	quad_type

14.25.10 [DENOVOLRAYTRACE]

Parameters

mix_tolerance

Tolerance for collapsing similar mixed materials into one.

Default 0.05

Type real number inside (0, 1)

rays_deterministic

If true, use face midpoints rather than stratified sampling.

Default False

Type boolean

rays_per_face

Number of ray trace rays to be fired per mesh face.

Default 4

Type positive square integer

axes

Axis/axes along which to fire rays for ray trace.

Default ‘xyz’

Type axis or axes (‘x’, ‘zy’, ‘xyz’)

Advanced parameters

These parameters are not meant for typical use.

void_matid

Material ID to be used when raytrace is outside the problem.

Default 0

Type integer

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
axes	raytrace_db	axes
mix_tolerance	raytrace_db	mix_tolerance
rays_deterministic	raytrace_db	deterministic
rays_per_face	raytrace_db	rays_per_face
void_matid	raytrace_db	void_matid

14.26 [MANUALWW]

14.26.1 Parameters

input

Manual weight window HDF5 file.

Type file path for reading (extension ‘.h5’)

normalization

Multiplicative normalization for weight windows.

Type positive real number

14.27 [RUN]

The [RUN] database enables support for running the Omnibus executable with an inline command `omnibus-run`. The auto-run feature will format and echo program output to the screen, and it automatically saves the output and error streams to disk.

If the `SCALE` and `DATA` environment variables are not set, `omnibus-run` will use the values determined at configuration time.

14.28 [RUN=serial]

Run as a serial process on the local machine, echoing output to the user. If the `omnibus-run` process is aborted, the `omnibus` command will also abort.

14.28.1 Advanced parameters

These parameters are not meant for typical use.

omnibus

Path to the Omnibus executable.

Default ‘/SCALE/bin/omnibus’

Type file path for reading

14.29 [RUN=mpi]

Run as an MPI process on the local machine, echoing output to the user. If the `omnibus-run` process is aborted, the `mpirun omnibus` command will also abort.

14.29.1 Parameters

np

Number of processors to run.

Type positive integer

mpiexec_args

Optional arguments passed to mpiexec..

Default []

Type list of strings

machinefile

Optional MPI machinefile for running on multiple nodes..

Default

“

Type file path for reading (empty value allowed)

14.29.2 Advanced parameters

These parameters are not meant for typical use.

mpiexec

Path to the MPI run command.

Default u'`/opt/clang/openmpi/bin/mpiexec`'

Type file path for reading

npflag

Number of processors flag.

Default u'-np'

Type string

omnibus

Path to the Omnibus executable.

Default '/SCALE/bin/omnibus'

Type file path for reading

14.30 [RUN=pbs]

Create a PBS run file for this job. A typical PBS run block will look like:

```
[RUN=pbs]
nodes      1
ppn       16
pmem     7900mb
walltime "24:00:00"
```

If the omnibus-run command is aborted while the job is run, the job **will not** be automatically aborted. You must run qdel separately to abort the job.

14.30.1 Parameters

nodes

Number of nodes to use.

Type positive integer

ppn

Number of processors per node.

Type positive integer

pmem

Amount of memory per processor (e.g. ‘7900mb’).

Default

‘’

Type string

extra_cmds

Extra command line arguments.

Default

‘’

Type string

name

Job name.

Type string with only a-z,A-Z, 0-9, - and _

walltime

Wall time limit.

Note: It is common to have colons as part of the wall time; since colons must be escaped in Omnibus ASCII input, you will almost always need to escape the wall time input parameter:

```
[RUN=pbs]
walltime "24:00:00"
```

Default

‘’

Type string

queue

Queue to use.

Default

‘’

Type string

join

Output joining flags.

Default ‘oe’

Type string

email

Email address of recipient.

Type string

when_email

When to email.

Default ‘ea’

Type string

qsub

PBS submission command or path.

Default ‘qsub’

Type string

qstat

PBS status command or path.

Default ‘qstat’

Type string

qdel

PBS deletion command or path.

Default ‘qdel’

Type string

detach

Whether to simply submit the job and to not follow it.

Default False

Type boolean

mpiexec_args

Optional arguments passed to mpiexec..

Default []

Type list of strings

14.30.2 Advanced parameters

These parameters are not meant for typical use.

mpiexec

Path to the MPI run command.

Default u’/opt/clang/openmpi/bin/mpiexec’

Type file path for reading

omnibus

Path to the Omnibus executable.

Default ‘/SCALE/bin/omnibus’

Type file path for reading

14.30.3 Additional defaults

- Email address defaults to `git config author.email`.
- The default job name is the problem input base name..
- Automatically set MPI arguments and processors for clusters.

14.31 [RUN=titan]

Create a PBS run file for this job, specifically for Cray Titan/EOS machines. A typical PBS run block will look like:

```
[RUN=titan]
nodes      1024
ppn       16
account   nfi000
walltime "12:00:00"
```

Just like with PBS, if the `omnibus-run` command is aborted while the job is run, the job **will not** be automatically aborted. You must run `qdel` separately to abort the job.

If the number of cores is less than or equal to 8 (the number of “shared core units”), the `-j 2` option will automatically be appended to stride the cores by 1.

Note: When running on Titan, `omnibus-run` must be executed from inside the Lustre file system, and the installation of Omnibus, as well as the necessary SCALE data, must reside on Lustre as well.

14.31.1 Parameters

`nodes`

Number of nodes to use.

Type positive integer

`ppn`

Number of MPI tasks per node.

Default 16

Type positive integer

`mem`

Memory required per MPI task (with G/M/K extension).

Default

“”

Type string

`account`

Account number to charge for time (e.g., NFI000).

Type string

`name`

Job name.

Type string with only a-z,A-Z, 0-9, - and _

walltime

Wall time limit.

Default

“”

Type string**queue**

Queue to use.

Default

“”

Type string**join**

Output joining flags.

Default ‘oe’**Type** string**email**

Email address of recipient.

Type string**when_email**

When to email.

Default ‘ea’**Type** string**qsub**

PBS submission command or path.

Default ‘qsub’**Type** string**qstat**

PBS status command or path.

Default ‘qstat’**Type** string**qdel**

PBS deletion command or path.

Default ‘qdel’**Type** string**detach**

Whether to simply submit the job and to not follow it.

Default False**Type** boolean**debug**

Whether to show the aprun layout.

Default False

Type boolean

14.31.2 Advanced parameters

These parameters are not meant for typical use.

aprun

Path to the aprun command.

Default u'./opt/clang/openmpi/bin/mpieexec'

Type file path for reading

omnibus

Path to the Omnibus executable.

Default '/SCALE/bin/omnibus'

Type file path for reading

threads_per_task

Number of threads per MPI task (-d).

Default 1

Type positive integer

extra_cmds

Extra command line arguments to turn on core dumping.

Default 'ulimit -c unlimited; export ATP_ENABLED=1'

Type string

14.31.3 Additional defaults

- Email address defaults to git config author.email.
- The default job name is the problem input base name..

POSTPROCESSING

In order to postprocess HDF5 data with the Python front-end, it's necessary to download `h5py`, which is often easily installed with package managers such as MacPorts.

15.1 Cylindrical tally postprocessing

To load the HDF5 database into Python:

```
>>> from omnibus.postprocess import cyltally
>>> tallies = cyltally.load('cyltally.h5')
INFO: Loading HDF5 from cyltally.h5
```

The result is a dictionary of tallies, each corresponding to a CYLTALLY entry in the Omnibus input, with the corresponding name attribute:

```
>>> tallies.keys()
['n:SVXF:VXF-3', 'n:TRRH:FT-E7']
>>> tal = tallies['n:SVXF:VXF-3']
>>> print tal
TallyResult: n:SVXF:VXF-3
```

The top-level tally has several attributes:

```
>>> print tal.description
neutron flux in SVXF target location VXF-3
>>> print tal.mesh.translation
[-9.15368 38.12784 0.]
>>> tal.bin_bounds.num_total_groups
200
>>> tal.multipliers
(u'flux',)
```

And its data can be dumped to a multigroup Silo file for visualization:

```
>>> tal.write_to_silo()
INFO: Writing cylindrical tally 'n:SVXF:VXF-3' to n_SVXF_VXF_3.silo
```

The next layer of the tally hierarchy are the multipliers specified for each tally, which we provide a dictionary-like shortcut for accessing:

```
>>> print tal.tallies
{'flux': <TallyMultiplierResult ...>}
>>> rxntal = tal['flux']
>>> print rxntal
TallyMultiplierResult: flux
```

These have the actual flux results stored as a TallyContainer of mean and variance:

```
>>> rxntal.total  
TallyContainer(mean=array([[[ 7.93285055e-05]]]), var=array([[  
1.46720455e-12]]))
```

The indexing for the subtally results is r, theta, z.

The TallyContainer also provides a relative error function for convenience, which will squelch “divide-by-zero” errors given when a tally bin encountered no particles, setting the relative error to ‘inf’:

```
>>> rxntal.total.re  
array([[[ 0.01526919]]])
```

If energy binning was used, the tally multiplier result will also have energy-binned tallies, indexed by energy-bin first, then r, theta, z:

```
>>> rxntal.binned.mean.shape  
(200, 1, 1, 1)
```

The tally results can be sliced and printed alongside the lower and upper energy groups:

```
>>> lower = rxntal.bin_bounds.get_lower_bounds()  
>>> upper = rxntal.bin_bounds.get_upper_bounds()  
>>> for (l, u, mean, rel) in zip(lower, upper,  
...      rxntal.binned.mean[:,0,0,0], rxntal.binned.re[:,0,0,0]):  
...     print "%.  
4e to %.  
4e: %10.  
3g | %.  
2f" % (l, u, mean, rel)  
...  
1.9640e+07 to 2.0000e+07: 0 | inf  
[snip]  
1.5000e-01 to 1.8400e-01: 1.16e-06 | 0.04  
1.2500e-01 to 1.5000e-01: 1.82e-06 | 0.03  
1.0000e-01 to 1.2500e-01: 3.64e-06 | 0.02  
7.0000e-02 to 1.0000e-01: 9.33e-06 | 0.02  
5.0000e-02 to 7.0000e-02: 1.09e-05 | 0.02  
4.0000e-02 to 5.0000e-02: 7.33e-06 | 0.02  
3.0000e-02 to 4.0000e-02: 8.19e-06 | 0.02  
2.1000e-02 to 3.0000e-02: 7.66e-06 | 0.02  
1.4500e-02 to 2.1000e-02: 5.2e-06 | 0.02  
1.0000e-02 to 1.4500e-02: 2.99e-06 | 0.02  
5.0000e-03 to 1.0000e-02: 2.41e-06 | 0.02  
2.0000e-03 to 5.0000e-03: 7.89e-07 | 0.03  
5.0000e-04 to 2.0000e-03: 1.32e-07 | 0.06  
1.0000e-05 to 5.0000e-04: 9.89e-09 | 0.17
```

CHAPTER
SIXTEEN

OMNIBUS.CONVERTERS PACKAGE

The Omnibus converters package allows results from other code systems to be read into Omnibus postprocessing containers for easy data analysis. This enables straightforward integration of other common nuclear engineering codes with powerful tools such as the `matplotlib` plotting library and the `pandas` data analysis library.

16.1 omnibus.converters.mctal module

Read MCNP mctal files into Omnibus postprocessing data structures.

Caution: MCNP is unable to represent complex user input (e.g. cell unions and multiplier selections) in the mctal file. Sometimes zeroes in the various “meshes” returned by this reader represent these complicated expressions. The MCNP input (or output file) will be needed to properly interpret these instances.

Note: When reading multigroup data, this reader sets an artificial small value for the lower energy of the lowest bin (which is implicitly created by MCNP). For correctness, it should be replaced with the problem’s lower cutoff energy for the tallied particle type.

`class omnibus.converters.mctal.Kcode(*args, **kwargs)`

Bases: `omnibus.postprocess.field.Field`

Eigenvalue solution properties.

Attributes

<code>dims</code>	Dimension names for each axis.
<code>keff</code>	Average col/abs/trk-len keff
<code>keff_fom</code>	Average col/abs/trk-len keff figure of merit
<code>keff_stdev</code>	Average col/abs/trk-len keff standard deviation
<code>location</code>	Get a description of where this field has been sliced from.
<code>num_active_cycles</code>	
<code>num_cycles</code>	
<code>num_histories_per_cycle</code>	Number of histories used in this cycle
<code>shape</code>	Get the shape of the data stored in this field..

Methods

keff

Average col/abs/trk-len keff

keff_fom

Average col/abs/trk-len keff figure of merit

keff_stddev

Average col/abs/trk-len keff standard deviation

num_histories_per_cycle

Number of histories used in this cycle

class omnibus.converters.mctal.**McnpAxis**(*args, **kwargs)

Bases: *omnibus.postprocess.field.LabelAxis*

Attributes for one ‘dimension’ of a tally.

Total and cumulative values are counted as part of the total length.

Attributes

mnemonic: string	Short name of the MCNP variable index.
------------------	--

Methods

describe_index(i)

index(value) Find the index of a value on this mesh.

index (value)

Find the index of a value on this mesh.

shape

The actual shape of this axis’ mesh.

class omnibus.converters.mctal.**McnpCellTallies**(*args, **kwargs)

Bases: *omnibus.postprocess.celltally.CellTallies*

Collection of all cell tallies present in a file.

Attributes

kcode :	Eigenvalue solution data
---------	--------------------------

Methods

class omnibus.converters.mctal.**McnpCellTallyResult**(*args, **kwargs)

Bases: *omnibus.postprocess.celltally.CellTallyResult*

Data corresponding to a single cell tally.

This corresponds to all the results from a single fNN specification in MCNP.

Attributes

name	(int) Tally number
pt	(str) Particle type: n p e
taltype	(str) Tally type: none point ring fip fir fic
axes	(list) Dimensions on the tally data
tfc :	Tally fluctuation info.

Methods

```
class omnibus.converters.mctal.McnpTallyField(*args, **kwargs)
Bases: omnibus.postprocess.field.Field
```

Multi-dimensional field for storing and accessing bulk tally data.

This provides accessors for retrieving the originally stored mean and variance, as well as calculation methods for returning relative error. Additionally, in the case of multi-dimensional data, it allows advanced slicing of the data.

Attributes

mean	(array (or float)) The mean values for this tally.
re	(array (or float)) Relative errors saved from MCNP.

Methods

bin_bounds

If this is an energy-binned tally, return the bin bounds.

If not energy-binned (or a slice on the energy axis has already been taken), this will raise a `KeyError`.

cells

Get an integer array of MCNP cells.

var

Calculate variance from the relative error.

This will return an array if the stored mean and variance are arrays.

$$\text{re} = \frac{\sqrt{\text{var}}}{\text{mean}}$$

```
class omnibus.converters.mctal.Reader(filepath)
```

Bases: `omnibus.textfile.TextFile`

Helper class to read mctal file into tally data structure.

Attributes

closed	
eof	Return whether we're at the end of the file

Methods

<code>load()</code>	Read and parse the file, setting <code>tallies</code> .
<code>read_values(num)</code>	Read <code>num</code> space-separated values from the file.

`load()`

Read and parse the file, setting `tallies`.

`read_values(num)`

Read `num` space-separated values from the file.

`tallies = None`

Tally data

`version = None`

Version number for switching on format

class `omnibus.converters.mctal.Tfc(*args, **kwargs)`

Bases: `omnibus.container.Container`

Tally fluctuation properties.

Attributes

<code>index</code> :	Index into tally bin we're reporting
<code>data</code>	(dtype=Tfc.dtype) Record field of data

`omnibus.converters.mctal.load(*args, **kwargs)`

Read cell tallies and kcode data from a mctal file.

16.2 omnibus.converters.meshtal module

Read MCNP meshtal files into Omnibus postprocessing data structures.

class `omnibus.converters.meshtal.McnpMeshTallies(*args, **kwargs)`

Bases: `omnibus.postprocess.mesh tally.MeshTallies`

Collection of all MCNP mesh tallies present in a file.

Methods

class `omnibus.converters.meshtal.McnpMeshTallyResult(*args, **kwargs)`

Bases: `omnibus.postprocess.mesh tally.MeshTallyResult`

Data corresponding to a single user tally.

Attributes

name	(int) Tally number
pt	(str) Particle type: n p e

Methods

`energy_bins`

Return energy bin boundaries (backwards compatibility)

`omnibus.converters.meshtal.load(*args, **kwargs)`

Read mesh tallies from a meshtal file.

16.3 omnibus.converters.monaco module

Read Monaco tallies into Omnibus postprocessing data structures.

Warning: Currently this class is only tested for multigroup cell tallies.

Note: When reading multigroup data, this reader sets an arbitrarily small value for the lower energy of the lowest bin (which is implicitly created by Monaco). For correctness, it should be replaced with the problem's lower cutoff energy for the tallied particle type.

`class omnibus.converters.monaco.MonacoCellTallyResult(*args, **kwargs)`

Bases: `omnibus.postprocess.tally.TallyResult`

Results for a single cell tally in Monaco.

Attributes

limits	(dict) Key->value map of region/unit/media on the input tally.
volume	(float) Volume of the tally.
multiplier	(float) Scalar multiplier used for the tally.

Methods

`omnibus.converters.monaco.load(*args, **kwargs)`

Read cell tallies from Monaco output files.

16.4 omnibus.converters.opus module

`omnibus.converters.opus.load(*args, **kwargs)`

Read concentrations from Opus files.

16.5 omnibus.converters.triton module

Convert Triton number density output to same format as Omnibus depletion concentration field.

Note: This converter relies on an external tool to convert a Triton Fortran intermediate file to a csv file. It additionally requires cell volumes and the total heavy metal mass to scale the results correctly.

`omnibus.converters.triton.load(*args, **kwargs)`

Read depletion data from a processed Triton number density field.

Parameters `csv_inp` : str

Path to the CSV file.

`vol_inp` : str

Path to the volume file.

`hm_mass` : float

Heavy metal mass.

16.6 omnibus.converters.vesta module

`omnibus.converters.vesta.load(*args, **kwargs)`

Read depletion results from a series of VESTA files.

16.7 Module contents

Convert third-party results into Omnibus postprocessing formats.

OMNIBUS.POSTPROCESS PACKAGE

17.1 Submodules

17.2 omnibus.postprocess.celltally module

class `omnibus.postprocess.celltally.CellTallies(*args, **kwargs)`
Bases: `omnibus.postprocess.tally.Tallies`

Collection of all cell tallies present in a file.

Methods

class `omnibus.postprocess.celltally.CellTallyMultiplierResult(*args, **kwargs)`
Bases: `omnibus.postprocess.tally.TallyMultiplierResult`

Cell union tally results for a single tally multiplier.

These store all the cell unions specified by the user for this multiplier. A “multiplier” can be a reaction or a response.

Energy-binned tallies will have the shape (num_groups, num_unions).

Attributes

<code>unions</code>	(list of strings) Cell union labels corresponding to each tally entry.
---------------------	--

`num_unions`

The number of cell unions in this tally.

class `omnibus.postprocess.celltally.CellTallyResult(*args, **kwargs)`
Bases: `omnibus.postprocess.tally.TallyResult`

Results for a single CELL tally, with all its multipliers.

This corresponds to all the results from a single [TALLY][CELL] block in an Omnibus input.

Attributes

unions	(list of strings) Cell union labels corresponding to each tally entry.
volumes	(array) Volumes used when the cell tally results were normalized. These may be given by the user, calculated via MCNP or the like, or unity if unknown. Variances and means in the tallies have <i>already</i> been normalized by these volumes.

Methods

summarize :	Print a summary of the energy-integrated results for this tally.
-------------	--

classmethod from_group (group, shared)

Build results for a single CELL tally from an HDF5 file.

This creates the union labels from cell lengths and *ids*. We use the shared data to convert cellids to cell labels.

classmethod from_group_v1 (group, *args, **kwargs)

Backward compatibility: build tallies from older HDF5 files.

classmethod from_group_v2 (group, shared)

Backward compatibility: build tallies from older HDF5 files.

This creates the union labels from cell lengths and labels.

classmethod from_group_v3 (group, shared)

Build results for a single CELL tally from an HDF5 file.

classmethod from_group_v4 (group, shared)

Build results for a single CELL tally from an HDF5 file.

This creates the union labels from cell lengths and *ids*. We use the shared data to convert cellids to cell labels.

num_unions

The number of cell unions in this tally.

summarize (file=None)

Print a summary of the energy-integrated results for this tally.

Parameters file : file-like object

The destination for the output stream. Default is `sys.stdout`.

class omnibus.postprocess.celltally.SharedCellTallyData (*args, **kwargs)

Bases: `omnibus.postprocess.tally.SharedTallyData`

Data shared by all tallies in a file.

This is mostly used in construction to pass problem metadata etc. between tallies.

Attributes

cell_labels: array	Labels of all cells in the problemzz
--------------------	--------------------------------------

Methods

class omnibus.postprocess.celltally.**SharedCellTallyMultiplierData** (*args,
 **kwargs)

Bases: *omnibus.postprocess.celltally.SharedCellTallyData*

Data shared by a single cell tally.

Attributes

cell_unions: array	Labels of all cells in the problemzz
--------------------	--------------------------------------

Methods

class omnibus.postprocess.celltally.**SingleTallyContainer**

Bases: tuple

Attributes

binned	Alias for field number 1
label	Alias for field number 0
total	Alias for field number 2

Methods

count(...)	
index((value, [start, ...))	Raises ValueError if the value is not present.

binned

Alias for field number 1

label

Alias for field number 0

total

Alias for field number 2

omnibus.postprocess.celltally.**dump_csv** (*args, **kwargs)

Dump a single TallyResult object to a file.

With energy binning, the file looks like:

-	-	RXN1	-	RXN1	-	RXN1	-	RXN2	-	RXN2	-	RXN2	-
-	-	U1	-	U2	-	U3	-	U1	-	U2	-	U3	-
Upper	Lower	mean	re										
E1	E2	###	#	##	#	##	#	##	#	##	#	##	#

E2	E3	###	# #	# # #	# #	# # #	# #	# # #	# #	# # #	# #	# # #	# #
E3	E4	###	# #	# # #	# #	# # #	# #	# # #	# #	# # #	# #	# # #	# #
E4	E5	###	# #	# # #	# #	# # #	# #	# # #	# #	# # #	# #	# # #	# #
	TOTAL	###	# #	# # #	# #	# # #	# #	# # #	# #	# # #	# #	# # #	# #

`omnibus.postprocess.celltally.load(*args, **kwargs)`

Get a list of CellTallyResult objects from an HDF5 file.

Each object in this list corresponds to a single [TALLY][CELL] entry given by the user. It may have several multipliers, each of which has data for all the enclosed cell unions.

`omnibus.postprocess.celltally.load_from_manager(manager)`

Load tallies from the output manager object.

17.3 omnibus.postprocess.collisions module

`class omnibus.postprocess.collisions.CollisionDiagnostic(*args, **kwargs)`

Bases: `omnibus.container.Container`

Collision diagnostic results for a problem.

The collision diagnostic calculates the average number of collisions per history, and average weight loss per history, in each material, in each nuclide, for each reaction.

Attributes

<code>data</code>	(pd.DataFrame or np.array) Record array of materials, zuids, MT numbers, and collision/weight loss per history.
<code>metadata</code>	(dict) Metadata associated with program run.
<code>num_histories</code>	(float) Number of active particle histories sampled.
<code>num_collisions</code>	(float) Total number of collisions during active histories.

Methods

`extract(**kwargs)` Extract a cross section view for a single key (matid/zaid/mt).

`from_h5_group(group[, name, version])`

`integrate(key)` Integrate collision results over all keys but the given one.

`reindex_from_materials(mixtures)` Change index from matid to material name.

`summarize([threshold, file])` Print a summary of important collisions in this tally.

`extract` (kwargs)**

Extract a cross section view for a single key (matid/zaid/mt).

```
>>> df.extract(mt=2)
```

`integrate` (key)

Integrate collision results over all keys but the given one.

`material_indexing`

Return whether we're indexed by 'matid' or 'mat'.

`reindex_from_materials` (mixtures)

Change index from matid to material name.

This permanently changes the key name; you'll have to reload the data if you want matids back.

```
>>> xmldata = OmnibusOutput.from_xml('output.xml')
>>> mixtures = xmldata.db['PHYSICS']['mixtures']
>>> coldat.reindex_from_materials(mixtures)
```

`summarize`(*threshold*=0.01, *file*=None)

Print a summary of important collisions in this tally.

This integrates over all materials, printing a list of reactions with the most collisions per history (over the given threshold value).

Parameters `threshold` : float

Fraction of total collisions per history at which we truncate output of most frequent nuclides per collision

`file` : file-like object

The destination for the output stream. Default is `sys.stdout`.

`omnibus.postprocess.collisions.load(*args, **kwargs)`

Get a single CollisionDiagnostic from an HDF5 file.

`omnibus.postprocess.collisions.load_from_manager(manager)`

Load tallies from the output manager object.

17.4 omnibus.postprocess.compositions module

class `omnibus.postprocess.compositions.Composition(*args, **kwargs)`

Bases: `omnibus.container.Container`

A single composition.

Attributes

<code>name</code>	(string) Composition name.
<code>components</code>	(dict) Map of ZAID -> number densities (atoms / b-cm)

Methods

<code>from_group</code> (root)	Build results for all compositions in the file.
<code>get(key[, default])</code>	

classmethod `from_group`(*root*)

Build results for all compositions in the file.

class `omnibus.postprocess.compositions.Comps(*args, **kwargs)`

Bases: `omnibus.container.Container`

All composition data from an HDF5 file

Attributes

compositions	(list of Composition) All compositions from the file
nuclides	(list of Nuclide) All nuclides from the file
metadata	(dict) Problem metadata.

Methods

from_group(root) Build results for all compositions in the file.

classmethod from_group (root)

Build results for all compositions in the file.

`omnibus.postprocess.compositions.load(handle)`

Read composition data from an HDF5 file.

`omnibus.postprocess.compositions.load_from_manager(manager)`

Load compositions from the output manager object.

17.5 omnibus.postprocess.cyltally module

class `omnibus.postprocess.cyltally.CylMesh(*args, **kwargs)`

Bases: `omnibus.container.Container`

Container for a cylindrical mesh.

The Cylindrical mesh is indexed with [r, z, theta].

Attributes

r	(array) Radial mesh points.
z	(array) Z-axis mesh points.
t	(array) Azimuthal (theta) mesh points.
translation	(array) Length-3 translation for the cylindrical mesh.
rotation	(array) 3x3 rotation matrix for the cylindrical mesh.

cell_shape

The proper shape for cell-centered data associated with the mesh.

returns: (r, z, theta).

class `omnibus.postprocess.cyltally.CylTallies(*args, **kwargs)`

Bases: `omnibus.postprocess.tally.Tallies`

Collection of all cylindrical tallies present in a file.

Methods

```
class omnibus.postprocess.cyltally.CylTallyMultiplierResult (*args, **kwargs)
Bases: omnibus.postprocess.tally.TallyMultiplierResult
```

Cylindrical mesh tally results for a single tally multiplier.

The cylindrical multiplier tally result reshapes the data to conform with the cylindrical mesh: energy-integrated tally results will have the indexing [r, z, theta], and energy-binned tallies are accessed like [e, r, z, theta].

A “multiplier” can be a reaction or a response.

Attributes

mesh	(<i>CylMesh</i>) The mesh used in this tally
------	--

```
class omnibus.postprocess.cyltally.CylTallyResult (*args, **kwargs)
Bases: omnibus.postprocess.tally.TallyResult
```

Results for a single CYLMESH tally, with all its multipliers.

This corresponds to all the results from a single [TALLY][CYLMESH] block in an Omnibus input.

Attributes

mesh	(<i>CylMesh</i>) The mesh used in this tally
------	--

Methods

write_to_silo :	Write all the embedded tallies to a Silo file.
-----------------	--

classmethod from_group (*group, shared*)

Build results for a single CYLMESH tally from an HDF5 file.

The version keyword is the Exnihilo revision number, used for backward compatibility.

classmethod from_group_v1 (*group, shared*)

Backward compatibility: build tallies from older HDF5 files.

classmethod from_group_v4 (*group, shared*)

Build results for a single CYLMESH tally from an HDF5 file.

The version keyword is the Exnihilo revision number, used for backward compatibility.

write_to_silo (*args, **kwargs)

Write all the embedded tallies to a Silo file.

Note that this requires Python bindings to be enabled in Exnihilo.

Parameters path :

Path for the Silo output.

```
omnibus.postprocess.cyltally.load (*args, **kwargs)
```

Get a list of TallyResult objects from an HDF5 file.

Each object in this list corresponds to a single [TALLY][CYLMESH] entry given by the user. It may have several multipliers, each of which has r/th/z/e or r/th/z data.

```
omnibus.postprocess.cyltally.load_from_manager (manager)
```

Load tallies from the output manager object.

17.6 omnibus.postprocess.depletion module

class `omnibus.postprocess.depletion.CrossSections(*args, **kwargs)`
Bases: `omnibus.postprocess.depletion.DepletionField`

Collapsed ORGEN one-group cross sections.

These are provided for every ORIGEN zaid and mt in each cell at each step.

Attributes

Methods

<code>write_to_csv:</code>	Write the cross sections to a CSV file.
----------------------------	---

as_dataframe()

Convert the cross sections to a dataframe for easier processing.

The zuids and MTs are expanded into separate axes.

write_to_csv(*args, **kwargs)

Dump the xs object to a file.

File looks like:

ZAID	MT	STEP	CELL 0	CELL 1	CELL 2
zaid1	mt1a	0	##	##	##
		1	##	##	##
	mt1b	0	##	##	##
		1	##	##	##
zaid2	mt2a	0	##	##	##
		1	##	##	##
	mt2b	0	##	##	##

class `omnibus.postprocess.depletion.DepletionField(*args, **kwargs)`

Bases: `omnibus.postprocess.field.Field`

Store multi-D data with axes and labels.

Unlike the tally field, this doesn't have variances, and they aren't necessarily statistically calculated quantities.

Attributes

<code>dims</code>	Dimension names for each axis.
<code>location</code>	Get a description of where this field has been sliced from.
<code>shape</code>	Get the shape of the data stored in this field..

Methods

<code>from_group(group, key)</code>

```
class omnibus.postprocess.depletion.DepletionTally(*args, **kwargs)
```

Bases: *omnibus.container.Container*

Results from a depletion cycle.

Note: due to the normalization possibly changing between steps, the step boundaries may be discontinuities for results that depend instantaneously on the normalization. Thus “power”, “flux”, “totpower”, “totflux” at the steps (times) labeled, e.g. “t1”, may not be well defined and instead the following convention is used.

For the primary outputs “t0”, “t1”, ... the values are those of the starting step, i.e., right after the corresponding time ($t_{<N>} + \text{epsilon}$), except for the final step “ $t_{<\text{last}>}$ ” for which it is the power of the ending last step, i.e., right before the corresponding time ($t_{<\text{last}>} - \text{epsilon}$)

For additional predictor outputs “t1p”, “t2p”, ... the values are those of the ending step, i.e., right before the corresponding time ($t_{<N>} - \text{epsilon}$).

There is no ambiguity for the middlestep “t0m”, “t1m”, ... values produced by middlestep schemes.

Attributes

num_dens	(array of doubles) Array of number densities. [step][cell][zaid]
xs	(array of doubles) Array of collapsed 1-grp cross sections from ORIGEN. [step][cell][zaid-mt]
power	(list of doubles) Array of power in all regions if depleting by power. [step][cell]
flux	(list of doubles) Array of flux in all regions at all steps. [step][cell]
volumes	(array of doubles) Array of volumes for all depletion regions. [cell]
simtime	(array of doubles) Array of running simulation time of the problem [step]
burnup	(array of doubles) Array of the running burnup of the problem (MWd/MTU) [step]
totpower	(array of doubles) Array of total power in the system for each time step if depl by power. [step]
totflux	(array of doubles) Array of total flux in the system for each time step if depl by flux. [step]
hm_mass	(scalar double) The total heavy metal mass in the system at beginning of calculation. (Metric tons of heavy metal.)
hm_fraction	(scalar double) The fraction of heavy metal mass in the system (for normalization)
metadata	(dict) The metadata from the problem run.

Methods

at_step :	Get a slice of the DepletionTally for a given time step number.
summarize :	Print a summary of the burn data for every time step.

at_step (step)

Return a view of the tally data at a single depletion time step.

Returns DepletionTally (or subclass): tally result evaluated at the given time step.

convert_to_time ()

Return a new depletion tally with only physical time steps.

This effectively removes data for “predictor” steps if present, allowing easier comparison with other depletion codes. It also replaces the “step” index with a “time” index.

It does not modify the original depletion tally.

Returns New depletion tally indexed by time.

classmethod `from_group`(*root*)

Build results for all depletion data.

summarize(*file=None*)

Print a summary of the depletion burn data.

Parameters `file` : file-like object

The destination for the output stream. Default is `sys.stdout`.

class `omnibus.postprocess.depletion.NumberDensities`(*args, **kwargs)

Bases: `omnibus.postprocess.depletion.DepletionField`

Number density of every nuclide in each cell at each step.

Attributes

<code>data</code>	(array of doubles) Array of all number densities. [step][cell][zaid]
-------------------	--

Methods

`write_to_csv` : Write the number densities to a CSV file.

collapse_nuclides(*kwargs)

Combine nuclides and add their number densities.

ORIGEN internal nuclides can include duplicate zuids.

write_to_csv(*args, **kwargs)

Dump the number densities to a file.

File looks like:

CELL	STEP	ZAID1	ZAID2	ZAID3
0	t0	##	##	##
	t1	##	##	##
1	t0	##	##	##
	t1	##	##	##

class `omnibus.postprocess.depletion.RegionData`(*args, **kwargs)

Bases: `omnibus.postprocess.depletion.DepletionField`

Data about every depletion region at each step.

Attributes

<code>dims</code>	Dimension names for each axis.
<code>location</code>	Get a description of where this field has been sliced from.
<code>shape</code>	Get the shape of the data stored in this field..

Methods

`from_group(group, key)`

class omnibus.postprocess.depletion.**StepData**(*args, **kwargs)
Bases: *omnibus.postprocess.depletion.DepletionField*

Data about some property at each step.

Attributes

<code>dims</code>	Dimension names for each axis.
<code>location</code>	Get a description of where this field has been sliced from.
<code>shape</code>	Get the shape of the data stored in this field..

Methods

<code>from_group(group, key)</code>

omnibus.postprocess.depletion.**load**(*args, **kwargs)
Load depletion tallies.

omnibus.postprocess.depletion.**load_from_manager**(manager)
Load depletion data from the output manager object.

17.7 omnibus.postprocess.field module

This module provides a wrapper to Numpy arrays that stores multidimensional data, allowing it to be sliced, reordered, and easily accessed.

class omnibus.postprocess.field.**DataAxis**(*args, **kwargs)
Bases: *omnibus.container.Container*

Metadata about an axis of a field.

The mesh must be sorted in ascending order for the index method to work.

Attributes

<code>centers</code>	Obtain mesh values that match the centering of the data.
<code>shape</code>	The actual shape of this axis' mesh.

Methods

<code>describe_index(i)</code>	
<code>index(value)</code>	Find the index of a value on this mesh.

centers

Obtain mesh values that match the centering of the data.

index(value)

Find the index of a value on this mesh.

shape

The actual shape of this axis' mesh.

```
class omnibus.postprocess.field.Field(*args, **kwargs)
```

Bases: *omnibus.container.Container*

Multi-dimensional field for storing and accessing multi-D bulk data.

It enables advanced slicing of the data in this field. It should be subclassed to be used, and the `_fields` attribute should be set to the relevant fields.

Attributes

axes	(list of DataAxis) Attributes about each dimension/axis of the data.
hyper-slice	(list of (axis, index) tuples.) If this is a view into larger tally data, the hyperslice is the list of axes and values at which those dimensions are being evaluated.

Methods

<code>as_dataframe()</code>	Convert the field to a Pandas dataframe for easier processing.
<code>axis(dim)</code>	Get the axis index corresponding to the given name.
<code>axis_index(dim)</code>	Get the axis index corresponding to the given dimension name.
<code>describe_index(i)</code>	Given an index (of the same dimensionality as this hyperslice), print the description of those indices.
<code>reorder(newdims)</code>	Reorder the indices of the data by dimension name.
<code>subset(**kwargs)</code>	Extract a subset of data along one or more axes.
<code>subset_by_index(**kwargs)</code>	Extract a subset of data along one or more axes.
<code>sum_over(axis_name)</code>	Return the sum over the given axis.
<code>xs(**kwargs)</code>	Extract a hyperslab view of the tally field data.
<code>xs_by_index(**kwargs)</code>	Extract a hyperslab view of the tally field data.

as_dataframe()

Convert the field to a Pandas dataframe for easier processing.

Fields are converted to series in the dataframe, and the axes are converted to a multi-index.

Note: It may be necessary to run call `result.sortlevel(inplace=True)` on the result to perform some Pandas operation. This is because some axes may not be lexicographically ordered.

axis (dim)

Get the axis index corresponding to the given name.

Examples

This enables easier access to meshes and the like.

```
>>> tal.axis('x').mesh
```

axis_index (dim)

Get the axis index corresponding to the given dimension name.

Examples

To collapse multi-dimensional data using the `numpy.sum` function, you can determine the appropriate axis index using this method:

```
>>> tal.total.sum(axis=tal.axis_index("x"))
```

`describe_index(i)`

Given an index (of the same dimensionality as this hyperslice), print the description of those indices.

`dims`

Dimension names for each axis.

`location`

Get a description of where this field has been sliced from.

It will return an empty string if this is the full, original field.

`reorder(newdims)`

Reorder the indices of the data by dimension name.

Parameters `newdims` : list of strings

New dimension names.

Examples

This allows conversion of data stored as ZYXE to XYZE, etc. The input argument is a list of dimensions that should end up in the relative order given:

```
>>> newvals = tal.reorder(("x", "y"))
```

`shape`

Get the shape of the data stored in this field..

`subset(**kwargs)`

Extract a subset of data along one or more axes.

Input arguments are lists of values on each axis.

`subset_by_index(**kwargs)`

Extract a subset of data along one or more axes.

Input arguments are either `numpy` boolean masks or array indices.

`sum_over(axis_name)`

Return the sum over the given axis.

This returns a new field.

`xs(**kwargs)`

Extract a hyperslab view of the tally field data.

Input arguments are values on each axis, similar to the `pandas` “`xs`” function.

`xs_by_index(**kwargs)`

Extract a hyperslab view of the tally field data.

Input arguments must be integer indices, slices, or `numpy` boolean arrays: to take a cross section by value, use the `xs` method.

class omnibus.postprocess.field.LabelAxis (*args, **kwargs)
Bases: omnibus.container.Container

Metadata about an axis of a field that acts like labels.

This is currently used for tally multipliers and unions.

Attributes

centers
shape

Methods

describe_index(i)
index(value)

omnibus.postprocess.field.build_axes (group, dimnames, shape)

From an HDF5 group or dict, build a list of axes.

17.8 omnibus.postprocess.fissionsite module

class omnibus.postprocess.fissionsite.FissionSites (*args, **kwargs)
Bases: omnibus.container.Container

Stores fission sites for all cycles.

Attributes

cycles	(dict) For each cycle, a Sites instance.
metadata	(dict) Problem metadata.

Methods

from_group(root)
write_to_silo(*args, **kwargs) Write all sites to multiple silo files.

write_to_silo (*args, **kwargs)

Write all sites to multiple silo files.

class omnibus.postprocess.fissionsite.Sites (*args, **kwargs)

Bases: omnibus.container.Container

Fission sites for a single cycle.

Attributes

sites	(array) x/y/z coordinates of all fission sites for all cycles
-------	---

Methods

<u>write_to_silo(path)</u>
--

`omnibus.postprocess.fissionsite.load(handle)`

Read fission sites from an HDF5 file.

`omnibus.postprocess.fissionsite.load_from_manager(manager)`

Load depletion data from the output manager object.

17.9 omnibus.postprocess.history module

class `omnibus.postprocess.history.Events(*args, **kwargs)`

Bases: `omnibus.container.Container`

Description of events for all particle histories.

Attributes

events :	Individual events recorded.
history_bounds	(dict) Start and stop indices into events for all the history IDs.
metadata	(dict) The metadata from the problem run.

Methods

<u>as_dataframe()</u>	Convert the event data to a Pandas dataframe.
<u>from_group(root)</u>	Build results for all histories in the file.
<u>get_history(history)</u>	
<u>write_to_silo(*args, **kwargs)</u>	Write all the stored to a Silo file.

as_dataframe()

Convert the event data to a Pandas dataframe.

Pandas requires flattened dataframes (it can't handle pos/dir being arrays) so we convert to x/y/z and u/v/w.

Additionally we turn the events into a Pandas collection instead of an opaque enumeration.

classmethod from_group (root)

Build results for all histories in the file.

write_to_silo (*args, **kwargs)

Write all the stored to a Silo file.

Note that this requires Python bindings to be enabled in Exnihilo.

Parameters path :

Path for the Silo output.

`omnibus.postprocess.history.load(handle)`
Read history events from an HDF5 file.

`omnibus.postprocess.history.load_from_manager(manager)`
Load history event data from the output manager object.

17.10 omnibus.postprocess.kcode module

`class omnibus.postprocess.kcode.Entropy(*args, **kwargs)`
Bases: `omnibus.container.Container`

Shannon entropy tally during the transport run.

TODO: for now, this only uses the first depletion solve.

Methods

<code>from_manager(manager)</code>	Build from the manager of data.
<code>plot(ax)</code>	Plot entropy for a single depletion step.

`classmethod from_manager(manager)`
Build from the manager of data.

`plot(ax)`
Plot entropy for a single depletion step.

`class omnibus.postprocess.kcode.KcodeTallies(*args, **kwargs)`
Bases: `omnibus.container.Container`

Kcode results.

Methods

<code>from_manager(manager)</code>	Build from the manager of data.
------------------------------------	---------------------------------

`classmethod from_manager(manager)`
Build from the manager of data.

`class omnibus.postprocess.kcode.Keff(*args, **kwargs)`
Bases: `omnibus.postprocess.tally.TallyField`

K-effective tallies during the transport run.

Attributes

<code>mean</code>	(float) Mean keff.
<code>var</code>	(float) Estimated variance of keff.
<code>num_active</code>	(int) Number of active cycles in the kcode solve.
<code>keff_pl</code>	(array) Keff path length estimators in each cycle.

Methods

<code>at_step(step)</code>	Return a view of the keff data at a single depletion time step.
<code>from_manager(manager)</code>	Build from the manager of data.
<code>plot(ax)</code>	Plot keff for a single depletion step.

`at_step(step)`

Return a view of the keff data at a single depletion time step.

If depletion is disabled, this will raise an error.

Returns Tally (or subclass): tally result evaluated at the given time step.

`classmethod from_manager(manager)`

Build from the manager of data.

`plot(ax)`

Plot keff for a single depletion step.

class omnibus.postprocess.kcode.**Spatial**(*args, **kwargs)

Bases: `omnibus.postprocess.field.Field`

Spatial distribution tally during the transport run.

TODO: for now, this only uses the first depletion calculation.

Attributes

<code>dims</code>	Dimension names for each axis.
<code>kurtosis</code>	
<code>location</code>	Get a description of where this field has been sliced from.
<code>normalized</code>	
<code>shape</code>	Get the shape of the data stored in this field..
<code>skewness</code>	

Methods

<code>from_group(root)</code>	Build from the manager of data.
<code>from_manager(manager)</code>	
<code>plot(ax)</code>	Plot spatial convergence diagnostics.

`classmethod from_group(root)`

Build from the manager of data.

`plot(ax)`

Plot spatial convergence diagnostics.

`omnibus.postprocess.kcode.load_from_manager(manager)`

Load tallies from the given output manager object.

17.11 omnibus.postprocess.manager module

class omnibus.postprocess.manager.JsonWriter
 Bases: object

Write a nicer version of the XML output.

This writes a JSON version of the XML data entries

Outputs

Methods

__call__(manager)
write_rst(w)

```
class omnibus.postprocess.manager.Manager(db,filename)
```

Bases: object

Manage output from an Omnibus run.

Attributes

<code>db</code>	(dict) Dictionary of parsed XML output.
<code>dirname</code>	(string) Directory enclosing the output.
<code>basename</code>	(string) Name of output file without extensions.

Methods

<code>finalize()</code>	Call to write an RST file after postprocessing.
<code>from_json(json_output_path)</code>	Create a manager from a postprocessed <code>.out.json</code> file.
<code>from_xml(xml_output_path)</code>	Create a manager from an <code>.out.xml</code> file.
<code>get_timing(block, timer)</code>	Return the node-zero time for the given timer in the given block.
<code>load_tallies(typename)</code>	
<code>process(p)</code>	Handle postprocessing.
<code>to_output_path(relpath)</code>	Convert a relative output path in the database to the absolute output path, regardless of whether the

basename = None

Name of output file without extension

cwd = None

Path to current working directory

db = None

XML output database (various metadata)

descriptions = None

Temporary storage for read description file: block -> [lines]

dirname = None
Base directory for output

filename = None
XML output filename

finalize()
Call to write an RST file after postprocessing.

classmethod from_json(json_output_path)
Create a manager from a postprocessed .out.json file.

classmethod from_xml(xml_output_path)
Create a manager from an .out.xml file.

get_timing(block, timer)
Return the node-zero time for the given timer in the given block.

mixtures
Return mixtures if available, or None if not.

outputs = None
Output files created by omnibus and postprocessors.

postprocessors = None
Postprocessors by class name

pp_by_block = None
Postprocessors for each block, added in process

process(p)
Handle postprocessing.

to_output_path(relpath)
Convert a relative output path in the database to the absolute output path, regardless of whether the cwd has changed or the manager was opened from another directory.

17.12 omnibus.postprocess.mesh tally module

class omnibus.postprocess.mesh tally. **Mesh** (*args, **kwargs)
Bases: *omnibus.container.Container*

Container for a cartesian mesh.

The mesh is indexed with [x, y, z].

Attributes

x	(array) X-axis mesh points.
y	(array) Y-axis mesh points.
z	(array) Z-axis mesh points.

Methods

`from_group(group)`

cell_shape

Return the proper shape for cell-centered data so it can be accessed with [x, y, z].

class omnibus.postprocess.meshtally.**MeshTallies** (*args, **kwargs)

Bases: *omnibus.postprocess.tally.Tallies*

Collection of all mesh tallies present in a file.

Methods

class omnibus.postprocess.meshtally.**MeshTallyMultiplierResult** (*args, **kwargs)

Bases: *omnibus.postprocess.tally.TallyMultiplierResult*

Mesh tally results for a single tally multiplier.

The multiplier tally result reshapes the data to conform with the mesh: energy-integrated tally results will have the indexing [x, y, z], and energy-binned tallies are accessed like [bin, x, y, z].

A “multiplier” can be a reaction or a response.

Attributes

mesh	(<i>Mesh</i>) The mesh used in this tally
------	---

class omnibus.postprocess.meshtally.**MeshTallyResult** (*args, **kwargs)

Bases: *omnibus.postprocess.tally.TallyResult*

Results for a single MESH tally, with all its multipliers.

This corresponds to all the results from a single [TALLY][MESH] block in an Omnibus input.

Attributes

mesh	(<i>Mesh</i>) The mesh used in this tally
------	---

Methods

`from_group(group, shared)` Build results for a single MESH tally from an HDF5 file.

`from_group_v1(group, shared)` Backward compatibility: build tallies from older HDF5 files.

`from_group_v2(group, shared)`

`from_group_v3(group, shared)`

`from_group_v4(group, shared)` Build results for a single MESH tally from an HDF5 file.

classmethod `from_group(group, shared)`

Build results for a single MESH tally from an HDF5 file.

classmethod `from_group_v1(group, shared)`

Backward compatibility: build tallies from older HDF5 files.

classmethod `from_group_v4` (*group, shared*)

Build results for a single MESH tally from an HDF5 file.

`omnibus.postprocess.meshtally.load(*args, **kwargs)`

Get a list of MeshTallyResult objects from an HDF5 file.

Each object in this list corresponds to a single [TALLY][MESH] entry given by the user. It may have several multipliers, each of which has x/y/z/e or x/y/z data.

17.13 omnibus.postprocess.pathlength module

17.14 omnibus.postprocess.rst module

class `omnibus.postprocess.rst.RstBlockType`

Bases: `type`

Metaclass used for easily generating RST writers for each block.

Methods

<code>__call__(...) <==> x(...)</code>	
<code>mro(() -> list)</code>	return a type's method resolution order

17.15 omnibus.postprocess.sensitivity module

17.16 omnibus.postprocess.tally module

This module contains base classes for tally postprocessing.

class `omnibus.postprocess.tally.BinAxis` (*args, **kwargs)

Bases: `omnibus.container.Container`

Metadata about an axis of a tally for multigroup data.

Attributes

`shape`

Methods

`describe_index(i)`

`index(value)`

class `omnibus.postprocess.tally.BinBounds` (*args, **kwargs)

Bases: `omnibus.container.Container`

Bin boundaries when tallying over energy or particle.

This includes energy bounds for all particle types. If no boundaries are present, `bool(bb)` will return `False`.

Examples

Since Omnibus tallies contain both neutron and photon data, one common requirement is to view tally data from a single particle. For this purpose, the `BinBounds` class contains a `slice_for_particle` method, which returns a Python `slice` object corresponding to a single particle type. For example, you can take an energy-dependent tally in a single cell and plot only the neutron fluxes:

```
>>> bin_bounds = tally.bin_bounds
>>> slc = bin_bounds.slice_for_particle('n')
>>> plt.plot(bin_bounds.lower_bounds[slc], tally.mean[slc],
...           drawtype='steps-lower')
```

Attributes

<code>n</code>	(array or None) If the particle is being tallied, neutron bin boundaries in descending order. If not being tallied, this is None.
<code>p</code>	(array or None) If the particle is being tallied, photon bin boundaries in descending order. If not being tallied, this is None.

Methods

<code>describe_group(g)</code>	
<code>from_group(group)</code>	Create a <code>BinBounds</code> from an HDF5 group.
<code>labeled_groups([particle])</code>	Particle and relative group labels for each stored bin.
<code>num_groups(particle)</code>	Return number of groups for a given particle.
<code>slice_for_particle(particle_type)</code>	Return a slice object that will give the range of absolute group indices for a particle.

`bounds`

Returns the tuple of group bounds (neutron, photon)

classmethod `from_group(group)`

Create a `BinBounds` from an HDF5 group.

Returns BinBounds object if bin boundaries are present, else None.

`labeled_groups(particle=None)`

Particle and relative group labels for each stored bin.

This generator method yields a tuple (particle type, relative group) for all groups of particle type `particle` (or both if particle is None).

Parameters `particle` : int

Particle type (default None for both particles), one of NEUTRON or PHOTON.

`lower_bounds`

Get an array of lower energy bounds for all groups.

This is primarily useful when both photon and neutron groups are present, where the lower/upper bounds have a discontinuity.

num_groups (*particle*)

Return number of groups for a given particle.

num_total_groups

The number of groups summed over all particles.

slice_for_particle (*particle_type*)

Return a slice object that will give the range of absolute group indices for a particle.

Use this to extract subsets of collapsed multigroup data, or use the “start” and “stop” attributes of the slice object if you need those.

upper_bounds

Get an array of upper energy bounds for all groups.

This is primarily useful when both photon and neutron groups are present, where the lower/upper bounds have a discontinuity.

class omnibus.postprocess.tally.**DifferenceField** (*args, **kwargs)

Bases: *omnibus.postprocess.field.Field*

Calculate the differences between two tallies.

This field stores the difference (relative error) between two tallies, and propagates the uncertainty between them.

The difference saved is:

$$\text{diff} = \frac{\text{mean}}{\text{mean}_{\text{ref}}} - 1$$

and the variance is:

$$\sigma_{\text{diff}}^2 = \left[\frac{\text{mean}}{\text{mean}_{\text{ref}}} \right]^2 \left[\frac{\sigma^2}{\text{mean}^2} + \frac{\sigma_{\text{ref}}^2}{\text{mean}_{\text{ref}}} \right]$$

It accepts a threshold value for the reference data comparisons. Expected data points with a relative error greater than or equal to this value will be ignored.

Attributes

delta	(array) The relative difference between the two given tallies.
var	(array) The variances corresponding to the means.
axes	(list of DataAxis) Attributes about each dimension/axis of the data.
hyper-slice	(list of (axis, index) tuples.) If this is a view into larger tally data, the hyperslice is the list of axes and values at which those dimensions are being evaluated.

Methods

<code>asciiplot_hist()</code>	Plot a histogram of (differences / errors)
<code>from_tallies</code> (reference, given[, re_threshold])	Build from two tally fields.

asciiplot_hist()

Plot a histogram of (differences / errors)

classmethod from_tallies (*reference, given, re_threshold=0.707*)

Build from two tally fields.

class omnibus.postprocess.tally.**SharedTallyData**(*args, **kwargs)
Bases: omnibus.container.Container

Data shared by all tallies in a file.

This is mostly used in construction to pass problem metadata etc. between tallies. It may include data (such as tally_name) that are changed depending on what tally is currently being processed.

Methods

from_group(root)

class omnibus.postprocess.tally.**Tallies**(*args, **kwargs)
Bases: omnibus.container.Container

Collection of all tallies present in a file.

Attributes

metadata :	Information about the run that created this tally data.
------------	---

Methods

from_group(root) Build results for all tallies in a “tallies” group.
iteritems()
iterkeys()
itervalues()

classmethod **from_group**(root)

Build results for all tallies in a “tallies” group.

class omnibus.postprocess.tally.**TallyField**(*args, **kwargs)
Bases: omnibus.postprocess.field.Field

Multi-dimensional field for storing and accessing bulk tally data.

This provides accessors for retrieving the originally stored mean and variance, as well as calculation methods for returning relative error. Additionally, in the case of multi-dimensional data, it allows advanced slicing of the data.

Attributes

mean	(array (or float)) The mean values for this tally.
var	(array (or float)) The variances corresponding to the means.
axes	(list of DataAxis) Attributes about each dimension/axis of the data.
hyper-slice	(list of (axis, index) tuples.) If this is a view into larger tally data, the hyperslice is the list of axes and values at which those dimensions are being evaluated.

Methods

`from_group(group, name)`
`without_last_energy_bin()` Return a new tally field with the last energy bin removed.

bin_bounds

If this is an energy-binned tally, return the bin bounds.

If not energy-binned (or a slice on the energy axis has already been taken), this will raise a `KeyError`.

re

Calculate relative error, returning inf if mean is zero.

This will return an array if the stored mean and variance are arrays.

$$\text{re} = \frac{\sqrt{\text{var}}}{\text{mean}}$$

without_last_energy_bin()

Return a new tally field with the last energy bin removed.

This is useful for getting Shift, Monaco, and MCNP tallies to match (because the latter two implicitly add a cutoff-to-lowest-energy bin).

If not energy-binned (or a slice on the energy axis has already been taken), this will raise a `KeyError`.

class `omnibus.postprocess.tally.TallyMultiplierResult (*args, **kwargs)`

Bases: `omnibus.container.Container`

Tally results for a single tally multiplier.

This contains the final values of multigroup and total data.

Attributes

<code>name</code>	(string) Short label of this multiplier.
<code>description</code>	(string) Longer description of the multiplier.
<code>bin_bounds</code>	(<code>BinBounds</code>) Bin boundaries for multigroup tallies, or <code>None</code> for tallies without binning.
<code>total</code>	(<code>TallyField</code>) Energy- and particle-integrated means and variances for this multiplier.
<code>binned</code>	(<code>TallyField</code>) Energy-binned means and variances for this multiplier.

has_energy_bins

Whether this multiplier has energy-binned tallies.

class `omnibus.postprocess.tally.TallyResult (*args, **kwargs)`

Bases: `omnibus.container.Container`

Set of results from a single tally.

This contains `TallyMultiplierResult` objects, which contain the data for each multiplier.

Attributes

`has_energy_bins` Whether this multiplier has energy-binned tallies.

name	(string) Tally name.
description	(string) User-specified description; defaults to value of name.
total	(TallyField) Energy-integrated tallies.
binned	(TallyField) Energy-binned tallies.
bin_bounds	(<i>BinBounds</i>) Bin boundaries if binned tallies are present. Evaluates as False if no boundaries are present.
tallies	(dict) Multiplier results for this tally, keyed by short name.
num_histories	(int or TallyField) Number of active histories used for this tally. If this is a depletion tally, it's an array of the number of histories per step.
metadata	(dict) Execution metadata associated with this tally: problem ID, etc.

Methods

<code>at_step(step)</code>	Return a view of the tally data at a single depletion time step.
<code>from_group(group, shared[, reorder])</code>	Build results for a single tally from an HDF5 group.

`at_step(step)`

Return a view of the tally data at a single depletion time step.

If depletion is disabled, this will raise an error.

Returns Tally (or subclass): tally result evaluated at the given time step.

`classmethod from_group(group, shared, reorder=None)`

Build results for a single tally from an HDF5 group.

It uses slices of the multi-tally data to create multiplier results.

`has_energy_bins`

Whether this multiplier has energy-binned tallies.

`multipliers`

Get a list of multipliers used in this tally.

`num_multipliers`

Number of multipliers.

`omnibus.postprocess.tally.get_tally_version(metadata)`

Return the tally compatibility version number.

This version number is only used internally and is not exposed to the user.

17.17 omnibus.postprocess.utils module

Common functions for postprocessing.

`exception omnibus.postprocess.utils.NoDataException`

Bases: exceptions.Exception

An exception class used by postprocessing.

This indicates that no suitable data is present in an output file, but that this is not an abnormal condition.

`class omnibus.postprocess.utils.PostProcessOutput`

Bases: object

Abstract base class for postprocess outputters.

A postprocessor output should raise a NoDataException in the constructor if inapplicable. It should have a “description” class attribute to provide a useful message if postprocessing fails. It should also have a “block” attribute for determining where in the RST output it should be called.

If postprocessing succeeds, the object will be retained so that it can provide more informative output to the user at the end of the run.

Attributes

outputs

block	
description	

Methods

__call__(manager)
write_rst(rstwriter)

`omnibus.postprocess.utils.extract(data)`

Extract data from h5py data objects.

`omnibus.postprocess.utils.group_to_dict(h5_root)`

Utility function for converting HDF5 files to dictionaries.

This is useful for creating unit tests that will run on systems without HDF5 present.

`omnibus.postprocess.utils.h5dump(path)`

Print an HDF5 file as a nicely formatted string.

`omnibus.postprocess.utils.load_from_h5(handle, Data_cls)`

Load standard-format replicated HDF5 data from a file.

This prints out appropriate metadata while loading.

`omnibus.postprocess.utils.load_metadata(md)`

Load metadata from the ‘metadata’ group of an HDF5 file.

The resulting dictionary contains version info,

`omnibus.postprocess.utils.to_enum_array(enum_dict)`

Convert a dictionary of string->value mappings to a value->string array.

17.18 Module contents

The user-accessible package for analysis of Omnibus output.

This postprocessing package provides utility functions and interfaces for processing data from an Omnibus run.

Part IV

User Guide: Insilico

CHAPTER
EIGHTEEN

INTRODUCTION

The Insilico package provides a front-end (**neutronics**) and libraries for running reactor applications using CASL's VERA input specification. This specification is documented in VERAInExt/verain/docs/verain_UM.pdf. The VERAInExt repository also contains the VERA input pre-processor. This repository is available through [CASL](#). An example VERA input for a 17×17 PWR assembly is as follows:

```
1 [CASEID]
2   title 'CASL AMA Problem 3a - Single 17x17 Assembly - Public'
3
4 [STATE]
5   power 0.0          ! %
6   tinlet 620.33      ! F - 600K
7   tfuel 600.0         ! K
8   boron 1300          ! ppmB
9   modden 0.743        ! g/cc
10  sym qtr
11  feedback off
12
13 [CORE]
14  size 1              ! one assembly
15  rated 17.67 0.6823  ! MW, Mlbs/hr
16  apitch 21.5
17  height 406.337
18
19 core_shape
20   1
21
22 assm_map
23   ASSY
24
25 lower_plate ss 5.0 0.5  ! mat, thickness, vol frac
26 upper_plate ss 7.6 0.5
27 lower_ref mod 20.0 1.0
28
29 bc_rad reflecting
30
31 mat he    0.0001786
32 mat zirc  6.56 zirc4
33 mat inc   8.19
34 mat ss    8.0
35
36 [ASSEMBLY]
37   title "Westinghouse 17x17"
38   npin 17
39   ppitch 1.26
```

```

40 fuel U31 10.257 94.5 / 3.1
41
42
43 cell 1      0.4096 0.418 0.475 / U31 he zirc
44 cell 3      0.561 0.602 / mod     zirc      ! guide/instrument tube
45 cell 4      0.418 0.475 /      he zirc      ! plenum
46 cell 5      0.475 /          zirc      ! end plug
47
48 lattice FUEL
49      3
50      1 1
51      1 1 1
52      3 1 1 3
53      1 1 1 1 1
54      1 1 1 1 1 3
55      3 1 1 3 1 1 1
56      1 1 1 1 1 1 1
57      1 1 1 1 1 1 1 1
58
59 lattice PLEN
60      3
61      4 4
62      4 4 4
63      3 4 4 3
64      4 4 4 4 4
65      4 4 4 4 4 3
66      3 4 4 3 4 4 4
67      4 4 4 4 4 4 4
68      4 4 4 4 4 4 4 4
69
70 lattice PLUG
71      3
72      5 5
73      5 5 5
74      3 5 5 3
75      5 5 5 5 5
76      5 5 5 5 5 3
77      3 5 5 3 5 5 5
78      5 5 5 5 5 5 5 5
79      5 5 5 5 5 5 5 5 5
80
81 axial ASSY 10.281
82      PLUG 11.951
83      FUEL 377.711
84      PLEN 393.711
85      PLUG 395.381
86
87 grid END inc 1017 3.866
88 grid MID zirc 875 3.810
89
90 grid_axial
91      END 13.884
92      MID 75.2
93      MID 127.4
94      MID 179.6
95      MID 231.8
96      MID 284.0
97      MID 336.2

```

```
98      END 388.2
99
100     lower_nozzle  ss 6.053 6250.0 ! mat, height, mass (g)
101     upper_nozzle ss 8.827 6250.0 ! mat, height, mass (g)
102
103 [EDITS]
104   axial_edit_bounds
105     11.951
106     15.817
107     24.028
108     32.239
109     40.45
110     48.662
111     56.873
112     65.084
113     73.295
114     77.105
115     85.17
116     93.235
117     101.3
118     109.365
119     117.43
120     125.495
121     129.305
122     137.37
123     145.435
124     153.5
125     161.565
126     169.63
127     177.695
128     181.505
129     189.57
130     197.635
131     205.7
132     213.765
133     221.83
134     229.895
135     233.705
136     241.77
137     249.835
138     257.9
139     265.965
140     274.03
141     282.095
142     285.905
143     293.97
144     302.035
145     310.1
146     318.165
147     326.23
148     334.295
149     338.105
150     346.0262
151     353.9474
152     361.8686
153     369.7898
154     377.711
155
```

```

156 [INSILICO]
157   cell_homogenize true
158   eq_set          spn_fv
159   SPN_order      3
160   Pn_order       1
161   tolerance      1e-6
162   dimension      3
163   Pn_correction  true
164
165   mesh           2
166   max_delta_z   1.27
167   num_groups     23
168
169   mat_library    casl_comp_r2.sh5
170   xs_library     lib252_hetbondoneabs-noabssigp
171
172 ! pin_partitioning true
173   num_blocks_i   4
174   num_blocks_j   4
175   num_z_blocks   1
176   num_sets       1
177
178   silo_output    p3
179
180   new_grp_bounds
181     8.2085e+05
182     1.1109e+05
183     5.5308e+03
184     1.8644e+02
185     3.7612e+01
186     3.5379e+01
187     2.7697e+01
188     2.1684e+01
189     2.0397e+01
190     1.5968e+01
191     7.1500e+00
192     6.7000e+00
193     6.3000e+00
194     1.0970e+00
195     1.0450e+00
196     9.5000e-01
197     3.5000e-01
198     2.0600e-01
199     1.0700e-01
200     5.8000e-02
201     2.5000e-02
202     1.0000e-02
203     1.0000e-05

```

In order to execute this problem through any VERA-supported application (including Insilico), it must be processed into an XML file. The VERAInExt **react2xml.pl** script is used to convert the ASCII input file to code-readable XML:

```
$ perl react2xml.pl 3a.inp 3a.xml
```

The XML can be edited directly by the user (you **better** know what you're doing!) as this provides a method for power users to directly affect code control settings that are not normally exposed at the pre-process level.

The XML file can then be fed to any VERA executable. The VERA executable provided by Insilico is **neutronics**.

It does a full neutronics simulation on the specified input (ie. no TH-coupling, etc.). There are VERA drivers that run coupled simulations that are available through [CASL](#).

In order to run **neutronics** do the following:

```
$ mpirun -np 4 ./neutronics -i 3a.xml
```

The neutronics simulation performs the following basic steps:

1. build problem model
2. build problem mesh or geometry
3. load and process cross section
4. run transport
5. produce integrated pin-power and flux output

Insilico currently supports 3 transport options from the Exnihilo packages Denovo and Shift

- Denovo S_N (discrete ordinates)
- Denovo SP_N (simplified spherical harmonics)
- Shift Monte Carlo

The complete [parameter list specification](#) is available internally for power users.

Part V

Appendices

SHIFT ACCEPTANCE TEST DESCRIPTIONS

19.1 Leakage test

These tests compare the leakage out of a 23.7 cm thick spherical shell with an inner radius of 1.3 cm in a void for various coupled neutron-photon problems. An isotropic Cf-252 Watt spectrum neutron spherical source with radius 0.1 cm is located at the center of the shell. An energy-binned tally calculates the neutron and photon flux exiting the sphere in a 1 cm thick spherical shell void region located at distance of 99.9 cm from the center of the problem.

Shift is run in fixed-source mode simulating 1e7 particle histories using continuous-energy and multigroup physics for various nuclides. A Shift reference solution for all of these tests is used to compare that the flux in each bin for each test is within 4 standard deviations of the Shift reference flux.

If the matplotlib module is installed these tests produce plots of the binned flux compared to the Shift reference, Monaco, and MCNP5 results. They also produce a plot of the flux ratios with the Shift reference result and a histogram plot of the standard deviation difference between the Monaco flux result and the test result in each bin.

19.2 Transmission tests

Requirements: numpy and h5py modules

These tests compare the simulated flux in a detector for neutron-only and photon-only streaming problems produced by Shift. The problem consists of a 2 mfp thick slab in a void with an incident mono-energetic beam on the left of the slab. The energy of the beam varies, along with the material number density of the slab to make it 2 mfp thick (the slab has a fixed width of 5 cm). The total flux is tallied in a 2 cm block region to the right of the slab.

The slab for the neutron-only tests consists of a single nuclide with varying number densities. The slab for the photon-only tests consists of a single element with varying number densities.

Shift runs a fixed-source problem with 1e6 particle histories for each test. Currently, a Shift reference result for the flux in the detector region has been generated in serial for each test and these values are compared to the result from each test. The flux should lie within 3 standard deviations of the reference Shift results regardless of the number of processors the problem is run.

19.3 Material scaling test

DENOVO ACCEPTANCE TEST DESCRIPTIONS

20.1 Adjoint SN transport test

These tests run adjoint Denovo SN transport problems and test the accuracy of this calculation.

20.2 Deterministic first-collision test

20.3 Symmetry test

These tests do things for symmetric problems for the Denovo deterministic transport solvers.

20.4 Two-dimensional transport test

These tests consist of 2D configurations that test the accuracy of the 2D deterministic solvers available in Denovo. The flux in each group and mesh cell is compared to a reference solution generated by Denovo.

20.5 AMPX plotting and XS generation

This set of python script and data shows how the AMPX test library was generated for the whole set of Denovo acceptance tests. The python script needs to be updated for changes elsewhere in Denovo.

CHAPTER
TWENTYONE

STYLE GUIDE

This chapter gives some grammar, spelling, and abbreviation standards to adopt when putting together any sort of publication, presentation, or poster related to Exnihilo.

21.1 Code/Package/Library name capitalization

First, use the following capitalization when referring to code, package, and library names (in alphabetical order for easy reference).

Table 21.1: Capitalization Standards of Entities

Exnihilo pack-age	Code	Library	Package	Program	Machine
Denovo	ADVANTG	BLAS	Atlas	CASL	Remus
Exnihilo	AMPX	LAPACK	Kokkos	VERA	Romulus
Insilico	DagMC	SuperLU	ORIGEN	OLCF	Summit
Nemesis	KENO	TriBITS	XSPROC	OIC	Titan
Omnibus	Lava				
Shift	MCNP				
Transcore	MPACT				
	OpenMC				
	Profugus				
	SCALE				
	Trilinos				
	VESTA				

For a list and capitalization standard of all Trilinos packages, please see its online documentation:
<http://trilinos.org/packages/>

21.2 Commonly used abbreviations/acronyms

Second, the following table gives abbreviations and acronyms we commonly use when referring to terms, methods, or packages.

Table 21.2: Common Abbreviations

Abbreviation	What does it mean
API	Application Programming Interface
CSV	Comma-Separated Value
CADIS	Consistent Adjoint-Driven Importance Sampling
CDF	Cumulative Distribution Function
CPU	Central Processing Unit
ENDF	Evaluated Nuclear Data Files
GMRES	Generalized Minimum Residual
GPU	General Processing Unit
HDF5	Hierarchical Data Format 5
HPC	High-Performance Computing
HZP	Hot Zero Power
KERMA	Kinetic Energy Released per Unit Mass
LWR	Light Water Reactor
MC	Monte Carlo
MG	Multigroup
MOC	Method of Characteristics
MSOD	Multiple-Set Overlapping Domain
PWR	Pressurized Water Reactor
RMS	Root Mean Squared
RTK	Reactor ToolKit
SCE	SCALE Continuous Energy
SMG	SCALE Multigroup
XML	Extended Markup Language

21.3 Commonly Used Terminology

Third, the following words or phrases are commonly used in our publications and these are the standards we have adopted when using them.

Table 21.3: Common Abbreviations

Term	Usage	Example
continuous-energy	adjective	I love continuous-energy data.
domain-decomposed, domain-replicated	adjective	We support domain-decomposed and domain-replicated geometries.
front end	noun	The Omnibus front end is awesome!
high-performance	adjective	Use high-performance computers.
intra-set, intra-block	adjective	Our code uses intra-set and intra-block communication.
massively parallel	adjective	We have a massively parallel code.
multigroup	adjective	I also love multigroup data.
multiset	adjective	Multiset decomposition is best.
object-oriented	adjective	Only use object-oriented languages.
path length	noun	Tallying path length can be hard.
pincell	noun	Reactor assemblies have pincells.
preprocess , postprocess	adjective, verb	We can use Omnibus to preprocess and postprocess results.
run time	noun	Parameters are set at run time.

CHAPTER
TWENTYTWO

LICENSE INFORMATION

The following licenses may apply to code distributed with Exnihilo.

22.1 Trilinos

BSD code in the Trilinos library licensed by Sandia contains the following notice:

Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive license for use of this work by or on behalf of the U.S. Government.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the Corporation nor the names of the contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY SANDIA CORPORATION "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL SANDIA CORPORATION OR THE CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

22.2 Google Test

Google Test is a test harness used by Exnihilo.

Copyright 2008, Google Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER
TWENTYTHREE

ACKNOWLEDGMENTS

23.1 Copyright

Notice: This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC0500OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for the United States Government purposes.

O

omnibus, ??
 omnibus.container, ??
 omnibus.converters, 158
 omnibus.converters.mctal, 153
 omnibus.converters.meshtal, 156
 omnibus.converters.monaco, 157
 omnibus.converters.opus, 158
 omnibus.converters.triton, 158
 omnibus.converters.vesta, 158
 omnibus.documentation, ??
 omnibus.exceptions, ??
 omnibus.lexer, ??
 omnibus.lexer.build, ??
 omnibus.lexer.preprocess, ??
 omnibus.lexer.tokenize, ??
 omnibus.logger, ??
 omnibus.mcnp, ??
 omnibus.omn, ??
 omnibus.omn.composition, ??
 omnibus.omn.denovo, ??
 omnibus.omn.depletion, ??
 omnibus.omn.geometry, ??
 omnibus.omn.manualww, ??
 omnibus.omn.physics, ??
 omnibus.omn.problem, ??
 omnibus.omn.response, ??
 omnibus.omn.root, ??
 omnibus.omn.run, ??
 omnibus.omn.shift, ??
 omnibus.omn.source, ??
 omnibus.omn.tally, ??
 omnibus.parser, ??
 omnibus.parser.abstract, ??
 omnibus.parser.applicability, ??
 omnibus.parser.commands, ??
 omnibus.parser.database, ??
 omnibus.parser.db_builder, ??
 omnibus.parser.defaulter, ??
 omnibus.parser.listvalidator, ??
 omnibus.parser.parameter, ??
 omnibus.parser.postprocessor, ??
 omnibus.parser.subdbrelation, ??
 omnibus.parser.validator, ??
 omnibus.postprocess, 185
 omnibus.postprocess.celltally, 159
 omnibus.postprocess.collisions, 162
 omnibus.postprocess.compositions, 163
 omnibus.postprocess.cyltally, 164
 omnibus.postprocess.depletion, 166
 omnibus.postprocess.field, 169
 omnibus.postprocess.fissionsite, 172
 omnibus.postprocess.history, 173
 omnibus.postprocess.kcode, 174
 omnibus.postprocess.manager, 176
 omnibus.postprocess.meshtally, 177
 omnibus.postprocess.pathlength, 179
 omnibus.postprocess.rst, 179
 omnibus.postprocess.sensitivity, 179
 omnibus.postprocess.tally, 179
 omnibus.postprocess.utils, 184
 omnibus.preprocess, ??
 omnibus.preprocess.mesh_problem, ??
 omnibus.rstwriter, ??
 omnibus.scale, ??
 omnibus.scripts, ??
 omnibus.scripts.omnibus_post, ??
 omnibus.scripts.omnibus_pre, ??
 omnibus.scripts.omnibus_run, ??
 omnibus.scripts.read_nuclides, ??
 omnibus.scripts.vacuum_omnibus_input,
 ??
 omnibus.scripts.xml_to_omn, ??
 omnibus.symmetry, ??
 omnibus.teedprocess, ??
 omnibus.testing, ??
 omnibus.testing.regression, ??
 omnibus.testing.unittest, ??
 omnibus.teuchos, ??
 omnibus.teuchos.reader, ??
 omnibus.teuchos.writer, ??
 omnibus.textfile, ??
 omnibus.timer, ??
 omnibus.utils, ??

```
omnibus.writers,??  
omnibus.writers.ascii,??  
omnibus.writers.json,??  
omnibus.writers.teuchos,??  
omnibus.writers.yaml,??  
omnibus.yamlload,??
```

Symbols

-b dir
 command line option, 25
 -c dir
 command line option, 25
 -m
 command line option, 25
 -n
 command line option, 25
 -p dir
 command line option, 25
 -s sysname
 command line option, 25
 -t variant
 command line option, 25
 -u
 command line option, 26

A

as_dataframe() (omnibus.postprocess.depletion.CrossSections
 method), 166
 as_dataframe() (omnibus.postprocess.field.Field method),
 170
 as_dataframe() (omnibus.postprocess.history.Events
 method), 173
 asciiplot_hist() (omnibus.postprocess.tally.DifferenceField
 method), 181
 assertDataEqual() (TestCase method), 44
 assertSoftEquiv() (TestCase method), 45
 assertTextEqual() (TestCase method), 44
 assertXmlFilesEqual() (TestCase method), 45
 at_step() (omnibus.postprocess.depletion.DepletionTally
 method), 167
 at_step() (omnibus.postprocess.kcode.Keff method), 175
 at_step() (omnibus.postprocess.tally.TallyResult method),
 184
 axis() (omnibus.postprocess.field.Field method), 170
 axis_index() (omnibus.postprocess.field.Field method),
 170

B

basename (omnibus.postprocess.manager.Manager

attribute), 176
 bin_bounds (omnibus.converters.mctal.McnpTallyField
 attribute), 155
 bin_bounds (omnibus.postprocess.tally.TallyField attribute), 183
 BinAxis (class in omnibus.postprocess.tally), 179
 BinBounds (class in omnibus.postprocess.tally), 179
 binned (omnibus.postprocess.celltally.SingleTallyContainer
 attribute), 161
 bounds (omnibus.postprocess.tally.BinBounds attribute),
 180
 build_axes() (in module omnibus.postprocess.field), 172

C

cell_shape (omnibus.postprocess.cyltally.CylMesh
 attribute), 164
 cell_shape (omnibus.postprocess.mesh tally.Mesh attribute), 178
 cells (omnibus.converters.mctal.McnpTallyField attribute), 155
 CellTallies (class in omnibus.postprocess.celltally), 159
 CellTallyMultiplierResult (class in omnibus.postprocess.celltally), 159
 CellTallyResult (class in omnibus.postprocess.celltally), 159
 centers (omnibus.postprocess.field.DataAxis attribute),
 169
 collapse_nuclides() (omnibus.postprocess.depletion.NumberDensities
 method), 168
 CollisionDiagnostic (class in omnibus.postprocess.collisions), 162
 command line option
 -b dir, 25
 -c dir, 25
 -m, 25
 -n, 25
 -p dir, 25
 -s sysname, 25
 -t variant, 25
 -u, 26
 Composition (class in omnibus.postprocess.compositions), 163

Comps (class in omnibus.postprocess.compositions), 163
convert_to_time() (omnibus.postprocess.depletion.DepletionTally method), 167
CrossSections (class in omnibus.postprocess.depletion), 166
cwd (omnibus.postprocess.manager.Manager attribute), 176
CylMesh (class in omnibus.postprocess.cyltally), 164
CylTallies (class in omnibus.postprocess.cyltally), 164
CylTallyMultiplierResult (class in omnibus.postprocess.cyltally), 165
CylTallyResult (class in omnibus.postprocess.cyltally), 165

D

DataAxis (class in omnibus.postprocess.field), 169
db (omnibus.postprocess.manager.Manager attribute), 176
DepletionField (class in omnibus.postprocess.depletion), 166
DepletionTally (class in omnibus.postprocess.depletion), 167
describe_index() (omnibus.postprocess.field.Field method), 171
descriptions (omnibus.postprocess.manager.Manager attribute), 176
DifferenceField (class in omnibus.postprocess.tally), 181
dims (omnibus.postprocess.field.Field attribute), 171
dirname (omnibus.postprocess.manager.Manager attribute), 177
dump_csv() (in module omnibus.postprocess.celltally), 161

E

energy_bins (omnibus.converters.meshtal.McnpMeshTallyResult attribute), 157
Entropy (class in omnibus.postprocess.kcode), 174
Events (class in omnibus.postprocess.history), 173
EXPECT_SOFT_EQ (C macro), 42
EXPECT_SOFT_EQ (C macro), 42
EXPECT_VEC_EQ (C macro), 42
EXPECT_VEC_SOFT_EQ (C macro), 43
extract() (in module omnibus.postprocess.utils), 185
extract() (omnibus.postprocess.collisions.CollisionDiagnostic method), 162

F

Field (class in omnibus.postprocess.field), 170
filename (omnibus.postprocess.manager.Manager attribute), 177
finalize() (omnibus.postprocess.manager.Manager method), 177

FissionSites (class in omnibus.postprocess.fissionsite), 172
from_group() (omnibus.postprocess.celltally.CellTallyResult class method), 160
from_group() (omnibus.postprocess.compositions.Composition class method), 163
from_group() (omnibus.postprocess.compositions.Comps class method), 164
from_group() (omnibus.postprocess.cyltally.CylTallyResult class method), 165
from_group() (omnibus.postprocess.depletion.DepletionTally class method), 167
from_group() (omnibus.postprocess.history.Events class method), 173
from_group() (omnibus.postprocess.kcode.Spatial class method), 175
from_group() (omnibus.postprocess.meshtally.MeshTallyResult class method), 178
from_group() (omnibus.postprocess.tally.BinBounds class method), 180
from_group() (omnibus.postprocess.tally.Tallies class method), 182
from_group() (omnibus.postprocess.tally.TallyResult class method), 184
from_group_v1() (omnibus.postprocess.celltally.CellTallyResult class method), 160
from_group_v1() (omnibus.postprocess.cyltally.CylTallyResult class method), 165
from_group_v1() (omnibus.postprocess.meshtally.MeshTallyResult class method), 178
from_group_v2() (omnibus.postprocess.celltally.CellTallyResult class method), 160
from_group_v3() (omnibus.postprocess.celltally.CellTallyResult class method), 160
from_group_v4() (omnibus.postprocess.celltally.CellTallyResult class method), 160
from_group_v4() (omnibus.postprocess.cyltally.CylTallyResult class method), 165
from_group_v4() (omnibus.postprocess.meshtally.MeshTallyResult class method), 179
from_json() (omnibus.postprocess.manager.Manager class method), 177
from_manager() (omnibus.postprocess.kcode.Entropy class method), 174
from_manager() (omnibus.postprocess.kcode.KcodeTallies class method), 174
from_manager() (omnibus.postprocess.kcode.Keff class method), 175
from_tallies() (omnibus.postprocess.tally.DifferenceField class method), 181
from_xml() (omnibus.postprocess.manager.Manager class method), 177

G

get_tally_version() (in module omnibus.postprocess.tally), 184
 get_timing() (omnibus.postprocess.manager.Manager method), 177
 group_to_dict() (in module omnibus.postprocess.utils), 185

H

h5dump() (in module omnibus.postprocess.utils), 185
 has_energy_bins (omnibus.postprocess.tally.TallyMultiplier attribute), 183
 has_energy_bins (omnibus.postprocess.tally.TallyResult attribute), 184

I

index() (omnibus.converters.mctal.McnpAxis method), 154
 index() (omnibus.postprocess.field.DataAxis method), 169
 integrate() (omnibus.postprocess.collisions.CollisionDiagnostic method), 162

J

JsonWriter (class in omnibus.postprocess.manager), 176

K

Kcode (class in omnibus.converters.mctal), 153
 KcodeTallies (class in omnibus.postprocess.kcode), 174
 Keff (class in omnibus.postprocess.kcode), 174
 keff (omnibus.converters.metal.Kcode attribute), 154
 keff_fom (omnibus.converters.mctal.Kcode attribute), 154
 keff_stdev (omnibus.converters.mctal.Kcode attribute), 154

L

label (omnibus.postprocess.celltally.SingleTallyContainer attribute), 161
 LabelAxis (class in omnibus.postprocess.field), 171
 labeled_groups() (omnibus.postprocess.tally.BinBounds method), 180
 load() (in module omnibus.converters.mctal), 156
 load() (in module omnibus.converters.meshtal), 157
 load() (in module omnibus.converters.monaco), 157
 load() (in module omnibus.converters.opus), 158
 load() (in module omnibus.converters.triton), 158
 load() (in module omnibus.converters.vesta), 158
 load() (in module omnibus.postprocess.celltally), 162
 load() (in module omnibus.postprocess.collisions), 163
 load() (in module omnibus.postprocess.compositions), 164
 load() (in module omnibus.postprocess.cyltally), 165

load() (in module omnibus.postprocess.depletion), 169
 load() (in module omnibus.postprocess.fissionsite), 173
 load() (in module omnibus.postprocess.history), 174
 load() (in module omnibus.postprocess.meshtally), 179
 load() (omnibus.converters.mctal.Reader method), 156
 load_from_h5() (in module omnibus.postprocess.utils), 185

load_from_manager() (in module omnibus.postprocess.celltally), 162
 load_from_manager() (in module omnibus.postprocess.collisions), 163
 load_from_manager() (in module omnibus.postprocess.compositions), 164
 load_from_manager() (in module omnibus.postprocess.cyltally), 165
 load_from_manager() (in module omnibus.postprocess.depletion), 169
 load_from_manager() (in module omnibus.postprocess.fissionsite), 173
 load_from_manager() (in module omnibus.postprocess.history), 174
 load_from_manager() (in module omnibus.postprocess.kcode), 175
 load_metadata() (in module omnibus.postprocess.utils), 185
 location (omnibus.postprocess.field.Field attribute), 171
 lower_bounds (omnibus.postprocess.tally.BinBounds attribute), 180

M

Manager (class in omnibus.postprocess.manager), 176
 material_indexing (omnibus.postprocess.collisions.CollisionDiagnostic attribute), 162
 McnpAxis (class in omnibus.converters.mctal), 154
 McnpCellTallies (class in omnibus.converters.mctal), 154
 McnpCellTallyResult (class in omnibus.converters.mctal), 154
 McnpMeshTallies (class in omnibus.converters.meshtal), 156
 McnpMeshTallyResult (class in omnibus.converters.meshtal), 156
 McnpTallyField (class in omnibus.converters.mctal), 155
 Mesh (class in omnibus.postprocess.meshtally), 177
 MeshTallies (class in omnibus.postprocess.meshtally), 178
 MeshTallyMultiplierResult (class in omnibus.postprocess.meshtally), 178
 MeshTallyResult (class in omnibus.postprocess.meshtally), 178
 mixtures (omnibus.postprocess.manager.Manager attribute), 177
 MonacoCellTallyResult (class in omnibus.converters.monaco), 157

multipliers (omnibus.postprocess.tally.TallyResult attribute), 184

N

NoDataException, 184

num_groups() (omnibus.postprocess.tally.BinBounds method), 180

num_histories_per_cycle (omnibus.converters.mctal.Kcode attribute), 154

num_multipliers (omnibus.postprocess.tally.TallyResult attribute), 184

num_total_groups (omnibus.postprocess.tally.BinBounds attribute), 181

num_unions (omnibus.postprocess.celltally.CellTallyMultiplier attribute), 159

num_unions (omnibus.postprocess.celltally.CellTallyResult attribute), 160

NumberDensities (class in omnibus.postprocess.depletion), 168

O

omnibus.converters (module), 158

omnibus.converters.mctal (module), 153

omnibus.converters.meshthal (module), 156

omnibus.converters.monaco (module), 157

omnibus.converters.opus (module), 158

omnibus.converters.triton (module), 158

omnibus.converters.vesta (module), 158

omnibus.postprocess (module), 185

omnibus.postprocess.celltally (module), 159

omnibus.postprocess.collisions (module), 162

omnibus.postprocess.compositions (module), 163

omnibus.postprocess.cytally (module), 164

omnibus.postprocess.depletion (module), 166

omnibus.postprocess.field (module), 169

omnibus.postprocess.fissionsite (module), 172

omnibus.postprocess.history (module), 173

omnibus.postprocess.kcode (module), 174

omnibus.postprocess.manager (module), 176

omnibus.postprocess.mesh tally (module), 177

omnibus.postprocess.pathlength (module), 179

omnibus.postprocess.rst (module), 179

omnibus.postprocess.sensitivity (module), 179

omnibus.postprocess.tally (module), 179

omnibus.postprocess.utils (module), 184

outputs (omnibus.postprocess.manager.Manager attribute), 177

P

plot() (omnibus.postprocess.kcode.Entropy method), 174

plot() (omnibus.postprocess.kcode.Keff method), 175

plot() (omnibus.postprocess.kcode.Spatial method), 175

postprocessors (omnibus.postprocess.manager.Manager attribute), 177

PostProcessOutput (class in omnibus.postprocess.utils), 184

pp_by_block (omnibus.postprocess.manager.Manager attribute), 177

PRINT_EXPECTED (C macro), 43

process() (omnibus.postprocess.manager.Manager method), 177

R

re (omnibus.postprocess.tally.TallyField attribute), 183

read_values() (omnibus.converters.mctal.Reader method), 156

ReIndex (class in omnibus.converters.mctal), 155

RegionData (class in omnibus.postprocess.depletion), 168

reindex_from_materials() (omnibus.postprocess.collisions.CollisionDiagnostic method), 162

reorder() (omnibus.postprocess.field.Field method), 171

RstBlockType (class in omnibus.postprocess.rst), 179

S

shape (omnibus.converters.mctal.McnpAxis attribute), 154

shape (omnibus.postprocess.field.DataAxis attribute), 169

shape (omnibus.postprocess.field.Field attribute), 171

SharedCellTallyData (class in omnibus.postprocess.celltally), 160

SharedCellTallyMultiplierData (class in omnibus.postprocess.celltally), 161

SharedTallyData (class in omnibus.postprocess.tally), 181

SingleTallyContainer (class in omnibus.postprocess.celltally), 161

Sites (class in omnibus.postprocess.fissionsite), 172

slice_for_particle() (omnibus.postprocess.tally.BinBounds method), 181

Spatial (class in omnibus.postprocess.kcode), 175

StepData (class in omnibus.postprocess.depletion), 169

subset() (omnibus.postprocess.field.Field method), 171

subset_by_index() (omnibus.postprocess.field.Field method), 171

sum_over() (omnibus.postprocess.field.Field method), 171

summarize() (omnibus.postprocess.celltally.CellTallyResult method), 160

summarize() (omnibus.postprocess.collisions.CollisionDiagnostic method), 163

summarize() (omnibus.postprocess.depletion.DepletionTally method), 168

T

Tallies (class in omnibus.postprocess.tally), 182

tallies (omnibus.converters.mctal.Reader attribute), [156](#)
TallyField (class in omnibus.postprocess.tally), [182](#)
TallyMultiplierResult (class in omnibus.postprocess.tally), [183](#)
TallyResult (class in omnibus.postprocess.tally), [183](#)
Tfc (class in omnibus.converters.mctal), [156](#)
to_enum_array() (in module omnibus.postprocess.utils), [185](#)
to_output_path() (omnibus.postprocess.manager.Manager method), [177](#)
total (omnibus.postprocess.celltally.SingleTallyContainer attribute), [161](#)

U

upper_bounds (omnibus.postprocess.tally.BinBounds attribute), [181](#)

V

var (omnibus.converters.mctal.McnpTallyField attribute), [155](#)
version (omnibus.converters.mctal.Reader attribute), [156](#)

W

without_last_energy_bin() (omnibus.postprocess.tally.TallyField method), [183](#)
write_to_csv() (omnibus.postprocess.depletion.CrossSections method), [166](#)
write_to_csv() (omnibus.postprocess.depletion.NumberDensities method), [168](#)
write_to_silo() (omnibus.postprocess.cyltally.CylTallyResult method), [165](#)
write_to_silo() (omnibus.postprocess.fissionsite.FissionSites method), [172](#)
write_to_silo() (omnibus.postprocess.history.Events method), [173](#)

X

xs() (omnibus.postprocess.field.Field method), [171](#)
xs_by_index() (omnibus.postprocess.field.Field method), [171](#)