Updated status of the ART neutronic fast reactor tools integration to the NEAMS Workbench

Updated status of the ART neutronic fast reactor tools integration to the NEAMS Workbench – FY-2018

Nuclear Science and Engineering Division

About Argonne National Laboratory

Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory's main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

DOCUMENT AVAILABILITY

Online Access: U.S. Department of Energy (DOE) reports produced after 1991 and a growing number of pre-1991 documents are available free at OSTI.GOV (<http://www.osti.gov/>), a service of the US Dept. of Energy's Office of Scientific and Technical Information.

Reports not in digital format may be purchased by the public from the National Technical Information Service (NTIS):

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312
www.ntis.gov
Phone: (800) 553-NTIS (6847) or (703) 605-6000
Fax: (703) 605-6900
Email: **orders@ntis.gov**

Reports not in digital format are available to DOE and DOE contractors from the Office of Scientific and Technical Information (OSTI):

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
www.osti.gov
Phone: (865) 576-8401
Fax: (865) 576-5728
Email: **reports@osti.gov**

Disclaimer

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 UChicago Argonne, LLC, nor any of their employees or officers, 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 document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, Argonne National Laboratory, or UChicago Argonne, LLC.

Updated status of the ART neutronic fast reactor tools integration to the NEAMS Workbench

Updated status of the ART neutronic fast reactor tools integration to the NEAMS Workbench – FY-2018

prepared by
N. Stauff, T. Kim
Nuclear Science and Engineering Division
Argonne National Laboratory

September 30, 2018

ABSTRACT

The Workbench initiative was launched in FY-2017 within the Nuclear Energy Advanced Modeling and Simulation (NEAMS) Integration Product Line to facilitate the transition from conventional tools to high-fidelity tools. The Workbench provides a common user interface for model creation, real-time validation, execution, output processing, and visualization for integrated codes. The integration of the Argonne Reactor Computation (ARC) suite of codes into the NEAMS Workbench was initiated in FY-2017. The ARC codes contain both legacy codes like DIF3D and REBUS-3 that were developed with over 30 years of experience, and newer NEAMS additions like MC²-3 and PERSENT.

The ARC integration into the NEAMS Workbench interface relies on the PyARC module which handles the pre- and post-processing of the native ARC codes input, and the runtime environment. The PyARC module together with the NEAMS Workbench interface are both released under Open Source Software licenses. This report describes the ARC capabilities available with the Workbench at the end of FY-2018, and a tutorial is provided together with the automatically-generated code documentation.

Integrating the ARC codes into the Workbench benefits directly the Advanced Reactor Technology (ART) Campaign by providing a set of controlled, maintained and validated scripts to generate ARC inputs, which promotes best practices, reduces the learning curve, and facilitates project collaboration. The second benefit from this project is to improve the user experience with the ARC codes: the Workbench interface provides assistance for building an input through auto-completion, real-time validation, access to templates, and visualization. Finally, the ARC integration into the NEAMS Workbench enables new modeling capabilities, those will be extremely useful for advanced reactor design and analyses. The PyARC module facilitates and automatizes complex calculations and workflows for reactor analysis enabling geometrical perturbations, cross-section update through depletion, etc. The Dakota/PyARC coupling in the Workbench was also demonstrated to enable mathematical optimization and sensitivity analysis/uncertainty quantification (SA/UQ) techniques with ARC neutronic simulations.

For demonstration purposes, the ARC codes were used through the Workbench for solving the Sodium-cooled Fast Reactor Uncertainty Analysis in Modelling (SFR-UAM) benchmark problems. The ARC codes are currently used at ANL through the Workbench by nuclear engineers for SFR core design analyses. Future efforts will focus on continuously adding modeling capabilities available with the ARC codes, training new users to continue building some experience, and on integrating high-fidelity codes such as PROTEUS into the Workbench.

Table of Contents

ABSTRACT	I
TABLE OF CONTENTS	II
LIST OF FIGURES	IV
LIST OF TABLES	V
1. INTRODUCTION	6
2. FRAMEWORK FOR ARC INTEGRATION	7
2.1 THE WORKBENCH INTERFACE	7
2.1.1 Common input	8
2.1.2 Templates	8
2.1.3 Visualization	9
2.2 PYARC MODULE	9
3. CAPABILITIES INTEGRATED	11
3.1 MC ² -3 [5]	11
3.2 DIF3D [7]	12
3.3 REBUS-3 [7]	13
3.4 PERSENT [8]	14
4. APPLICATION TO OECD/NEA BENCHMARK	16
4.1 FULL SFR CORE BENCHMARKS	16
4.2 ASSEMBLY DEPLETION BENCHMARK	17
5. DAKOTA/PYARC COUPLING	19
5.1 WORKFLOW IMPLEMENTED	19
5.2 BENEFITS OF THE DAKOTA/PYARC COUPLING	20
6. CONCLUSIONS AND FUTURE WORK	21
REFERENCES	22
APPENDIX A: TUTORIAL	24
1. GEOMETRY	25
1.1 MATERIALS	25
1.2 BLENDS	26
1.3 SURFACES	27
1.4 REACTOR	29
1.4.1 <i>reactor_hex_lattice and reactor_square_lattice</i>	29
1.4.2 <i>assembly</i>	31
2. CALCULATIONS	33
2.1 MCC3	33
2.2 DIF3D	35

2.2.1	<i>rebus</i>	36
2.2.2	<i>persent</i>	37
3.	EXECUTION AND POST-PROCESSING	39
	APPENDIX B: PYARC DOCUMENTATION	41

LIST OF FIGURES

Figure 2-1. Structure of the ARC integration in the Workbench.	7
Figure 2-2. CMB interface for PyARC model creation.	8
Figure 3-1. Example of multi-group XS plot automatically generated with the Workbench from ISOTXS edit file.	12
Figure 3-2. Example of visualizations available for DIF3D and REBUS-3: neutron flux spectrum (left) and power map (right) calculated and plotted with the Workbench.	13
Figure 3-3. Multi-step depletion procedure implemented in PyARC.	14
Figure 3-4. Equilibrium cross-section iteration procedure implemented in PyARC.	14
Figure 3-5. Sodium void worth distribution [kg^{-1}] calculated and plotted within the Workbench.	15
Figure 4-1. Radial layout of the Large Oxide (left) and Medium Metallic (right) SFR cores.	16
Figure 4-2. Radial layout of the large oxide core assembly.	18
Figure 5-1. Schematic of the Dakota/PyARC coupling.	19
Figure A0-1. Core layout of the SFR model used in this example.	24
Figure A0-2. Structure of the common input.	25
Figure A1-1. Hexagon description with orientation=y.	27
Figure A1-2. Axial layout of the assemblies in the example.	28
Figure A1-3. Radial layout of the fuel assembly and pin in the example.	28
Figure A1-4. Radial layout for the hexagonal lattice.	29
Figure A1-5. Radial layout for the square lattice.	30
Figure A1-6. Core radial layout in the example.	31
Figure A1-7. Fuel assembly pin lattice and pin wire geometry in the example.	33
Figure A2-1. Input example and associated RZ geometry.	35
Figure A2-2. Illustration of time-step equivalence to irradiation calculation performed with REBUS.	37
Figure A3-1. Example of VisIt display and steps for a user to set it up.	40

LIST OF TABLES

Table 4-1. ARC/Workbench results for the UAM-SFR full core benchmark problems. 17

Table 4-2. ARC/Workbench results for the UAM-SFR assembly depletion benchmark
problem. 18

1. Introduction

One of the objectives of the Nuclear Energy Advanced Modeling and Simulation (NEAMS) Integration Product Line (IPL) is to facilitate the deployment of the high-fidelity codes developed within the NEAMS program. The Workbench [1] initiative was launched in FY-2017 by the IPL to facilitate the transition from conventional tools to high fidelity tools [2]. The Workbench provides a common user interface for model creation, real-time validation, execution, output processing, and visualization for integrated codes. The integration of the Argonne Reactor Computation (ARC) suite of codes into the NEAMS Workbench was initiated in FY-2017 [3].

The ARC suite of codes [4] gathers neutronics, thermal hydraulics, safety, and fuel behavior analysis codes. The current focus of the Workbench integration is on the deterministic neutronic codes. It includes MC²-3 [5] for multi-group cross-section processing, DIF3D [6] for flux calculation, REBUS-3 [7] for depletion and equilibrium calculations, and PERSENT [8] for perturbation theory calculations (perturbation, sensitivity and uncertainty quantification). These ARC codes are used at national laboratories, universities, and companies for fast reactor analysis. They gather more than 30 years of development, went through extensive validation and verification, and can solve complex physics phenomena in a very efficient way. However, these codes require knowledge on reactor physics and experience on fast reactor design in order to be familiar with the extent of their capabilities, and users mostly rely on scripts, developed based on their experiences, to generate inputs. Integrating the ARC codes into the NEAMS Workbench was initiated to address these challenges and to improve user experience with these codes by taking advantage of the various benefits brought by the Workbench interface.

The status of the ARC integration [9] is described in this report, focusing on FY-18 developments. The ARC integration framework in the Workbench is introduced in Chapter 2. The status of the ARC capabilities integrated are discussed in Chapter 3. A tutorial explaining the different code options is provided (in Appendix A: Tutorial), together with the automatically-generated code documentation (in Appendix B: PyARC Documentation). The developed capabilities were employed to model the Sodium-cooled Fast Reactor Uncertainty Analysis in Modelling (SFR-UAM) benchmark exercises as described in Chapter 4. The coupling between the Dakota and the ARC codes was enabled within the Workbench and is described in Chapter 5. Finally, Chapter 6 draws the conclusions and discusses future developments.

2. Framework for ARC Integration

Figure 2-1 illustrates how the Workbench interface connects with the ARC codes. This is a “black box” type of integration where the Workbench must rely on an opaque wrapper module (called PyARC) that conducts the native input formatting. One of the benefits of the “black box” type of integration is that the user is shielded from the original input of the legacy codes. There are several components to the integration that are described in this section:

- **Workbench interface:** It is developed at ORNL and several components of this interface are required for a code’s integration:
 - Common input
 - Templates
 - Visualization
- **PyARC module:** this is a python module required for “black box” integration that contains the logic for processing the code’s inputs, running them, and post-processing the outputs. This module is the glue between the Workbench interface and the ARC codes.

Figure 2-1. Structure of the ARC integration in the Workbench.

2.1 The Workbench Interface

The Workbench [1, 10] interface is developed at ORNL and designed to assist new users, while not obstructing experienced ones. The Workbench provides a common user interface for model creation, real-time validation, execution, output processing, and visualization for integrated codes. For instance, the user is guided by the auto-completion capability in the Workbench to build its core model in the “common input” structure.

2.1.1 Common input

The Workbench input format adopted is described as the “common input” since it is used to generate inputs for MC²-3, DIF3D, REBUS-3, and PERSENT for integrated problem-dependent cross-section preparation, core analysis, depletion, and sensitivity/uncertainty quantification. The main benefits of the “common input” strategy is to insure every native input uses consistent information and to facilitate project collaboration (since one PyARC input contains all the problem definition, while the MC²-3, REBUS-3, and PERSENT inputs only contain part of the information).

This “common input” allows modeling a reactor geometry in an intuitive and flexible way and was developed with continuous involvement of ARC users. It uses the Standard Object Notation (SON) [11] format that enables the auto-completion, real-time input validation, and access to templates, through the Workbench interface.

The structure of the input is shown in Figure A0-2 and a tutorial was developed (in Appendix A: Tutorial) to explain in detail the input logic to new users. The common input is defined in the “arc.sch” file that takes ~5000 lines of code. Detailed documentation of all the input options is provided (in Appendix B: PyARC Documentation). The user has direct access to the input keyword definitions through the Workbench user-interface upon auto-completion.

Finally, the computational model builder (CMB) [12] interface is being developed by Kitware and will provide “drag-and-drop” input creation capability and real-time model visualization, as shown in Figure 2-2. Preliminary capabilities were successfully tested in FY-2018 and development will continue in FY-2019.

Figure 2-2. CMB interface for PyARC model creation.

2.1.2 Templates

The Workbench contains the HierarchicAL Input Template Engine (HALITE) developed to expand hierarchical input data into code-specific input. Templates are used to assist users in

generating the common input within the Workbench. The common input templates were developed in parallel to the schema and the common input. Those are blocks of input with default values accessible for convenience to the user. A total of **87 templates** were generated for the ARC codes. Templates are also relied upon by the Dakota/PyARC coupling, as explained in Chapter 5. Templates are not currently used with PyARC for native input generation, but their use will be investigated in the future.

2.1.3 Visualization

The Workbench provides different built-in types of visualization capabilities that the ARC integration benefits from. Added in FY-2018 is a feature to provide visualization of user input problems in real time through built-in input visualization capabilities (see Figure 2-1). This visualization capability supports 2D/3D hexagonal and Cartesian geometries.

The VisIT tool [13] is integrated into the Workbench and allows direct visualization of the ARC post-processed outputs, as illustrated in Figure 2-1. The Workbench interface supports plotting capabilities using line plots, histograms, bar charts, etc. Two types of line plots were implemented to display the multi-group cross sections processed by MC²-3 (as illustrated in Figure 3-1), and the region-wise flux spectrum printed by DIF3D and REBUS-3 (also shown in Figure 3-2).

2.2 PyARC Module

The PyARC module is the glue between the Workbench interface and the ARC codes. It is being released by ANL under an open-source software license. For a “black box” integration, this wrapper is essential as it contains the logic to:

- extract information from the common input entered through the Workbench
- perform additional verifications on the core model that the validation engine of the Workbench cannot perform (~150 checks are performed)
- pre-process the information, calculating for instance homogenized atom densities in different regions
- generate the ARC codes’ native inputs
- handle the runtime environment, which can be very complex. For instance, running MC²-3 elementary cell calculations in parallel on different CPUs to calculate fine-mesh cross-sections, followed by TWODANT to calculate region-wise flux spectrum, then by MC²-3 for broad-mesh condensation of the cross-sections, then by REBUS to calculate depleted compositions, then by PERSENT to calculate neutronic feedback coefficients on depleted core compositions, etc.
- post-process the outputs, printing out summary files gathering the main results of the different codes’ outputs.

The PyARC module gathers ~6,500 lines of Python code developed through a collaborative environment on GitLab so that new additions are tracked and reviewed. The PyARC module relies on the following sub-modules:

- PyARCModel: loads the input, performs list of additional verification, performs pre-processing on the input
- PyARCUtils: contains utilities procedures
- PyARCUserObject: defines variables and procedures that are used throughout the code
- PyMCC3, PyREBUS, PyPERSENT: contain the logic for input writing, execution, and post-processing for each code
- PyRzmfIxCODE: contain the logic for input writing, execution, and post-processing for TWODANT and PARTISN
- The PyARC module also relies on the PySCL module that is developed by ORNL to provide the standard composition library (SCL) in a consistent way between different codes

Unit tests are developed for regression testing after each code modification and prior to committing and pushing modifications on the main branch. Currently, **116 unit tests** are implemented to check the common input processing, interpretation of the standard composition library, input generation of MC²-3, DIF3D, REBUS-3, TwoDant, PARTISN, PERSENT, execution of the codes, and post-processing of the outputs. Consequently, the unit tests check the pre-processing, input writing, execution, and post-processing logic of PyARC.

ORNL also implemented the continuous integration (CI) testing and deployment (CD) bundle infrastructure for the PyARC software. This checks all tests at each 'push' to the code repository and ensures all features are functional on Linux and Mac operating systems, and subsequently bundles the new PyARC version into an easily deployable file.

3. Capabilities Integrated

One of the benefits of the “black box” type of integration adopted with the PyARC module is that the user is shielded from the original input of the legacy codes. The associated challenge is that some of the options and code’s capabilities may not be made available to the user through the “black box”. The ARC integration using such a “black box” approach focuses on the most popular and important capabilities of each code to serve more efficiently most user needs. This section summarizes the status of the ARC codes integration within the NEAMS Workbench completed in FY-2018. Continuing efforts are underway to implement additional modeling capabilities based on the ARC and NEAMS codes.

Upon each calculation, the Workbench generates some output files specific to each code, together with some common files summarizing the calculations performed:

- “.*timeline*”: real-time printout of the timeline of the calculation
- “.*summary*”: summary results from all calculations performed
- “.*inp*”: concatenation of all input files
- “.*out*”: concatenation of all output files
- “.*zip*”: all individual input, output, isotxs, and summary files from the calculation

3.1 MC²-3 [5]

The MC²-3 code is developed within the NEAMS program for multi-group cross-section processing for both fast and thermal spectrum reactors. From the Workbench, one can generate cross-sections for pre-generated or user-defined energy-group structure with different scattering orders. The user can merge cross-sections to define lumped fission products used in REBUS-3. Neutron slowing down equation can be solved over a homogenized cell or over a heterogeneous geometry [14] based on 1D cylindrical or slab geometries as implemented in FY-2018. Newer developments made available in MC²-3 [5] should be implemented in the future, such as the calculation of gamma cross-sections, thermal cross-sections, and the integration of the MOC flux solver for heterogeneous calculations based on 2D and 3D cell geometries.

For region-wise group condensation, two approaches were implemented within the Workbench. The first approach consists in generating neutron leakage files from the fuel regions that can be used as external sources in the non-fuel regions (for instance, reflector). The second approach consists of using TOWDANT [15] or PARTISN [16], those are S_n neutron transport equation solvers, for fine-group (1000 – 2000 groups) flux calculation using an equivalent 2D (RZ or XY) core model. In terms of output processing, the multi-group cross-sections generated in the ISOTXS file can be plotted automatically in the Workbench interface, as illustrated in Figure 3-1.

Upon completion, PyARC returns the TWODANT, PARTISN and MC²-3 inputs and outputs (in the “.*inp*” and “.*out*” files) and the multi-group cross-section files:

- The “.*isotxs*” files are binary files that can be re-used by DIF3D/REBUS/PERSENT to avoid re-running the initial MC²-3 calculation

- The “.isotxs.edit” files are text files containing all the multi-group XS results that can be opened directly with the Workbench to plot automatically the cross-section using the “ISOTXS – ISOTOPE XS” processor (as shown in Figure 3-1).

Different isotxs files may be generated at different depletion steps, the following nomenclature logic is used:

- “.isotxs_R_0”: Reference (un-perturbed), depletion time-step 0 (provided composition or beginning of equilibrium cycle);
- “.isotxs_D_4”: “D” perturbation, depletion time-step 4.

Figure 3-1. Example of multi-group XS plot automatically generated with the Workbench from ISOTXS edit file.

3.2 DIF3D [7]

The DIF3D code is a legacy ARC tool used for neutron and gamma flux calculations on various types of geometries, based on pre-generated multi-group cross-sections. The multi-group cross-sections can be generated using MC²-3 calculations or a compatible set of previously calculated multi-group cross-sections.

The 3D-Hexagonal types of geometries were supported through the Workbench in FY-2017 and FY-2018 work included also 2D-Hexagonal and 2D/3D-Cartesian types of geometries. The DIF3D code includes 3 neutron solvers (Nodal, Finite Difference, and Variant [17]) that were all enabled. Currently, only neutron flux calculations are integrated into the Workbench, while gamma calculations should be implemented in the future.

PyARC returns the full DIF3D input and output (in the “.inp” and “.out” files). Post-processing of DIF3D output was implemented by printing the main information of interest to a user (e.g., the

neutron flux in different core areas, integrated flux and power per assemblies) in the summary file (“*.summary*”). When opening this “*.summary*” file with the Workbench, the user can use the “*flux spectrum*” processing to automatically plot the neutron flux spectrum. Direct visualization of the power density, neutron flux, atom densities, etc., is enabled by opening the generated “*.vtk*” file with VisIT through the Workbench, as also illustrated in Figure 3-2.

Figure 3-2. Example of visualizations available for DIF3D and REBUS-3: neutron flux spectrum (left) and power map (right) calculated and plotted with the Workbench.

3.3 REBUS-3 [7]

REBUS-3 is a legacy ARC code used for fuel cycle analysis using DIF3D solvers. It allows a wide range of fuel cycle modeling options such as assembly shuffling, enrichment or cycle length search at equilibrium state. In terms of post-processing, the same capabilities developed for DIF3D are made available with REBUS-3 at every time-step of the depletion calculation. In particular, the “*.rebus_XX.vtk*” file is generated by REBUS-3 at time XX days. Some specific information are also printed out in the “*.summary*” file such as the peak burnup and fast fluence, the enrichment modification factor, etc.

The main challenge associated with integrating REBUS-3 is to allow a user-specified decay chain, without making the Workbench input too complicated. This was achieved by directly providing an external text file containing the decay chain input from REBUS-3 (cards 09, 24, 25), which is being parsed in PyARC.

The once-through depletion capability was integrated with the original option to re-calculate the multi-group cross-sections at different time-step of the depletion, as illustrated in Figure 3-3. This capability can be especially relevant when modeling very high burnup fuel or a reactor with thermalized neutron flux.

Figure 3-3. Multi-step depletion procedure implemented in PyARC.

The enrichment or cycle length search options at equilibrium state were also integrated, where the user can define reprocessing plants, external feeds, and fuel-cycle strategies. The original option was implemented to iterate between the multi-group cross-sections computed with MC²-3 at beginning of equilibrium cycle, and the equilibrium search calculation performed with REBUS-3, as illustrated in Figure 3-4.

Figure 3-4. Equilibrium cross-section iteration procedure implemented in PyARC.

3.4 PERSENT [8]

PERSENT is a perturbation theory code developed within the NEAMS program and based on the neutron transport equation in a 2D or 3D geometry. It allows calculating reactivity feedback coefficients, sensitivity coefficients [18], and nuclear data uncertainties. Perturbation and sensitivity calculations were implemented on eigenvalue, beta and lambda problems and can be automatically run at different depletion steps (computed with REBUS-3). The user can define which materials are perturbed with a change in density or in temperature or which surfaces are perturbed (only for direct DIF3D perturbations, as explained below). The cross-sections of the perturbed composition can be automatically re-calculated both for perturbation and for sensitivity calculations. The nuclear data uncertainties can be estimated on the eigenvalue, beta, lambda, and on the reactivity coefficients automatically by providing a covariance matrix (in a PERSENT-compatible file format).

Some original features were implemented in FY-2018 to take advantage of the flexible PyARC integration. First, direct DIF3D calculations were enabled as an alternative to PERSENT perturbation theory calculations. This can be especially useful for double checking the perturbation theory result and for modeling geometric perturbations such as the axial and radial feedback coefficients or the control rod worth. Second, to optimize the workflow of PERSENT calculations, one added the option to perform preliminary un-perturbed DIF3D calculations to use generated flux files (adjoint and forward) in different PERSENT runs. Finally, every PERSENT or DIF3D calculations can be run in parallel on different CPUs.

Visualization of the perturbation results is enabled within the Workbench using VisIt [13]: for instance, “*.persent_P_ref.vtk*” is the vtk file generated by PERSENT for perturbation *P*. For

September 30, 2018

illustration purposes, Figure 3-5 shows the distribution of the sodium void worth calculated on an SFR design and plotted by VisIt within the Workbench.

Figure 3-5. Sodium void worth distribution [kg⁻¹] calculated and plotted within the Workbench.

4. Application to OECD/NEA Benchmark

The OECD/NEA sub-group on Uncertainty Analysis in Modelling (UAM) for Design, Operation and Safety Analysis of Sodium-cooled Fast Reactors (SFR-UAM) has been formed under the NSC/WPRS/EGUAM and specified a series of benchmarks [19]. For demonstration purposes of the implemented capabilities available in the Workbench, the fuel assembly depletion benchmark and the full SFR core benchmarks proposed within the SFR UAM [20] were modeled using the ARC codes through the Workbench. The detailed description of the benchmark specifications is available in [21]. All calculations were performed with the ENDF/B-VII.1 nuclear data library [22] and were compared with the previously obtained results using the ARC codes outside the Workbench [20].

4.1 Full SFR Core Benchmarks

Two SFR cores were selected for the UAM-SFR benchmark [20, 21]: those are the large oxide core proposed by CEA and the medium metallic core proposed by ANL. The oxide-fueled SFR core is a large 3600 MWth core that exhibits a low reactivity swing during the equilibrium burn cycle. It uses Oxide Strengthened Steel (ODS) cladding with helium bond and is based on the “fat pin with small wire” concept that enables to reach self-breeding without the use of fertile blanket. The performance of the ODS cladding allows an average burnup around 100 GWd/t_{HM} for a corresponding cycle length of 410 equivalent full power days with one fifth reloading scheme. The radial core layout obtained with the Workbench visualization tool is shown in Figure 4-1. It consists of 453 driver fuel, 330 radial reflector and 33 control assemblies.

Figure 4-1. Radial layout of the Large Oxide (left) and Medium Metallic (right) SFR cores.

The medium size SFR core generates 1000 MWth and uses metallic fuel. It was developed for the study of fast reactor design options under the Global Nuclear Energy Partnership (GNEP) program. It is a compact core concept with a transuranics (TRU) conversion ratio of ~0.7, and a one-year cycle length assuming 90% capacity factor. Conventional or reasonably proven materials were utilized in the ABR core concepts so that the core stays within current fast reactor

technology knowledge base. The radial layout of the 1000 MWth ABR metallic benchmark core is shown in Figure 4-1 and consists of 180 driver fuel, 114 radial reflector, 66 radial shield, and 19 control assemblies.

The results obtained at ANL with the PyARC through the NEAMS Workbench are displayed in Table 4-1 and can be compared with the preliminary results from contributing participants for this core benchmark published in Ref. [20]. In this paper, 6 international institutes participated to the medium metallic core benchmark, and 9 participated to the large oxide core benchmark. The previous results from ANL are relatively similar to the one displayed in Table 4-1, and small discrepancies could be explained by newer version of the code and nuclear data library, improved workflow available within PyARC, and minor inconsistencies that were fixed. One did notice though an inconsistency with the under-estimation of the 5-cm control rod worth, which discrepancy is still under investigation. Apart from this parameter, the Workbench ARC results fit within two standard deviation of the solutions obtained by participants.

The nuclear data uncertainties displayed in Table 4-1 were computed using the COMMARA 2.0 covariance matrix [23]. These are consistent with the nuclear data uncertainties displayed by ANL in Ref. [20] for the medium metallic core. It should be noted that the nuclear data uncertainties coming from geometrical perturbations (on the control rod worth, axial and radial expansions), cannot yet be directly computed through the PyARC module. This capability should be implemented in the future but may require some minor development in PERSENT.

Table 4-1. ARC/Workbench results for the UAM-SFR full core benchmark problems.

	Large Oxide		Medium Metallic	
	Param.	UQ [%]	Param.	UQ [%]
K-effective	1.0138	1.37%	1.0148	1.30%
β_{eff}	[pcm] 349	1.16%	[pcm] 329	1.14%
Control Rod Worth (fully inserted)	-6,150	-	-10,350	-
Control Rod Worth (5cm from top)	-111	-	-240	-
Doppler Constant	-873	4.80%	-398	5.86%
Na Void Worth	1,695	8.25%	1,322	19.67%
	[pcm/K]		[pcm/K]	
1% Sodium	0.455	7.66%	0.428	12.36%
1% Wrapper	0.024	6.60%	0.023	7.90%
1% Cladding	0.036	6.40%	0.040	7.91%
1% Fuel	-0.299	1.87%	-0.556	2.00%
1% Fuel + Axial	-0.124	-	-0.253	-
1% Grid	-0.752	-	-1.155	-

4.2 Assembly Depletion Benchmark

The description of the oxide-fueled assembly model is detailed in [21] and its radial layout is shown on Figure 4-2. Preliminary results from 9 contributing participants for this depletion benchmark were published in [19]. ANL contributed to this benchmark using the ARC codes through the Workbench, with reference calculation route with heterogeneous processing of the

cross-sections from ENDF/B-VII.1, and a detailed decay chain. The depletion is performed with the REBUS-3 code and neutron flux solved with DIF3D finite difference diffusion solver on this infinite lattice geometry. The reference results obtained are shown in Table 4-2. It should be noted that small inconsistency in the flux level results were observed with benchmark participants and are under investigation.

Figure 4-2. Radial layout of the large oxide core assembly.

For additional information, the procedure implemented in PyARC that updates the multi-group XS at each depletion step (shown in Figure 3-3) was tested, and the k-effective results obtained are within 40 pcm of the reference results. No significant impact on other parameters (flux, and reactivity coefficients) were observed, confirming that updating cross-section through burnup is not critical for typical SFR depletion analyses.

Table 4-2. ARC/Workbench results for the UAM-SFR assembly depletion benchmark problem.

Burn-up step [days] (cum.)	0	102.5	205	307.5	410
k-infinite	1.1310	1.1323	1.1332	1.1339	1.1343
<i>k-infinite - XS updated</i>	1.1310	1.1321	1.1329	1.1334	1.1339
Flux [n/cm ² -s]	2.49E+15	2.48E+15	2.48E+15	2.47E+15	2.47E+15
Na void reactivity [pcm]	2935	-	2970	-	3004
Doppler reactivity [pcm]	-621	-	-585	-	-555
Delayed neutron frac. [pcm]	346	-	342	-	338

5. Dakota/PyARC Coupling

The Workbench provides a common user interface for model creation allowing for its integrated codes to communicate and work together with limited coupling development. The feasibility and benefits of using the Workbench as a coupling mechanism between the Dakota [24] code and PyARC was demonstrated in FY-2018. The Dakota software maintained by Sandia National Laboratory is a sensitivity analysis/uncertainty quantification (SA/UQ) and optimization toolkit with over 20 years of supporting development. Dakota provides advanced mathematical methods to vary one code's input parameters and analyze the output results for optimization and uncertainty quantification analyses.

5.1 Workflow Implemented

The workflow in Figure 5-1 was developed to allow Dakota to drive the PyARC calculations within the Workbench interface. For SA/UQ or optimization analyses, the PyARC input is perturbed by a sequence of random values from Dakota. After the ARC runs are performed with different sampled input values, Dakota evaluates the user-specified responses of interest. For SA/UQ types of analyses, Dakota performs statistical analysis on response functions. For optimization problems, it selects best performing solutions and generate new samples.

Figure 5-1. Schematic of the Dakota/PyARC coupling.

Three files are required in this workflow:

- Dakota input: Input built by the analyst with the aid of the Workbench that describes the sensitivity, uncertainty or optimization problem options.
- PyARC “common input”: This is the PyARC input built by the analyst with the aid of the Workbench, but saved in a **template** format where some input values are replaced with variables defined in the Dakota input.
- PyARC.driver: Input built by the analyst with the aid of the Workbench that connects the Dakota input to the PyARC application. It contains the logic to extract (with customizable grep commands or links to post-processing scripts) different results from the ARC output summary file.

5.2 Benefits of the Dakota/PyARC Coupling

The Dakota/ARC coupling would be a significant effort to set up outside of the Workbench since the individual ARC codes use different input logic. This would require to develop a script that propagates the input parameters sampled throughout the different codes, which is effectively done with the PyARC module. The new Dakota driver developed at ORNL further streamlines the coupling process.

This coupling enables new capabilities to ARC users as demonstrated in FY-2018, where Dakota was used for driving sensitivity analysis and uncertainty quantification (SA/UQ) calculations, and core design optimization with the ARC codes.

- Optimization problems

Mathematical optimization methods can be used to investigate a space of input options and find the most promising solutions (usually a compromise between targeted performance). This is especially well suited for advanced reactor design work, where many core options need to be evaluated (size and number of fuel pins, different fuel forms, etc.) to assess their impact on core performance (irradiation testing capabilities, inherent safety performance, etc.). Dakota provides a wide range of advanced optimization methods that can be used to effectively investigate the design space and find the best performing core concepts.

This was demonstrated in FY-2018 [25] where multi-criteria optimization of the Advanced Burner Test Reactor (ABTR) was performed employing the Dakota/PyARC coupling through the Workbench. Different multi-criteria optimization methods based on Genetic Algorithms were evaluated and surrogate models were effectively developed to reduce the computation time from the ARC calculations.

- SA/UQ problems

Dakota extends the SA/UQ capability currently available in the ARC codes (with PERSENT) to propagate the uncertainty on any type of input parameter, and to observe its impact on any output result. In fact, there are different benefits/challenges associated with solving SA/UQ problems through adjoint-based perturbation theory (available with PERSENT [8]) and through stochastic sampling (with Dakota [24]). The adjoint-based method is usually cheaper in terms of computational resources, is well suited to treat a large number of uncertain parameters such as uncertainties on multi-group cross-sections, and can provide detailed information such as the impact of cross-section values in any energy group, in any core location, on different core parameters. It is usually applied to see the uncertainty impact on the eigenvalue, on reactivity effects and on reaction rates. The stochastic sampling method provides a more general approach that can be applicable to any uncertainty problem considered (including those with changes in core geometry), to analyze the impact uncertain parameters may have on any output of the problem. However, it may require many simulations to reach targeted levels of confidence. The benefits of using Dakota to drive PyARC simulations for SA/UQ problems was demonstrated in Ref. [26], where uncertainties coming from some manufacturing tolerance were assessed on SFR using the Latin Hypercube Sampling method (LHS).

6. Conclusions and Future Work

The Workbench initiative was launched in FY-2017 within the NEAMS program to facilitate the transition from conventional tools to high-fidelity tools, employing a bottom-up approach. The Workbench provides a common user interface for model creation, real-time validation, execution, output processing, and visualization for integrated codes.

This report details the efforts under way for integrating the Argonne Reactor Computation (ARC) suite of codes into the Workbench. The ARC codes contain both legacy codes like DIF3D and REBUS-3 that were developed with over 30 years of experience, and newer NEAMS additions like MC²-3 and PERSENT. These codes are extremely attractive by their wide capabilities and computational efficiency. Integrating the ARC codes into the Workbench benefits directly the ART community by providing a set of controlled, maintained and validated scripts to generate ARC inputs, which promotes best practices, reduces the learning curve, and facilitates project collaboration. The second benefit from this project is to improve the user experience with the ARC codes: the Workbench interface provides assistance for building an input through auto-completion, real-time validation, access to templates, and visualization.

Last but not least, the ARC integration into the NEAMS Workbench enables new modeling capabilities, those will be extremely useful for advanced reactor design and analyses. The PyARC module facilitates and automatizes complex calculations and workflows for reactor analysis enabling geometrical perturbations, cross-section update through depletion, etc. The Dakota/PyARC coupling in the Workbench was also demonstrated to enable mathematical optimization [25] and SA/UQ techniques [26] with ARC neutronic simulations.

For demonstration purposes, the ARC codes were used through the Workbench for solving the SFR-UAM benchmark problems. The ARC codes are currently used at ANL through the Workbench by nuclear engineers for SFR core design analysis [25, 26, 27]. University faculties were trained in early FY-2018 through a webinar [28] on using the ARC codes through the Workbench. Future efforts will focus on continuously adding existing modeling capabilities available with the ARC codes, training new users to continue building some experience, and on integrating high-fidelity codes such as PROTEUS into the Workbench.

References

- [1] B. T. Rearden, R. A. Lefebvre, “Objectives of the NEAMS Workbench,” ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).
- [2] B. T. Rearden, R. A. Lefebvre, A. B. Thompson, B. R. Langley, N.E. Stauff, “Introduction to the Nuclear Energy Advanced Modeling and Simulation Workbench,” M&C 2017, Jiju Island, South Korea, April (2017).
- [3] N. Stauff, N. Gaughan, and T. Kim, “ARC integration into the NEAMS Workbench,” ANL/NE-17/31, September 30, 2017.
- [4] ARC 11.0: Code System for Analysis of Nuclear Reactors, Argonne National Laboratory (2014). Available from Available from Radiation Safety Information Computational Center as CCC-824.
- [5] Changho Lee, Yeon Sang Jung, and Won Sik Yang, “MC²-3: Multigroup Cross Section Generation Code for Fast Reactor Analysis,” ANL/NE-11-41 Rev.3, August 31 (2018).
- [6] K. L. Derstine, “DIF3D: A Code to Solve One-, Two-, and Three-Dimensional Finite Difference Diffusion Theory Problems,” ANL-82-64, Argonne National Laboratory (1984).
- [7] B. J. Toppel, “A User’s Guide to the REBUS-3 Fuel Cycle Analysis Capability,” ANL-83-2, Argonne National Laboratory (1983).
- [8] M. A. Smith, W. S. Yang, A. Mohamed, E. E. Lewis, “Perturbation and Sensitivity Tool Based on the VARIANT Option of DIF3D,” ANS Transactions 107, San Diego, Nov. 11-15 (2012).
- [9] Nicolas E. Stauff, Taek K. Kim, Robert A. Lefebvre, Brandon R. Langley, Bradley T. Rearden, “Integration of the Argonne Reactor Computation codes into the NEAMS Workbench,” ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).
- [10] R. A. Lefebvre, A. B. Thompson, B. R. Langley, B. T. Rearden, “NEAMS Workbench 1.0 Beta Status,” ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).
- [11] Robert A. LEFEBVRE, Brandon R. LANGLEY, and Jordan P. LEFEBVRE, “Workbench Analysis Sequence Processor”, ORNL/TM-2017/619, UT-Battelle, LLC, Oak Ridge National Laboratory (2017).
- [12] Kitware: Computation Model Builder (2017). <http://www.computationalmodelbuilder.org/>
- [13] LLNL: VisIT Visualization Tool (2002– 2016). <https://wci.llnl.gov/codes/visit>
- [14] C. H. Lee, N. E. Stauff, “Improved Reactivity Worth Estimation of MC2-3/DIF3D in Fast Reactor Analysis,” Proceedings of ANS Sumer Meeting, paper 14201, San Antonio, Texas (2015).
- [15] R. E. Alcouffe, F. W. Brinkley, D. R. Marr, and R. D. O’Dell, “User’s Guide for TWODANT: A Code Package for Two-Dimensional, Diffusion-Accelerated, Neutral-Particle Transport,” LA-10049-M, Los Alamos National Laboratory (1990).

- [16] R. E. Alcouffe, R. S. Baker, J. A. Dahl, S.A. Turner, and Robert Ward, “PARTISN: A Time-Dependent, Parallel Neutral Particle Transport Code System,” LA-UR-08-07258 (Revised Nov. 2008).
- [17] G. Palmiotti et al, “Variational nodal transport methods with anisotropic scattering,” Nuclear Science and Engineering, Vol. 115, pp. 233-243 (1993).
- [18] G. Aliberti and M. Smith, “PERSENT: need of a deterministic code for sensitivity analysis in 3D geometry and transport theory,” Proceedings of PHYSOR2014, Kyoto, Japan (2014).
- [19] Gerald Rimpault et al, “Objectives and Status of the OECD/NEA sub-group on Uncertainty Analysis in Best-Estimate Modelling (UAM) for Design, Operation and Safety Analysis of SFRs (SFR-UAM),” FR’17, Yekaterinburg, Russia.
- [20] Nicolas E. STAUFF et al., “Evaluation of the OECD/NEA/SFR-UAM Neutronics Reactivity Feedback and Uncertainty Benchmarks,” FR’17, Yekaterinburg, Russia.
- [21] Benchmark For Uncertainty Analysis In Modelling (UAM) For Design, Operation and Safety Analysis of SFRs, Core Definitions (version 1.5-6 – 2016 September 13th)
- [22] A. C. Kahler et al., “ENDF/B-VII.1 Neutron Cross Section Data Testing with Critical Assembly Benchmarks and Reactor Experiments,” Nuclear Data Sheets, 112, pp. 2997-3036 (2011).
- [23] M. Herman et al, “COMMARA-2.0 Neutron Cross Section Covariance Library,” BNL-94830-2011 (2011).
- [24] Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.7 User’s Manual.
- [25] K. Zeng, Nicolas Stauff, “Multi-criteria optimization of the Advanced Burner Test Reactor,” journal paper under development (2018).
- [26] Nicolas E. Stauff, Robert A. Lefebvre, Laura Swiler, Bradley T. Rearden, “Coupling of DAKOTA with the ARC suite of codes in the NEAMS Workbench for Uncertainty Quantification,” ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).
- [27] Bo Feng and Nicolas Stauff, “High Power Density Annular Fuel in a Fast Test Reactor,” ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).
- [28] ARC Workbench Webinar, <https://bluejeans.com/s/2fi9p>, October 12, (2017).

Appendix A: Tutorial

The objective of this tutorial is to assist a new or experienced ARC user getting started with the NEAMS Workbench for nuclear core analysis.

Throughout this tutorial, a core modeling example is proposed based on a simple 100 MW thermal SFR core model, which layout is displayed in Figure A0-1. The full example inputs and a sample of outputs are available in the online [tutorial](#). This directory contains also the following list of input/output:

- *arc_tutorial.son*: Workbench input
- *decay_chain_Mo.son*: simple decay chain used
- *lumped_Mo.son*: lumped element definition
- *arc_tutorial.summary*: summary output file with main results
- *arc_tutorial.timeline.out*: timeline of calculation
- *arc_tutorial.inp*: associated ARC inputs
- *arc_tutorial.out*: associated ARC outputs
- *arc_tutorial.isotxs.edit*: generated ISOTXS file (text format)
- *arc_tutorial.rebus_000000.vtk*: generated REBUS VTK file

Figure A0-1. Core layout of the SFR model used in this example.

The workbench PyARC “common input” is made up of a series of blocks containing **keywords** (bolded in this tutorial). This tutorial goes through the description of the **geometry** card in section 1, and with the **calculations** cards in section 2. More detailed description of the output files and of the post-processing visualization available are shown in section 3. For detailed description of

each card, the reader is invited to access the input documentation (Appendix B: PyARC Documentation).

Most keywords require other keywords to follow them, creating ‘levels’ of blocks within blocks. When a keyword with a higher level is selected by autocomplete, a pair of brackets will appear along with the keyword. All other keywords that come below must be inside these brackets and their order can be changed. The user can access the list of keywords available in a block through the auto-completion. It is recommended that the user keeps the end brackets lined up with the corresponding keyword and indent for each new (lower) level. Comments are entered after the ‘%’ symbol. The main structure of the common input is presented in Figure A0-2 and each main block/sub-block is presented in this tutorial.

```
=arc
geometry{ % description of the core model
  materials{ } % list of materials as atom/weight fractions, densities or formulas
  blends{ } % list of mixtures of materials for homogenized modeling
  surfaces{ } % list of surfaces used for geometry description (hexagon, plane, cylinder, square)
  reactor{ } % description of the core, with lattice of assemblies
  % each assembly is defined by stacking several axial sub-assemblies
}
calculations{ % description of the list of calculations
  mcc3{ } % description for MCC3 (+ Twodant or PARTISN) calculations
  dif3d{ % description for DIF3D calculations
    rebus{ } % description for REBUS calculation
    percent{ } % description for PERSENT calculations
  }
}
end
```

Figure A0-2. Structure of the common input.

1. geometry

The **geometry** block is where the user defines the geometry modeled. It contains three mandatory sub-levels: **materials**, **surfaces** and **reactor**. The **blends** block is not mandatory but can also be defined within **geometry**. This section describes each of these sub-blocks. Upon completion of the **geometry** block, the user can visualize the core geometry with the “*View/View Geometry*” command.

1.1 materials

The **materials** block is mandatory since at least one **material** sub-block should be defined. A **material** sub-bloc requires a name (which should be unique) and a temperature (in Kelvin). The Workbench allows several different ways to define the isotopic content of a **material**: with weight fractions (**wfrac**), atom fractions (**afrac**), atom densities (**adens**), weight densities (**wdens**), and atom formula (**aform**). Different types can be used within the same input. It should be noted:

- in **adens** one can use the **lumped_element_aden** to specify atom density of lumped fission products defined in **calculations/mcc3/lumped_element_text_file**
- **aform** is useful for defining compounds. In “**aform/afrac(x)**”, **x** is the number of atoms in the compound

Example: Let’s define different materials for the core composition and set their temperature to 300K:

- *fuel*: UO₂, weight density = 10 g/cm³, 17at% U-235 enrichment
- *coolant*: Na-23, weight density = 0.85 g/cm³
- *structure*: Fe-56, atom density = 5.10⁻² at/cm-barn

```
materials{
  material ( structure ) {
    temp  = 300      % temperature in Kelvin
    dens{
      aden ( fe56 ) = 5e-2 % atom density in atom/barn-cm
    } }
  material ( fuel ) {
    wdensity = 10      % weight density in g/cm3
    temp  = 300      % temperature in Kelvin
    aform( 2 ){        % atom count
      afrac ( O16 ) = 1 % atom fraction sum to one
    }
    aform( 1 ){        % atom count
      afrac ( u235 ) = 0.17 % atom fraction sum to one
      afrac ( u238 ) = 0.83 % atom fraction sum to one
    } }
  material ( coolant ) {
    temp  = 300      % temperature in Kelvin
    wdens{
      wden ( na23 ) = 0.85 % weight density in g/cm3
    } } }
}
```

1.2 blends

The **blends** block is not mandatory but can be used to combine materials to describe volume-homogenized compositions. When the volume fraction provided sums to less than one, the remaining is considered as void. **Blends** require definition of individual materials.

Example: Let’s define a blend called *mix_ref* made of 20vol% *coolant* and 80vol% *structure*.

```
blends{
```

```
blend ( mix_ref ) {
  volf ( coolant ) = 0.2 % volume fraction sum to less than or equal to one
  volf ( structure ) = 0.8 % volume fraction sum to less than or equal to one
} }
```

1.3 surfaces

The second step toward building the geometry is to define surfaces. The choices currently available are hexagons with **orientation**=y or x (the y or x axis intersects two opposite vertices of the hexagon, see Figure A1-4) and **normal** = z (the hexagonal prism extends vertically in the z direction), cylinders along the z axis, squares along the x/y axes, and planes perpendicular to the x, y, and z axis. For hexagons, “pitch” means flat-to-flat distance and “radius” is on the outer ring. All dimensions are in meters.

Figure A1-1. Hexagon description with orientation=y.

Example: In this example, two types of assemblies will be defined as described with the axial assembly layout in Figure A1-2. The radial fuel assembly layout, which will be described explicitly, is shown in Figure A1-3.

- We need first to define axial surfaces *z0* (0m), *z1* (1m), *z2* (2m), *z3* (3m)
- Then, we need two hexagonal surfaces *hex_outer* (**pitch** = 21cm), *hex_inner* (**pitch** = 20cm), and *hex_pin* (**pitch** = 2.5cm) that will represent the pitch of the pin lattice
- Finally, we define the cylinder representing the fuel pin with *cyl_fuel* (**radius** = 1.0 cm) and *cyl_clad* (**radius** = 1.2 cm).

```
surfaces{ % dimensions in meter
  plane ( z0 ) { z = 0.0 }
  plane ( z1 ) { z = 1.0 }
  plane ( z2 ) { z = 2.0 }
  plane ( z3 ) { z = 3.0 }
  hexagon ( hex_pin ){ orientation = x normal = z pitch = 0.025 }
```



```

hexagon ( hex_outer ) { orientation = y normal = z pitch = 0.21 }
hexagon ( hex_inner ) { orientation = y normal = z pitch = 0.20 }
cylinder ( cyl_fuel ) { axis = z radius = 0.01 }
cylinder ( cyl_clad ) { axis = z radius = 0.012 }
}

```

Figure A1-2. Axial layout of the assemblies in the example.

Figure A1-3. Radial layout of the fuel assembly and pin in the example.

1.4 reactor

After **surfaces**, the next keyword is **reactor**, which is required as this is where one will describe the lattice of assemblies. Here the **lower_axial_surf** and **upper_axial_surf** refer to the entire system and need to be defined with associated boundary conditions (extrapolated, reflective, or extrapolated). Other axial sections can be defined later under **sub_assembly**. For a 2D type of core geometry, the lower and upper axial surfaces should not be defined. In this case, only one **sub_assembly** should be defined per assembly without upper and lower axial surfaces. If the **lower_axial_surf** is specified, it must refer to a plane with $z=0$. All upper and lower axial surfaces must refer to planes in 'z' (perpendicular to z axis).

Example: In this example, we are modeling a SFR core with 3-D (hexagonal-Z) geometry. We need to specify upper ($z3$) and lower ($z0$) surfaces and will apply an *extrapolated* boundary condition.

```
reactor{
  lower_axial_surf = z0
  lower_boundary_condition = extrapolated
  upper_axial_surf = z3
  upper_boundary_condition = extrapolated
```

1.4.1 reactor_hex_lattice and reactor_square_lattice

Next, one will define the lattice of assemblies using exactly one of the following keywords: **reactor_hex_lattice** and **reactor_square_lattice**. This is where the general layout of the core's hexagonal or square lattice is defined. The example focuses on the hexagonal lattice, but the square lattice input is similar. Figure A1-4 and Figure A1-5 are important as they show the location in a ring where the indexing starts.

Figure A1-4. Radial layout for the hexagonal lattice.

Figure A1-5. Radial layout for the square lattice.

The lattice definition requires to define **pitch**, which takes the name of the hexagon previously defined under **surfaces** for the assembly boundary, **num_ring**: the number of rings in the lattice (center is ring 1). Assuming for now that in each ring, every assembly in that ring is the same type, **fill** says which type of assembly is in which ring. The list goes in order starting with ring 1 and moving outward. The **replace_ring** or **replace** options are used to define any assemblies that are different from the rest of the ring. The details of the different assemblies will be defined in Section 1.4.2 under **assembly**. There is the option to use a “**null**” assembly in order not to define an assembly (in corner locations).

Example: In this example, we are modeling 6 rings of assemblies, the pitch should refer to the *hex_outer* surface previously defined. The assemblies are located according to Figure A1-6.

```
reactor_hex_lattice {
  pitch      = hex_outer
  num_ring   = 6
  fill       = [ afuel afuel afuel afuel afuel aref ]
  replace{ ring=3 index=1 name=aref }
  replace{ ring=3 index=5 name=aref }
  replace{ ring=3 index=9 name=aref }
}
```

Figure A1-6. Core radial layout in the example.

1.4.2 *assembly*

Now that the layout of the lattice is built, the next step consists in defining each assembly, which is done with the **assembly** block. The assembly is automatically defined as hexagonal or square depending on the type of lattice above-defined. It is made of several **sub_assembly** blocks, each representing a different axial region. At least one **sub_assembly** is necessary for every **assembly**. If the assembly is uniform throughout, only one **sub_assembly** is needed.

Example: In the example, the assembly *aref* is made of mixture composition *mix_ref* throughout the length of the assembly (as seen in Figure A1-2).

```
assembly (aref) {
  sub_assembly (rad_ref) {
    lower_axial_surf = z0
    upper_axial_surf = z3
    material=mix_ref
  }
}
```

In the above-described example, the sub-assembly is filled with one material. For explicitly defined sub-assembly, the sub-assembly can be filled with explicitly defined pins (using the **hex_lattice** or **square_lattice**) and with solid regions (using **radial_region**). The ARC codes will only use homogenized composition, so the volume fraction of each elementary material calculated in the homogenized material is printed out in the summary file. Here is a list of recommendations for defining a geometry explicitly:

- It is recommended that the user not leave any space undefined. One way around this is to make a “void” material that has a very low atom density.
- In “pin/sub_pin”, don’t include **inner_surf** for the central region.

- For **wire_wrap**, the **path** is the height of one turn of the wire (see Figure A1-7). The **wire_wrap** is not available in the **assembly_square** geometry.

Example: In the example, the assembly *afuel* is made of 3 axial sub-assemblies. The upper and lower sub-assemblies (*low_ref* between *z0* and *z1* and *up_ref* between *z2* and *z3*) take a mixture composition *mix_ref* (as seen in Figure A1-2), while the middle region *mid_fuel* (between *z1* and *z2*) use explicitly defined composition. The *mid_fuel* sub-assembly takes 5 rings of *pin1* with lattice pitch defined with *hex_pin*, *coolant* being the pin-surrounding material. The *pin1* are made of *fuel* within *cyl_fuel* and *structure* within *cyl_clad*. A wire wrap is defined surrounding *pin1* with diameter of 1mm and path of 20cm. Finally, a hexagonal duct is defined as a radial region surrounding the lattice of pins, within *hex_inner* and *hex_outer*.

```
assembly ( afuel ) {
  sub_assembly ( low_ref ) {
    lower_axial_surf = z0
    upper_axial_surf = z1
    material=mix_ref
  }
  sub_assembly ( mid_fuel ) {
    lower_axial_surf = z1
    upper_axial_surf = z2
    hex_lattice {
      pitch      = hex_pin
      num_ring    = 5
      outer      = coolant
      fill       = [ pin1 pin1 pin1 pin1 pin1 ]
      pin ( pin1 ) {
        sub_pin {
          material = fuel
          outer_surf = cyl_fuel    % cylinder name
        }
        sub_pin {
          material = structure
          inner_surf = cyl_fuel    % cylinder name
          outer_surf = cyl_clad    % cylinder name
        }
      }
      wire_wrap{
        diameter = 0.001 % dimensions in meter
        path = 0.2 % dimensions in meter
        material = structure
      }
    }
  }
  radial_region {
    material = structure
    inner_surf = hex_inner    % surface name
    outer_surf = hex_outer
  }
}
```

```

} }
sub_assembly ( up_ref ) {
  lower_axial_surf = z2
  upper_axial_surf = z3
  material=mix_ref
} }

```

Figure A1-7. Fuel assembly pin lattice and pin wire geometry in the example.

2. calculations

The **calculations** block describes the list of calculations requested by the user on the defined geometry. It contains the **mcc3** and **dif3d** sub-levels, **dif3d** contains also the **rebus** and **persent** sub-levels. It also contains the **num_cpu_max** option which provides the maximum number of CPUs used for running in parallel different MC²-3 cells, DIF3D and PERSENT calculations (each calculation is always run on 1 CPU, it is not being parallelized).

2.1 mcc3

The MC²-3 code can be run as a stand-alone code to generate multi-group cross-sections (XS), or can be used together with a 2D full-core calculation code (such as TwoDant or PARTISN). By default, the cell calculation performed with **mcc3** is on a mixture type of geometry, unless the **hete_cell_options** block is specified. Every sub-assembly modelled in the geometry must be called for whether in **associated_sub_assembly** or in **other_sub_assembly_using_XS** when it uses cross-section generated in another cell based on the compositions of another sub-assembly.

When the 2-step procedure with TWODANT or PARTISN is not used, the fissile regions should use the **bucketing_search** option. For a non-fissile region, the user can choose to use the neutron leakage data from another cell with the **external_source** option.

Example: In our example, we will be using the 1-step procedure and condense multi-group cross-sections into the **ANL33** 33-group structure, using the **ENDF/B-VII.0** nuclear data library, with 3rd

order scattering. We provide an external file "**lumped_Mo.son**" that contains the isotopic composition of the lumped **MoFP** element. Two cells are defined:

- cell "**a**" takes the **mid_fuel** homogenized composition. One will request buckling search on this cell.
- Cell "**b**" takes **rad_ref** homogenized composition. The cross-sections obtained will be used in the **low_ref** and **up_ref** sub-assemblies (those contain the same material). Since these are reflector regions, buckling-search is de-activated, but one uses the leakage file from cell "**a**" as external source for cell "**b**".

```
mcc3{
  xslib      = "endf7.0"
  egrouppname = ANL33
  scattering_order = 3
  lumped_element_text_file( MoFP ) = "lumped_Mo.son"

  cell( a ){
    associated_sub_assembly = mid_fuel
    buckling_search=true
  }
  cell( b ){
    associated_sub_assembly = rad_ref
    other_sub_assembly_using_XS = [ low_ref up_ref ]
    buckling_search=false
    external_source=a
  }
}
```

How to get an equivalent RZ geometry? calculate the number of each type of assemblies (whose sub-assemblies are represented by *mcc3_id*). Calculate the area for each radial region: the area of an hexagon is equal to $A = \frac{\sqrt{3}}{2} p^2$ with p the pitch of the assembly. Find the cumulative "equivalent radius" by solving for *r* in $A = \pi r^2$. In **RZ_geometry**, fill out the areas with *mcc3_id* to define the RZ core modeled:

- the number of columns must equal number of R boundaries and number of rows must equal number of Z boundaries
- the first row of *mcc3 ID*'s is the bottom of the core and the last row is the top of the core.

Example of RZ geometry and associated PyARC input

Figure A2-1. Input example and associated RZ geometry.

```

rzmflx_code_options{
  egrouppname = ANL1041
  code = twodant
  R_boundaries = [ r01 r04 r08 r12 ]
  R_nodes_distance= 0.05
  Z_boundaries = [ z1 z2 z3 ]
  Z_nodes_distance= 0.05
  bottom_boundary_condition = vacuum
  top_boundary_condition = vacuum
  right_boundary_condition = vacuum
  left_boundary_condition = reflective
  SN_angular_order = 12
  core_2d_geometry = [ a b a r
                       c b d r
                       e f e r ]
}

```

2.2 dif3d

In this block, the user needs to specify which neutron solver to use (DIF3D-FD, DIF3D-Nodal, or VARIANT) and provide the associated options. The type of geometrical symmetry used to model the core should be specified here (sixth_core, fourth_core, third_core, half_core, full_core). Some important notes on the DIF3D options:

- Symmetry specification will force symmetry even if the whole core that the user built does not have that type of symmetry, without giving any error message. The user should make sure to select the appropriate geometry type.
- **hexagonal_infinite_lattice** and **cartesian_infinite_lattice** only work with one subassembly and force reflective boundary conditions
- hexagonal and triangular geometries require **reactor/core_hexlattice**

- cartesian geometry requires **reactor/core_squarelattice**
- **extract_volume_high_fastflux** extracts the total volume with fast flux level higher than the value provided with this option. This requires refined **num_axial_burnup_zones** in order to give accurate results.
- The **isotxs**=“previous” tells the code to use the cross sections isotxs file generated from the previously defined **mcc3** calculation. If the user already has a valid isotxs file, it can be specified here – this will be used for reference (un-perturbed) calculation. However, when using this option, the user should keep a valid and consistent **mcc3** input block.

Example: In our example, the core power is set to 100 MW, we will use fill core geometry (no symmetry), with maximum axial mesh size of 5 cm. The finite difference solver is employed.

```
dif3d{
  power          = 100.0e6
  geometry_type  = hexagonal_full_core
  isotxs         = previous
  max_axial_mesh_size = 0.05
  dif_fd_options{ }
```

2.2.1 rebus

The REBUS-3 code is used for once-through depletion and equilibrium search fuel cycle simulation based on the DIF3D options previously specified. The **equilibrium_search** option is used for equilibrium problems. A few important notes for equilibrium problems:

- they cannot be run while updating XS at each time-step (**xs_update_at_each_timestep**)
- an external feed cannot be used as “enriched” feed (**feeds_enrich**) in one **path** and as “other” feed (**feeds_other**) in another **path**
- **path** definition is required for every sub-assembly containing fissile to define reactor charge specification for every sub-assembly

Example: A once-through depletion strategy shown in Figure A2-2 is implemented where the core is irradiated for 3 cycles of 100 efpd, with 10 days of shutdown between each cycle. Each cycle is subdivided into 2 intervals. One of the reference decay chain provided, based on the lumped-Mo (**MoFP**) is used.

```
rebus{
  cycle_length      = 100.0 % in full power days
  shutdown_time_between_cycle = 10.0 % in days
  num_cycles        = 3
  num_subintervals  = 2
```



```

decay_chain{
  list_isotopes      = [u234 u235 u236 u238 np237 pu236 pu238 pu239 pu240 pu241 pu242
                        am241 am242m am243 cm242 cm243 cm244 cm245 cm246]
  list_lumped_elements = [MoFP]
  list_dummy_elements = [dum1 dum2]
  text_file          = "decay_chain_Mo.son"
}

```

Figure A2-2. Illustration of time-step equivalence to irradiation calculation performed with REBUS.

2.2.2 *percent*

PERSENT is used for perturbation theory and sensitivity calculations. There are two main keywords under **present** for specifying calculations: **pert_calc**, and **sens_calc**. **pert_calc** is used for perturbation theory calculations. It requires to define a perturbed state for the core considered to calculate the difference in reactivity with the reference state. **sens_calc** is used for sensitivity calculations on multi-group cross-sections calculated with MC²-3. Sensitivity calculations can be performed on the reference core or on a perturbed state, and can be followed by uncertainty calculations by providing a covariance matrix with the **covariance_matrix_text_file** option.

Here are a few important notes for **percent** problems specification:

- Percent calculations require using **variant_options** in **dif3d** or defining **variant_options** in the **percent** block (this can be useful especially if the user wishes to run REBUS with equilibrium option, which is typically used with a diffusion theory solver for timely computation)
- The **rebus_output_file** option is optional, it takes the path to a REBUS output file and allows to skip the REBUS calculation. However, when using this option, the user should keep a valid and consistent **rebus** input block
- Perturbation and sensitivity calculations can be performed on any time-step of the REBUS calculation using the **depletion_step** option
- **surface_perturbation**: Both reference and perturbed surfaces must be from the same type (hexagons, cylinders, planes, or squares). In **pert_calc**, this option requires **pert_option** =

direct (this runs DIF3D for reference and perturbed cases). In **sens_calc**, this option requires **sens_type** = Keff: the sensitivity problem is run on the perturbed geometry only.

- **density_perturbation_factor** and **new_temp** cannot both be used in one same **material_perturbation** block
- **sensitivity_factor** should not be equal to 1.0

As for MC²-3, the DIF3D and PERSENT calculations will be run in parallel on the maximum number of CPUs allowed in **calculations/num_cpu_max**. The **dif3d_refcalc_depsteps** takes a list of depletion steps on which to run a set of reference DIF3D calculation. This allows to generate the forward and adjoint flux files to be used in recurring perturbation and sensitivity calculations at each given time-steps. The DIF3D calculation will use the reference ISOTXS file (at step 0) if an ISOTXS file at the exact time-step requested was not previously generated.

Example: Two perturbation problems are proposed based on the Variant options (661, P1P0). The reference DIF3D calculation is performed on the 4th depletion step (which is equivalent to 160th day of the simulation, as shown in Figure A2-2) to generate reference adjoint and forward flux files. The “N” perturbation computes the *coolant* density coefficient by reducing the *coolant* density by 1%. The “D” perturbation computes the Doppler coefficient by increasing the temperature of the *fuel* to 600K. Both perturbations are performed using the first order perturbation theory.

```

persent{
  variant_options{
    polynomial_approx_source    = 6
    polynomial_approx_fluxes    = 6
    polynomial_approx_leakages  = 1
    angular_approx              = 1
    anisotropic_scattering_approx = 0
  }
  dif3d_refcalc_depsteps = [4]
  pert_calc ( N ) {
    depletion_step = 4
    pert_type      = keff
    material_perturbation ( coolant ) {
      list_isotopes_perturbed = [ all_from_material ]
      density_perturbation_factor = 0.99
    }
    xs_updated = false
    pert_option = first_order_perturbation_theory
  }
  pert_calc ( D ) {
    depletion_step = 4

```

```

pert_type    = keff
material_perturbation ( fuel ) {
  list_isotopes_perturbed  = [ all_from_material ]
  new_temp=600
}
xs_updated = true
pert_option = first_order_perturbation_theory
}

```

3. Execution and post-processing

Example: After execution of the PyARC example input (with **run**), here is the list of inputs/outputs files that are generated and how to visualize some of the automatically available information:

- **“.timeline”**: real-time printout of the timeline of the calculation
- **“.summary”**: summary results from all calculations performed. Use *processor: flux spectrum* to plot the flux spectrum (as shown in Figure 3-2)
- **“.isotxs”**: binary files of 33gp cross-sections
 - o **“.isotxs_R_0”**: **R**eference (un-perturbed), time-step **0**
 - o **“.isotxs_D_4”**: **D** perturbation, time-step **4**
- **“.isotxs.edit”**: text files of 33gp cross-sections. Use *processor: ISOTXS* to plot the multi-group cross-sections (as shown in Figure 3-1)
- **“.vtk”**: files containing 3D density, flux, power and reactivity coefficients. See Figure A3-1 that illustrates the VisIt plot and assists the user through the few steps required to set up the VisIt visualization.
 - o **“dif3d.vtk”**: vtk file generated by DIF3D
 - o **“persent_D_ref.vtk”**: vtk **reference** file generated by PERSENT for perturbation **D**
 - o **“rebus_000050.vtk”**: vtk file generated by REBUS at time **50** days
- **“.out”**: all raw output files concatenated
- **“.inp”**: all native ARC input files concatenated
- **“.zip”**: gathers all input, output, isotxs, and summary files

Figure A3-1. Example of VisIt display and steps for a user to set it up.

Appendix B: PyARC Documentation

Package Overview

The PyARC package serves as the wrapper between the NEAMS Workbench and the ARC codes. This glue contains the logic to:

- extract information from the common input entered through the Workbench
- perform additional verifications that the validation engine of the Workbench cannot perform
- pre-process the information, calculating for instance homogenized atom densities in different regions
- generate the ARC codes' inputs
- handle the runtime environment, for instance running MC2-3 elementary cell calculations in parallel on different CPUs
- post-process the outputs, printing out summary files with main results of the different codes' outputs.

The PyARC module is developed in Python through a collaborative environment on GitLab at code.ornl.gov. Pylint is used during the development process to check compatibility with PEP8 coding standards. This module is developed under the leadership of ANL with the assistance and contribution of the Workbench team at ORNL.

Table of Contents

1. [Getting Started](#) - Getting Started with PyARC
 - [Requirements](#) - Requirements for PyARC
 - [Licences](#) - Codes licences and installation required
 - [Repository](#) - Optional: How to access the PyARC repository
 - [Deployment](#) - Extract and install PyARC distribution
 - [Executables](#) - Linking ARC codes executables to PyARC
 - [Linking](#) - Linking PyARC to the Workbench
 - [Messages](#) - Important messages to early users
2. [User Help](#) - Additional information for a user
 - [References](#) - List of references
 - [Tutorial](#) - Links to tutorial, sample inputs and benchmark problems
 - [Release History](#) - Release History
 - [Contact](#) - contact information for questions
3. [PyARC](#) - Components of the PyARC package
4. [Unit Tests](#) - Testing pre/post processing and execution logic of the ARC codes
5. [Input Definition](#) - ARC input definition
 - [Grammar](#) - NEAMS Workbench grammar for ARC
 - [Templates](#) - Pre-generated ARC input templates used by NEAMS Workbench for input autocompletion
 - [Highlighters](#) - NEAMS Workbench font style instructions for ARC SON input
 - [Schema](#) - Input description and rules of ARC

Getting Started

For any individuals wanting to use the PyARC bundle for ARC calculations through the Workbench, below are the installation requirements and tests proposed.

Requirements

- Python-2.7
- Git (for developers)
- MPI (optional)

Licences

RSICC licence required:

- PARTISN (optional)

ANL or RSICC licence required:

- MCC3
- ARC

Repository

For users starting from an access to the repository:

- You will need to save your ssh-key in code.ornl.gov.
- Clone PyARC `git clone git@code.ornl.gov:neams-workbench/PyARC.git ~/PyARC`
- Change directory into pyARC `cd ~/PyARC`
- Execute the dependencies logic `./scripts/dependencies.sh`

Deployment

For users starting from the repository, executing the `scripts/dependencies.sh` script will produce a PyARC.tar.gz bundle.

Conducting the following steps will extract and install the PyARC distribution:

1. `mkdir PyARC && cd PyARC` # Create a PyARC install directory
2. `tar -xvf path/to/PyARC.tar.gz` # extract installation contents
3. `cd test && python -m unittest discover -v` && verify installation by running tests

Executables

In order to connect PyARC to the ARC codes (MCC2-3, DIF3D, etc.) you need to provide a resource configuration file.

Create the `.pyarc.rc` file within the PyARC/repo directory:

```
[exec]
path = /path/to/ARC/bin
mpi_path = /path/to/MPI/bin/mpiexec.x (this is optional)
```

Optional: Update the `/path/to/MPI/bin/mpiexec.x` to point out to your local installation of MPI executable - this is optional and allows running PARTISN in parallel.

Required: Update the `/path/to/ARC/bin` to point to your local installation of the ARC codes. The location installation is minimally expected to be:

```
/projects/PyARC/arc/bin/ISOTXS_merge.x
/projects/PyARC/arc/bin/dif3d.x
/projects/PyARC/arc/bin/dif3d_to_vtk.x
/projects/PyARC/arc/bin/mcc3.x
/projects/PyARC/arc/bin/rebus.x
/projects/PyARC/arc/bin/twodant.x
/projects/PyARC/arc/bin/partisn.x

/projects/PyARC/arc/bin/lib:
lib.gamma.e70
lib.mcc.e70
lib.pw.200.e70
```

Linking

- `Workbench/Configurations -> new Arc -> executable` : give full path to PyARC.py

Messages

- Don't trust the code's results blindly (benchmark exercises are encouraged)
- Report any bugs to nstauff@anl.gov
- Feedbacks and ideas for improvement are encouraged
- Users can develop additional post-processing / plotting tools that can be integrated in the main ARC/Workbench
- Update frequently the code (new updates may lead to changes in results)

- Backward compatibility (input may change in future updates)

User Help

References

To cite PyARC, please use the following reference:

- Nicolas E. Stauff, Taek K. Kim, Robert A. Lefebvre, Brandon R. Langley, Bradley T. Rearden, "Integration of the Argonne Reactor Computation codes into the NEAMS Workbench," ANS Summer meeting, Philadelphia, PA, USA, June 17-21, 2018.

This is an additional list of useful references:

- Nicolas E. Stauff, Robert A. Lefebvre, Laura Swiler, Bradley T. Rearden, "Coupling of DAKOTA with the ARC suite of codes in the NEAMS Workbench for Uncertainty Quantification," ANS Summer meeting, Philadelphia, PA, USA, June 17-21, 2018.
- N. Stauff, N. Gaughan, and T. Kim, "ARC integration into the NEAMS Workbench," [ANL/NE-17/31](#), September 30, 2017.
- B. T. Rearden, R. A. Lefebvre, A. B. Thompson, B. R. Langley, N.E. Stauff, "Introduction to the Nuclear Energy Advanced Modeling and Simulation Workbench," M&C 2017, Jiju Island, South Korea, April (2017).

Tutorial

A tutorial was developed to help train new users and contains a [presentation](#) and an associated [input](#).

Additional information on the status of the ARC integration into the NEAMS Workbench is available with the FY17 milestone [report](#). This report contains a tutorial in Appendix A that should be useful for new users, even though some options may not be fully up-to-date.

Up-to-date sample inputs are provided as well:

- [Sample1](#) - uses explicit description of an assembly, with infinite REBUS lattice calculation using lumped fission products
- [Sample2](#) - uses full core description with homogenized assemblies, MC2-3 2-steps (Twodant) and DIF3D running FD
- [Sample3](#) - uses full core description (1/3 symmetry) with homogenized assemblies, MC2-3 1-step (external fission source), VARIANT, PERSENT calculations for perturbation and sensitivity problems

The user has also access to up-to-date inputs generated as contributions for the SFR-UAM benchmark [Gerald Rimpault et al, "Objectives and Status of the OECD/NEA sub-group on Uncertainty Analysis in Best-Estimate Modelling (UAM) for Design, Operation and Safety Analysis of SFRs (SFR-UAM)," FR'17, Yekaterinburg, Russia.].

- [MET1000](#) - Reference input for medium-size metallic-fueled (MET1000) SFR core
- [MOX3600](#) - Reference input for large-size oxide-fueled (MOX3600) SFR core
- [AssemblyDepletion](#) - Reference input for MOX3600 infinite assembly depletion problem

Release History

Please read the [changelog](#) document to see what changes were made in the last releases.

Contact

For question, please contact:

Nicolas Stauff nstauff@anl.gov

PyARC

The PyARC module relies on the following sub-modules:

- [PyARC](#): main executable that loads and executes the Workbench input and postprocesses the outputs
- [PyARCModel](#): performs list of additional verification and pre-processing on the input
- [PyMCC3](#): contain the logic for input writing, execution, and post-processing for MCC3
- [PyRzmflxCode](#): contain the logic for input writing, execution, and post-processing for TWODANT and PARTISN that can be used for 2D core calculations to condense multi-group cross-sections generated with MCC3
- [PyREBUS](#): contain the logic for input writing, execution, and post-processing for REBUS

- [PyPERSENT](#): contain the logic for input writing, execution, and post-processing for PERSENT
- [PyARCUserObject](#): defines variables and procedures that are used throughout the code
- [PyARCUtills](#): contains utilities procedures
- The PyARC module also relies on the [PySCL](#) module that is developed by ORNL to provide the standard composition library (SCL) in a consistent way between different codes

Unit Tests

A new user should check proper installation of PyARC through the following list of unit tests and execution tests.

Unit tests are developed for regression testing. Currently, 95 unit tests are implemented to check the common input processing, interpretation of the standard composition library, input generation of MC2-3, DIF3D, REBUS-3, TwoDant, PERSENT, execution logic of the different codes, and post-processing of the outputs.

Consequently, the unit tests check the pre-processing, input writing, execution, and post-processing logic of PyARC.

From the root PyARC/test directory run `python -m unittest discover -v` to execute all unit tests

Input Definition

The PyARC Input definition is composed of the NEAMS Workbench Grammar and ARC SON input schema.

Grammar

The [arc.wbg](#) file contains the grammar for the ARC SON input and the paths to the schema, templates and highlighters.

Templates

The `etc` folder contains the `templates` folder, which lists all the templates developed for the PyARC SON input. Templates are used to assist users in generating the common input within the Workbench. The common input templates were developed in parallel to the schema and the common input. Those are blocks of input with default values accessible for convenience to the user. For instance, the user can directly access input blocks for different decay chains to fill out the REBUS-3 input, through the available templates ([lumped FP](#), [explicit](#), [simplified](#)). Users can build their own templates or modify existing ones if parts of their input should be re-used.

Highlighters

The [arc.wbh](#) file contains the default fonts options for the ARC SON input. This controls the look-and-feel of input as rendered in the NEAMS Workbench.

Schema

The [arc.sch](#) file contains the rules and descriptions of the ARC input as needed by PyARC and used in the NEAMS Workbench. This input is described as “common input” as it is used to generate inputs for MC2-3, DIF3D, REBUS-3, and PERSENT for integrated problem-dependent cross-section preparation, core analysis, depletion, and sensitivity/uncertainty analysis. The [arc.sch](#) schema is developed in close collaboration between ANL and ORNL teams and contains ~2000 lines of input definition. Both the common input and input schema are formatted in Standard Object Notation (SON) format. For more information on the “common input” formatting, please refer to the user manual (in development).

Table Of Contents

- [root](#)
 - [arc](#)
 - [geometry](#)
 - [materials](#)
 - [blends](#)
 - [surfaces](#)
 - [reactor](#)
 - [calculations](#)

- [mcc3](#)
- [dif3d](#)

- [lumped element external list](#)

- [afrac](#)

- [Referenced Choice Lists](#)

[/](#)

Name	Type	HowMany	ValueType	Restrictions	Description
arc	SubObject	0 or 1			Input definition for ARC codes
lumped element external list	SubObject	0 or 1			Description of a list of isotopes in a new lumped element

[/arc](#)

Input definition for ARC codes

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
geometry	SubObject	1			[required] Description of the core model
calculations	SubObject	0 or 1			Description of the list of calculations

[/arc/geometry](#)

[required] Description of the core model

How Many: 1

Name	Type	HowMany	ValueType	Restrictions	Description
materials	SubObject	1			[required] List of materials
blends	SubObject	0 or 1			List of mixtures of materials
surfaces	SubObject	1			[required] List of surfaces used for geometry description
reactor	SubObject	0 or 1			[required] Description of the core with lattice of assemblies

[/arc/geometry/materials](#)

[required] List of materials

How Many: 1

Name	Type	HowMany	ValueType	Restrictions	Description
material	SubObject	1 or more			[required] Material definition

</arc/geometry/materials/material>

[required] Material definition

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String		
wdensity	KeyedValue	0 or 1	Real	Range [0.0,+INF)	Weight density of material [g/cm3]
adensity	KeyedValue	0 or 1	Real	Range [0.0,+INF)	Atom density of material [at/barn-cm]
temp	KeyedValue	1	Real	Range [0.0,+INF)	[required] Temperature [Kelvin]
wfracs	SubObject	0 or 1			Isotopic composition defined as weight fractions
adens	SubObject	0 or 1			Isotopic composition defined as atom densities
afracs	SubObject	0 or 1			Isotopic composition defined as atom fractions
wdens	SubObject	0 or 1			Isotopic composition defined as weight densities
aform	SubObject	0 or more			Isotopic composition defined as atom formulas
lumped_element_aden	TaggedValue	0 or more			Atom density of lumped element specified in calculations/mcc3

</arc/geometry/materials/material/wfracs>

Isotopic composition defined as weight fractions

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
wfrac	TaggedValue	1 or more			

</arc/geometry/materials/material/wfracs/wfrac>

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	Choices REF: CompNames REF: NuclideIds	
value	Value	1	Real	Range [0.0,1.0]	

</arc/geometry/materials/material/adens>

Isotopic composition defined as atom densities

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
aden	TaggedValue	1 or more			

</arc/geometry/materials/material/adens/aden>

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	Choices REF: CompNames REF: NuclideIds	
value	Value	1	Real	Range [0.0,+INF)	

</arc/geometry/materials/material/afracs>

Isotopic composition defined as atom fractions

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
afrac	TaggedValue	1 or more			

</arc/geometry/materials/material/afracs/afrac>

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	Choices REF: CompNames REF: NuclideIds	
value	Value	1	Real	Range [0.0,1.0]	

</arc/geometry/materials/material/wdens>

Isotopic composition defined as weight densities

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
wden	TaggedValue	1 or more			

</arc/geometry/materials/material/wdens/wden>

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	Choices REF: CompNames REF: NuclideIds	
value	Value	1	Real	Range [0.0, +INF)	

</arc/geometry/materials/material/aform>

Isotopic composition defined as atom formulas

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	Integer	Range [1, +INF)	
afrac	TaggedValue	1 or more			

</arc/geometry/materials/material/aform/afrac>

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	Choices REF: CompNames REF: NuclideIds	
value	Value	1	Real	Range [0.0, 1.0]	

/arc/geometry/materials/material/lumped_element_aden

Atom density of lumped element specified in calculations/mcc3

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	InputKeys lumped_element_text_file_id	
value	Value	1	Real	Range (0.0, +INF)	

</arc/geometry/blends>

List of mixtures of materials

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
blend	SubObject	1 or more			[required] Mixtures definition

</arc/geometry/blends/blend>

[required] Mixtures definition

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String		
volf	TaggedValue	1 or more			[required] Volume fraction of materials

</arc/geometry/blends/blend/volf>

[required] Volume fraction of materials

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	InputKeys material id	
value	Value	1	Real	Range [0.0,1.0]	

</arc/geometry/surfaces>

[required] List of surfaces used for geometry description

How Many: 1

Name	Type	HowMany	ValueType	Restrictions	Description
cylinder	SubObject	0 or more			Cylinder definition
plane	SubObject	0 or more			Plane definition
hexagon	SubObject	0 or more			Hexagon definition
square	SubObject	0 or more			Square definition

</arc/geometry/surfaces/cylinder>

Cylinder definition

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String		
axis	KeyedValue	1	String	Choices z	
radius	KeyedValue	1	Real	Range (0.0,+INF)	

</arc/geometry/surfaces/plane>

Plane definition

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String		
x	KeyedValue	0 or 1	Real	Range (0.0,+INF)	
y	KeyedValue	0 or 1	Real	Range (0.0,+INF)	
z	KeyedValue	0 or 1	Real	Range [0.0,+INF)	

</arc/geometry/surfaces/hexagon>

Hexagon definition

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String		
orientation	KeyedValue	1	String	Choices x y	
normal	KeyedValue	1	String	Choices z	
pitch	KeyedValue	0 or 1	Real	Range (0.0,+INF)	
radius	KeyedValue	0 or 1	Real	Range (0.0,+INF)	

</arc/geometry/surfaces/square>

Square definition

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String		
axis	KeyedValue	2	String	Choices x y	
pitch	KeyedValue	1	Real	Range (0.0,+INF)	

</arc/geometry/reactor>

[required] Description of the core with lattice of assemblies

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
reactor_hex_lattice	SubObject	0 or 1			Hexagonal lattice definition within a reactor
reactor_square_lattice	SubObject	0 or 1			Square lattice definition within a reactor
lower_axial_surf	KeyedValue	0 or 1	String	InputKeys plane_id	Lower axial surface of the core - must be a plane with z = 0
upper_axial_surf	KeyedValue	0 or 1	String	InputKeys plane_id	Upper axial surface of the core
lower_boundary_condition	KeyedValue	0 or 1	String	Choices vacuum reflective extrapolated	Boundary condition on the lower axial surface
upper_boundary_condition	KeyedValue	0 or 1	String	Choices vacuum reflective extrapolated	Boundary condition on the upper axial surface
assembly	SubObject	1 or more			[required] Assembly definition

[/arc/geometry/reactor/reactor_hex_lattice](#)

Hexagonal lattice definition within a reactor

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
pitch	KeyedValue	1	String	InputKeys hexagon_id	[required] Pitch of the lattice - takes a hexagonal surface
num_ring	KeyedValue	1	Integer	Range [1,+INF)	
fill	Array of Size num_ring	1	String	InputKeys assembly_id	
replace_ring	TaggedValue	0 to num_ring			Replace a full ring with list of assemblies
replace	SubObject	0 or more			Replace one element of a ring

[/arc/geometry/reactor/reactor_hex_lattice/replace_ring](#)

Replace a full ring with list of assemblies

How Many: 0 to [num_ring](#)

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	Integer	Range [2, num_ring]	
value	Value	6 or more	String	Choices null InputKeys assembly_id	

[/arc/geometry/reactor/reactor_hex_lattice/replace](#)

Replace one element of a ring

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
ring	KeyedValue	1	Integer	Range [2, num_ring]	
index	KeyedValue	1	Integer	Range [1,+INF)	
name	KeyedValue	1	String	Choices null InputKeys assembly_id	

[/arc/geometry/reactor/reactor_square_lattice](#)

Square lattice definition within a reactor

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
pitch	KeyedValue	1	String	InputKeys square_id	[required] Pitch of the lattice - takes a square surface
num_ring	KeyedValue	1	Integer	Range [1,+INF)	
fill	Array of Size num_ring	1	String	InputKeys assembly_id	
replace_ring	TaggedValue	0 to num_ring			Replace a full ring with list of assemblies
replace	SubObject	0 or more			Replace one element of a ring

[/arc/geometry/reactor/reactor_square_lattice/replace_ring](#)

Replace a full ring with list of assemblies

How Many: 0 to [num_ring](#)

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	Integer	Range [2, num_ring]	
value	Value	6 or more	String	Choices null InputKeys assembly_id	

//

[/arc/geometry/reactor/reactor_square_lattice/replace](#)

Replace one element of a ring

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
ring	KeyedValue	1	Integer	Range [2, num_ring]	
index	KeyedValue	1	Integer	Range [1,+INF)	
name	KeyedValue	1	String	Choices null InputKeys assembly_id	

//

[/arc/geometry/reactor/assembly](#)

[required] Assembly definition

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String		
sub_assembly	SubObject	1 or more			[required] Sub-assembly definition

//

[/arc/geometry/reactor/assembly/sub_assembly](#)

[required] Sub-assembly definition

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	InputKeys cell associated sub assembly cell other sub assembly using XS	
lower_axial_surf	KeyedValue	0 or 1	String	InputKeys plane id	Lower axial surface of the sub-assembly
upper_axial_surf	KeyedValue	0 or 1	String	InputKeys plane id	Upper axial surface of the sub-assembly
material	KeyedValue	0 or 1	String	InputKeys material id blend id	Homogenized material defining the sub-assembly
hex_lattice	SubObject	0 or 1			Hexagonal lattice and pins definition within a sub-assembly
square_lattice	SubObject	0 or 1			Square lattice and pins definition within a sub-assembly
radial_region	SubObject	0 or more			Radial region definition as a radial layer within a sub-assembly

[/arc/geometry/reactor/assembly/sub_assembly/hex_lattice](#)

Hexagonal lattice and pins definition within a sub-assembly

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
pitch	KeyedValue	1	String	InputKeys hexagon id	[required] Pitch of the lattice - takes a hexagonal surface
num_ring	KeyedValue	1	Integer	Range [1,+INF)	
outer	KeyedValue	1	String	InputKeys material id blend id	
fill	Array of Size num_ring	1	String	InputKeys pin id	
replace_ring	TaggedValue	0 to num_ring			Replace a full ring with list of pins
replace	SubObject	0 or more			Replace one element of a ring
pin	SubObject	1 or more			Pin definition - a pin is made of sub-pins and wire wrap

[/arc/geometry/reactor/assembly/sub_assembly/hex_lattice/replace_ring](#)

Replace a full ring with list of pins

How Many: 0 to [num_ring](#)

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	Integer	Range [2, num_ring]	
value	Value	6 or more	String	Choices null InputKeys pin_id	

[/arc/geometry/reactor/assembly/sub_assembly/hex_lattice/replace](#)

Replace one element of a ring

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
ring	KeyedValue	1	Integer	Range [2, num_ring]	
index	KeyedValue	1	Integer	Range [1,+INF)	
name	KeyedValue	1	String	Choices null InputKeys pin_id	

[/arc/geometry/reactor/assembly/sub_assembly/hex_lattice/pin](#)

Pin definition - a pin is made of sub-pins and wire wrap

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String		
sub_pin	SubObject	1 or more			Sub-pin definition as a cylindrical layer within a pin
wire_wrap	SubObject	0 or 1			Wire-wrap definition

[/arc/geometry/reactor/assembly/sub_assembly/hex_lattice/pin/sub_pin](#)

Sub-pin definition as a cylindrical layer within a pin

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
material	KeyedValue	1	String	InputKeys material_id blend_id	
inner_surf	KeyedValue	0 or 1	String	InputKeys cylinder_id	
outer_surf	KeyedValue	1	String	InputKeys cylinder_id	

[/arc/geometry/reactor/assembly/sub_assembly/hex_lattice/pin/wire_wrap](#)

Wire-wrap definition

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
diameter	KeyedValue	1	Real	Range (0.0,+INF)	
path	KeyedValue	1	Real	Range (0.0,+INF)	
material	KeyedValue	1	String	InputKeys material id blend id	

//

[/arc/geometry/reactor/assembly/sub_assembly/square_lattice](#)

Square lattice and pins definition within a sub-assembly

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
pitch	KeyedValue	1	String	InputKeys square id	[required] Pitch of the lattice - takes a square surface
num_ring	KeyedValue	1	Integer	Range [1,+INF)	
outer	KeyedValue	1	String	InputKeys material id blend id	
fill	Array of Size num_ring	1	String	InputKeys pin id	
replace_ring	TaggedValue	0 to num_ring			Replace a full ring with list of pins
replace	SubObject	0 or more			Replace one element of a ring
pin	SubObject	1 or more			Pin definition - a pin is made of sub-pins

//

[/arc/geometry/reactor/assembly/sub_assembly/square_lattice/replace_ring](#)

Replace a full ring with list of pins

How Many: 0 to [num_ring](#)

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	Integer	Range [2, num_ring]	
value	Value	6 or more	String	Choices null InputKeys pin id	

//

[/arc/geometry/reactor/assembly/sub_assembly/square_lattice/replace](#)

Replace one element of a ring

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
ring	KeyedValue	1	Integer	Range [2, num_ring]	
index	KeyedValue	1	Integer	Range [1,+INF)	
name	KeyedValue	1	String	Choices null InputKeys pin_id	

////////////////////////////////////
[/arc/geometry/reactor/assembly/sub_assembly/square_lattice/pin](#)

Pin definition - a pin is made of sub-pins

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String		
sub_pin	SubObject	1 or more			Sub-pin definition as a cylindrical layer within a pin

////////////////////////////////////
[/arc/geometry/reactor/assembly/sub_assembly/square_lattice/pin/sub_pin](#)

Sub-pin definition as a cylindrical layer within a pin

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
material	KeyedValue	1	String	InputKeys material_id blend_id	
inner_surf	KeyedValue	0 or 1	String	InputKeys cylinder_id	
outer_surf	KeyedValue	1	String	InputKeys cylinder_id	

////////////////////////////////////
[/arc/geometry/reactor/assembly/sub_assembly/radial_region](#)

Radial region definition as a radial layer within a sub-assembly

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
material	KeyedValue	1	String	InputKeys material id blend id	
inner_surf	KeyedValue	0 or 1	String	InputKeys cylinder id hexagon id square id	
outer_surf	KeyedValue	1	String	InputKeys cylinder id hexagon id square id	

[/arc/calculations](#)

Description of the list of calculations

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
num_cpu_max	KeyedValue	0 or 1	Integer	Range [1,+INF)	Maximum number of CPUs used for parallelized calculations
mcc3	SubObject	0 or 1			Description for MCC3 (+ Twodant or Partisn) calculations
dif3d	SubObject	0 or 1			Description for DIF3D (+ REBUS and PERSENT) calculations

[/arc/calculations/mcc3](#)

Description for MCC3 (+ Twodant or Partisn) calculations

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
xslib	KeyedValue	1	String	Choices endf6.0 endf7.0 endf7.1	[required] Nuclear data library version
egroupname	KeyedValue	0 or 1	String	Choices anl33 anl70 anl230 anl1041 anl2082	Energy group structure (final condensation)
egroupvals	Array of Size 1 or more Increasing Values	0 or 1	Real	Range (0.0,+INF)	Customized energy group structure
scattering_order	KeyedValue	1	Integer	Range [0,+INF)	[required] Scattering order
inelastic_treatment	KeyedValue	0 or 1	String	Choices approximate rigorous	Type of inelastic treatment
force_mixture_calc	KeyedValue	0 or 1	String	Choices true false	Option that forces 0-D calculations on all cells
cell	SubObject	1 or more			[required] Definition of a new cell - each sub-assembly must be specified in a cell
rzmflx_code_options	SubObject	0 or 1			Description of the 2-D geometry and code options
lumped_element_text_file	TaggedValue	0 or more			Link to text file containing description of a lumped element

[/arc/calculations/mcc3/cell](#)

[required] Definition of a new cell - each sub-assembly must be specified in a cell

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	Choices a b c d e f g h i j k l m n o p q r s t u v w x y z	
associated_sub_assembly	KeyedValue	1	String	InputKeys sub_assembly_id sub_assembly_id	[required] Sub-assembly which composition is used for cell calculation
other_sub_assembly_using_XS	Array of Size 1 or more	0 or 1	String	InputKeys sub_assembly_id sub_assembly_id	List of sub-assemblies that will use the cross-sections generated with this cell
buckling_search	KeyedValue	0 or 1	String	Choices true false	Buckling search for fissile regions
external_source	KeyedValue	0 or 1	String	InputKeys cell_id	Another cell's ID which neutron leakage is used as external source for a non-fissile cell
hete_cell_options	SubObject	0 or 1			Description of the 1-D geometry used for heterogeneous XS treatment

[/arc/calculations/mcc3/cell/hete_cell_options](#)

Description of the 1-D geometry used for heterogeneous XS treatment

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
geometry_type	KeyedValue	1	String	Choices cylinder slab	[required] Type of 1-D geometry
max_mesh_size	KeyedValue	1	Real	Range (0.0,+INF)	[required] Number in meters of the maximum 1d mesh size
boundary_condition	KeyedValue	0 or 1	String	Choices reflective periodic	
external_composition	KeyedValue	0 or 1	String	InputKeys cell id	Another cell's ID which fissile composition is used as surrounding material (required for non-fissile cell)
r_location	Array of Size 2 or more	1	String	InputKeys plane id cylinder id	[required] Surface boundaries of the different areas
r_composition	Array of Size 2 or more	1	String	InputKeys material id	[required] Material in each area defined by r_location

/arc/calculations/mcc3/rzmflx_code_options

Description of the 2-D geometry and code options

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
bottom_boundary_condition	KeyedValue	0 or 1	String	Choices vacuum reflective	
top_boundary_condition	KeyedValue	0 or 1	String	Choices vacuum reflective	
right_boundary_condition	KeyedValue	0 or 1	String	Choices vacuum reflective	
left_boundary_condition	KeyedValue	0 or 1	String	Choices vacuum reflective	
code	KeyedValue	1	String	Choices twodant partisn	
egroupname	KeyedValue	1	String	Choices anl33 anl70 anl230 anl1041 anl2082	[required] Energy group structure (used for 2-D calculation)
R_boundaries	Array of Size 1 or more	0 or 1	String	InputKeys cylinder id	[required for RZ geometry]
R_nodes_distance	KeyedValue	0 or 1	Real	Range (0.0,+INF)	[required for RZ geometry]
Z_boundaries	Array of Size 1 or more	0 or 1	String	InputKeys plane id	[required for RZ geometry]
Z_nodes_distance	KeyedValue	0 or 1	Real	Range (0.0,+INF)	[required for RZ geometry]
X_boundaries	Array of Size 1 or more	0 or 1	String	InputKeys plane id	[required for XY geometry]
X_nodes_distance	KeyedValue	0 or 1	Real	Range (0.0,+INF)	[required for XY geometry]
Y_boundaries	Array of Size 1 or more	0 or 1	String	InputKeys plane id	[required for XY geometry]
Y_nodes_distance	KeyedValue	0 or 1	Real	Range (0.0,+INF)	[required for XY geometry]
SN_angular_order	KeyedValue	1	Integer	Range (0,+INF)	
core_2d_geometry	Array of Size 1 or more	1	String	InputKeys cell id	[required] list of MCC3 cell IDs in each area defined with R/Z/X/Y boundaries

/arc/calculations/mcc3/lumped_element_text_file

Link to text file containing description of a lumped element

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String		
value	Value	1	String		

Name	Type	HowMany	ValueType	Restrictions	Description
power	KeyedValue	1	Real	Range (0.0,+INF)	[required] Neutron power of the full-core [W]
run_dif3d	KeyedValue	0 or 1	String	Choices true false	Option to skip DIF3D stand-alone calculation
geometry_type	KeyedValue	1	String	Choices hexagonal_full_core hexagonal_sixth_core hexagonal_third_core hexagonal_infinite_lattice triangular_full_core cartesian_full_core cartesian_half_core cartesian_fourth_core cartesian_infinite_lattice	[required] Type of geometry modelled and symmetry specification
isotxs	KeyedValue	0 or 1	String		Link to multi-group XS data
max_axial_mesh_size	KeyedValue	0 or 1	Real	Range (0.0,+INF)	[required for Hex-Z or XYZ geometries] maximal axial distance between each calculation node used in the DIF3D model
max_radial_mesh_size	KeyedValue	0 or 1	Real	Range (0.0,+INF)	[required for XY or XYZ geometries] maximal radial distance between each calculation node used in the DIF3D model
num_axial_burnup_zones	TaggedValue	0 or more			Axial subdivision of a sub-assembly (useful if contains depleted materials)
extract_volume_high_fastflux	KeyedValue	0 or 1	Real	Range (0.0,+INF)	Option to extract total volume with fast flux level higher than value provided here in n/cm2-s - requires refined 'num_axial_burnup_zones' in order to be accurate
user_defined_area	SubObject	0 or more			For printout purposes the user can fillout specific region in an area
variant_options	SubObject	0 or 1			Options for VARIANT solver of DIF3D
dif_fd_options	SubObject	0 or 1			Options for Finite Difference solver of DIF3D
dif_nod_options	SubObject	0 or 1			Options for Nodal solver of DIF3D
rebus	SubObject	0 or 1			Description REBUS calculations
persent	SubObject	0 or 1			Description for PERSENT calculations
thcalcs	SubObject	0 or 1			Description for Simple Thermal Hydraulic calculations

[/arc/calculations/dif3d/num_axial_burnup_zones](#)

Axial subdivision of a sub-assembly (useful if contains depleted materials)

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	InputKeys sub_assembly_id sub_assembly_id	
value	Value	1	Integer	Range [2,+INF)	

[/arc/calculations/dif3d/user_defined_area](#)

For printout purposes the user can fillout specific region in an area

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	Integer	Range [0,99]	
ring	KeyedValue	1	Integer	Range [1,+INF)	Ring number as defined in core lattice
index	KeyedValue	1	Integer	Range [1,+INF)	Radial index number as defined in core lattice
axial_index	KeyedValue	1	Integer	Range [1,+INF)	Axial index number - this is the axial location in sub-assembly starting from bottom

[/arc/calculations/dif3d/variant_options](#)

Options for VARIANT solver of DIF3D

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
polynomial_approx_source	KeyedValue	1	Integer	Range [1, polynomial_approx_fluxes]	
polynomial_approx_fluxes	KeyedValue	1	Integer	Range [1,12]	
polynomial_approx_leakages	KeyedValue	1	Integer	Range [0, polynomial_approx_fluxes]	
angular_approx	KeyedValue	1	Integer	Range [1,60]	
anisotropic_scattering_approx	KeyedValue	1	Integer	Range [0, scattering_order]	
omega_acceleration	KeyedValue	0 or 1	String	Choices true false	

/arc/calculations/dif3d/dif_fd_options

Options for Finite Difference solver of DIF3D

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
hex_triangular_subdivision	KeyedValue	0 or 1	Integer	Range [0,+INF)	For triangularz geometry - specifies a triangular subdivision of the assembly mesh

////////////////////////////////////
/arc/calculations/dif3d/dif_nod_options

Options for Nodal solver of DIF3D

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
coarse_mesh_rebalance	KeyedValue	1	String	Choices true false	

////////////////////////////////////
</arc/calculations/dif3d/rebus>

Description REBUS calculations

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
cycle_length	KeyedValue	0 or 1	Real	Range (0.0,+INF)	Cycle length specification [in EFPD] - non-equilibrium problems only
num_subintervals	KeyedValue	0 or 1	Integer	Range [1,+INF)	Number of subintervals per cycle - non-equilibrium problems only
subinterval_length	Array of Size 1 or more	0 or 1	Real	Range (0.0,+INF)	List of subinterval lengths per cycle [in EFPD] - non-equilibrium problems only
num_cycles	KeyedValue	0 or 1	Integer	Range [1,+INF)	Number of cycle - non-equilibrium problems only
shutdown_time_between_cycle	KeyedValue	0 or 1	Real	Range (0.0,+INF)	Shutdown time between each cycle [in days] - non-equilibrium problems only
xs_update_at_each_timestep	KeyedValue	0 or 1	String	Choices true false	Option to re-calculate XS at each timestep - non-equilibrium problems only
equilibrium_search	SubObject	0 or 1			Options for equilibrium problems
decay_chain	SubObject	1			[required] Decay chain description

////////////////////////////////////
[/arc/calculations/dif3d/rebus/equilibrium_search](#)

Options for equilibrium problems

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
enrichment_search	SubObject	0 or 1			
cycle_length_search	SubObject	0 or 1			
external_feed	SubObject	1 or more			[required] External feed isotopic description
reprocessing	SubObject	0 or more			Reprocessing strategy description
path	SubObject	1 or more			[required] Reactor charge/discharge specification for every sub-assembly containing fissile elements

////////////////////////////////////

[/arc/calculations/dif3d/rebus/equilibrium_search/enrichment_search](#)

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
cycle_length	KeyedValue	0 or 1	Real	Range (0.0,+INF)	Cycle length specification [in EFPD]
num_subintervals	KeyedValue	0 or 1	Integer	Range [1,4]	Number of subintervals per cycle
subinterval_length	Array of Size 1 to 4	0 or 1	Real	Range (0.0,+INF)	List of subinterval lengths per cycle [in EFPD]
keff_target	KeyedValue	1	Real	Range (0.0,+INF)	[required] Minimum value of k-effective targeted
enrich_high	KeyedValue	1	Real	Range (0.0,1.0)	[required] Upper range of fissile enrichment for the search
enrich_low	KeyedValue	1	Real	Range (0.0, enrich_high)	[required] Lower range of enrichment for the search
frac_cycle_keff_target_met	KeyedValue	1	Real	Range (0.0,1.0]	[required] Fraction of the cycle where k-effective target must be met
MCC3_XS_iterations	KeyedValue	0 or 1	Integer	Range [0,2]	Number of MCC3/REBUS iterations requested to converge on XS at BOEC

////////////////////////////////////

[/arc/calculations/dif3d/rebus/equilibrium_search/cycle_length_search](#)

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
cycle_length_upper	KeyedValue	1	Real	Range (0.0,+INF)	[required] Upper range of cycle length
cycle_length_lower	KeyedValue	1	Real	Range (0.0, cycle_length_upper)	[required] Lower range of cycle length
num_subintervals	KeyedValue	1	Integer	Range [1,+INF)	Number of subintervals per cycle
keff_target	KeyedValue	1	Real	Range (0.0,+INF)	[required] Minimum value of k-effective targeted
enrich	KeyedValue	1	Real	Range (0.0,1.0)	[required] fissile enrichment for the search
frac_cycle_keff_target_met	KeyedValue	1	Real	Range (0.0,1.0]	[required] Fraction of the cycle where k- effective target must be met

////////////////////////////////////
[/arc/calculations/dif3d/rebus/equilibrium_search/external_feed](#)

[required] External feed isotopic description

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String		
afracs	SubObject	1			
total_volume	KeyedValue	0 or 1	Real	Range (0.0,+INF)	Total volume in reserve for the external feed

////////////////////////////////////
[/arc/calculations/dif3d/rebus/equilibrium_search/external_feed/afracs](#)

How Many: 1

Name	Type	HowMany	ValueType	Restrictions	Description
afrac	TaggedValue	1 or more			

////////////////////////////////////
[/arc/calculations/dif3d/rebus/equilibrium_search/external_feed/afracs/afrac](#)

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	InputKeys decay_chain list_isotopes decay_chain list_lumped_elements	
value	Value	1	Real	Range [0.0,1.0]	

[/arc/calculations/dif3d/rebus/equilibrium_search/reprocessing](#)

Reprocessing strategy description

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String		
reprocessing_time	KeyedValue	0 or 1	Real	Range [0.0,+INF)	
storage_time	KeyedValue	0 or 1	Real	Range [0.0,+INF)	
recovery_frac_enrich	TaggedValue	1 or more			[required] Fraction of isotope recovered and defined as enrich material
recovery_frac_other	TaggedValue	0 or more			Fraction of isotope recovered and defined as other material

[/arc/calculations/dif3d/rebus/equilibrium_search/reprocessing/recovery_frac_enrich](#)

[required] Fraction of isotope recovered and defined as enrich material

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	InputKeys decay_chain list_isotopes	
value	Value	1	Real	Range [0.0,1.0]	

[/arc/calculations/dif3d/rebus/equilibrium_search/reprocessing/recovery_frac_other](#)

Fraction of isotope recovered and defined as other material

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	InputKeys decay_chain list_isotopes decay_chain list_lumped_elements	
value	Value	1	Real	Range [0.0,1.0]	

=====

[/arc/calculations/dif3d/rebus/equilibrium_search/path](#)

[required] Reactor charge/discharge specification for every sub-assembly containing fissile elements

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	InputKeys sub_assembly_id sub_assembly_id	
max_burnup	KeyedValue	0 or 1	Real	Range (0.0,1.0)	Maximum burnup in atom percent (only enrichment search) - REBUS will adjust cycle length to meet this constrain
fabr_mat_wdens	KeyedValue	1	Real	Range (0.0,+INF)	[required] Weight density of the heavy nuclei fabricated material
num_batches	KeyedValue	1	Integer	Range [1,+INF)	[required] Number of batches in core for the sub-assembly before discharge
fabrication_time	KeyedValue	0 or 1	Real	Range [0.0,+INF)	
enrich_modif_fact	KeyedValue	1	Real	Range [0.0,1.0]	[required] Factor modifying the enrichment in one path relative to another
feeds_enrich	Array of Size 1 or more	1	String	InputKeys reprocessing_id external_feed_id	[required] List of preferred feeds (by preference order) for the enriched material
feeds_other	Array of Size 1 or more	0 or 1	String	InputKeys reprocessing_id external_feed_id	List of preferred feeds (by preference order) for the other material

=====

[/arc/calculations/dif3d/rebus/decay_chain](#)

[required] Decay chain description

How Many: 1

Name	Type	HowMany	ValueType	Restrictions	Description
list_isotopes	Array of Size 1 or more	1	String	Choices REF: CompNames REF: NuclidIds	[required] List of all isotopes part of the decay chain (heavy nuclei and fission products if explicitly defined)
list_lumped_elements	Array of Size 1 or more	0 or 1	String	InputKeys lumped element text file id	List of all lumped elements part of the decay chain
list_dummy_elements	Array of Size 1 or more	0 or 1	String		List of all dummy elements part of the decay chain
text_file	KeyedValue	1	String		[required] Path to decay chain file that provides the decay chain in REBUS-3 format

[/arc/calculations/dif3d/persent](#)

Description for PERSENT calculations

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
rebus_output_file	KeyedValue	0 or 1	String		Path to REBUS output file - allows to skip REBUS calculation
dif3d_refcalc_depsteps	Array of Size 1 or more	0 or 1	Integer	Range [0,99]	List of depletion steps to run preliminary stand-alone reference DIF3D calculations to provide flux files to speed-up recurring PERSENT calculations
variant_options	SubObject	0 or 1			[required if not specified in DIF3D block] Description of VARIANT options
pert_calc	SubObject	0 or more			Description of Perturbation calculation
sens_calc	SubObject	0 or more			Description of Sensitivity calculation

[/arc/calculations/dif3d/persent/variant_options](#)

[required if not specified in DIF3D block] Description of VARIANT options

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
polynomial_approx_source	KeyedValue	1	Integer	Range (-INF,+INF) InputKeys variant options polynomial_approx_fluxes	
polynomial_approx_fluxes	KeyedValue	1	Integer	Range [1,12]	
polynomial_approx_leakages	KeyedValue	1	Integer	Range [0, polynomial_approx_fluxes]	
angular_approx	KeyedValue	1	Integer	Range [1,60]	
anisotropic_scattering_approx	KeyedValue	1	Integer	Range [0, scattering_order]	
omega_acceleration	KeyedValue	0 or 1	String	Choices true false	

[/arc/calculations/dif3d/persent/pert_calc](#)

Description of Perturbation calculation

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	InputKeys material id	
list_isotopes_perturbed	Array of Size 1 or more	1	String	Choices all_from_material REF: CompNames REF: NuclideIds InputKeys lumped element text file id	
density_perturbation_factor	KeyedValue	0 or 1	Real	Range (0.0,+INF)	factor by which the density of the isotopes perturbed will be multiplied
new_temp	KeyedValue	0 or 1	Real	Range (0.0,+INF)	new temperature of the isotopes perturbed (requires xs_updated = true)

[/arc/calculations/dif3d/persent/pert_calc/surface_perturbation](#)

Description of a perturbation on a surface

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	InputKeys hexagon id square id cylinder id plane id	
value	Value	1	String	InputKeys hexagon id square id cylinder id plane id	

[/arc/calculations/dif3d/persent/sens_calc](#)

Description of Sensitivity calculation

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	Choices a b c d e f g h i j k m n o p q t u v w x y z	
depletion_step	KeyedValue	1	Integer	Range [0,+INF)	[required] Time step number of the REBUS depletion calculation
xs_updated	KeyedValue	1	String	Choices true false	[required] Indicate whether cross-sections will be recalculated for the perturbed state and for the reference state of depletion_step>0
sensitivity_factor	KeyedValue	1	Real	Range (0.0,+INF)	[required] Factor by which the cross-sections are perturbed for the sensitivity analysis
sens_type	KeyedValue	1	String	Choices keff beta lambda drho	[required] Describes the type of sensitivity problem specified
sensitivity_regions	Array of Size 1 or more	0 or 1	String	InputKeys cell id	List of XS cells on which the sensitivities are calculated - default is all
list_isotopes	Array of Size 1 or more	1	String	Choices all_from_sensitivity_regions REF: CompNames REF: NuclideIds InputKeys lumped element text file id	[required] List of isotopes on which the sensitivities are calculated

Name	Type	HowMany	ValueType	Restrictions	Description
list_reactions	Array of Size 1 or more	1	String	Choices fission chi capture elastic inelastic nu n2n p1elastic	[required] List of reactions on which the sensitivities are calculated
covariance_matrix_text_file	KeyedValue	0 or 1	String		[optional] Link to covariance matrix for UQ calculations
material_perturbation	SubObject	0 or more			Description of the type of perturbation on a material
pert_option	KeyedValue	0 or 1	String	Choices general_perturbation_theory	[required for drho problem]
surface_perturbation	TaggedValue	0 or more			Description of a perturbation on a surface

=====

/arc/calculations/dif3d/persent/sens_calc/material_perturbation

Description of the type of perturbation on a material

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	InputKeys material id	
list_isotopes_perturbed	Array of Size 1 or more	1	String	Choices all_from_material REF: CompNames REF: NuclideIds InputKeys lumped element text file id	
density_perturbation_factor	KeyedValue	0 or 1	Real	Range (0.0,+INF)	factor by which the density of the isotopes perturbed will be multiplied
new_temp	KeyedValue	0 or 1	Real	Range (0.0,+INF)	new temperature of the isotopes perturbed (requires xs_updated = true)

=====

/arc/calculations/dif3d/persent/sens_calc/surface_perturbation

Description of a perturbation on a surface

How Many: 0 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	InputKeys hexagon id square id cylinder id plane id	
value	Value	1	String	InputKeys hexagon id square id cylinder id plane id	

////////////////////////////////////
[/arc/calculations/dif3d/thcalcs](#)

Description for Simple Thermal Hydraulic calculations

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
dif3d_BOC_output_file	KeyedValue	0 or 1	String		Path to DIF3D output file at BOC
dif3d_EOC_output_file	KeyedValue	0 or 1	String		Path to DIF3D output file at EOC
time_step_BOC	KeyedValue	0 or 1	Integer	Range (-INF,+INF) InputKeys percent dif3d refcalc depsteps	Time step for BOC calculation
time_step_EOC	KeyedValue	0 or 1	Integer	Range (-INF,+INF) InputKeys percent dif3d refcalc depsteps	Time step for EOC calculation
coolant_inlet_temp	KeyedValue	1	Real	Range (0.0,+INF)	Coolant average inlet temperature in Kelvin
coolant_temp_elevation	KeyedValue	1	Real	Range (0.0,+INF)	Average coolant temperature elevation in Kelvin
cladding_type	KeyedValue	1	String	Choices ht9 ods9 ods12	Material type for cladding
bond_type	KeyedValue	1	String	Choices na he	Material type for bond element
fuel_type	KeyedValue	1	String	Choices metal oxide	Material type for fuel
coolant_type	KeyedValue	1	String	Choices na	Material type for coolant
fuel_pin_inner_surf	KeyedValue	0 or 1	String	InputKeys cylinder id	Surface for pin inner dimension
fuel_pin_outer_surf	KeyedValue	1	String	InputKeys cylinder id	Surface for pin outer dimension
cladding_pin_inner_surf	KeyedValue	1	String	InputKeys cylinder id	Surface for cladding inner dimension
cladding_pin_outer_surf	KeyedValue	1	String	InputKeys cylinder id	Surface for cladding outer dimension
wire_wrap_diameter	KeyedValue	1	Real	Range [0.0,+INF)	Dimater of wire wrap in m
duct_inner_surf	KeyedValue	1	String	InputKeys hexagon id	Surface for duct inner dimension
assembly_pitch_surf	KeyedValue	1	String	InputKeys hexagon id	Surface for assembly pitch dimension

Name	Type	HowMany	ValueType	Restrictions	Description
pin_length	KeyedValue	1	Real	Range (0.0,+INF)	Length of the pin on which is the pressure drop calculated, in m
num_pin_per_assembly	KeyedValue	1	Integer	Range (0,+INF)	Number of fuel pin per assembly
assembly_radial_power_fact	KeyedValue	1	Real	Range [1.0,+INF)	Adjustment factor takes into account the variation in power radially within the fuel assembly (to correct from sub-channel TH calculations)
assembly_stage_fact	KeyedValue	1	Real	Range [1.0,+INF)	Adjustment factor takes into account the variation in the power within the assemblies during the equilibrium cycle due to REBUS assumptions
num_channels	TaggedValue	1 or more			Number of channels per assembly definition

[/arc/calculations/dif3d/thcalcs/num_channels](#)

Number of channels per assembly definition

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	InputKeys sub_assembly_id	
value	Value	1	Integer	Range [1,+INF)	

[/lumped_element_external_list](#)

Description of a list of isotopes in a new lumped element

How Many: 0 or 1

Name	Type	HowMany	ValueType	Restrictions	Description
afrac	TaggedValue	1 or more			

[/lumped_element_external_list/afrac](#)

How Many: 1 or more

Name	Type	HowMany	ValueType	Restrictions	Description
id	Tag	1	String	Choices REF: CompNames REF: Nuclidelds	
value	Value	1	Real	Range [0.0,1.0]	

[Start of Input](#)

Referenced Choice Lists

CompNames

ac225 ac226 ac227 ag107 ag109 ag110m ag111 al27 am241 am242 am242m am243 am244 am244m ar36 ar38 ar40 as75 au197 b10 b11 ba130 ba132
ba133 ba134 ba135 ba136 ba137 ba138 ba140 be7 be9 bi209 bk249 bk250 br79 br81 c ca40 ca42 ca43 ca44 ca46 ca48 cd106 cd108 cd110 cd111
cd112 cd113 cd114 cd115m cd116 ce136 ce138 ce139 ce140 ce141 ce142 ce143 ce144 cf249 cf250 cf251 cf252 cf253 cf254 cl35 cl37 cm241 cm242
cm243 cm244 cm245 cm246 cm247 cm248 cm249 cm250 co58 co58m co59 cr50 cr52 cr53 cr54 cs133 cs134 cs135 cs136 cs137 cu63 cu65 dy156
dy158 dy160 dy161 dy162 dy163 dy164 er162 er164 er166 er167 er168 er170 es253 es254 es255 eu151 eu152 eu153 eu154 eu155 eu156 eu157 f19
fe54 fe56 fe57 fe58 fm255 ga69 ga71 gd152 gd153 gd154 gd155 gd156 gd157 gd158 gd160 ge70 ge72 ge73 ge74 ge76 h1 h2 h3 he3 he4 hf174 hf176
hf177 hf178 hf179 hf180 hg196 hg198 hg199 hg200 hg201 hg202 hg204 ho165 ho166m i127 i129 i130 i131 i135 in113 in115 ir191 ir193 k39 k40 k41
kr78 kr80 kr82 kr83 kr84 kr85 kr86 la138 la139 la140 li6 li7 lu175 lu176 mg24 mg25 mg26 mn55 mo100 mo92 mo94 mo95 mo96 mo97 mo98 mo99 n14
n15 na22 na23 nb93 nb94 nb95 nd142 nd143 nd144 nd145 nd146 nd147 nd148 nd150 ni58 ni59 ni60 ni61 ni62 ni64 np235 np236 np237 np238 np239
o16 o17 p31 pa231 pa232 pa233 pb204 pb206 pb207 pb208 pd102 pd104 pd105 pd106 pd107 pd108 pd110 pm147 pm148 pm148m pm149 pm151
pr141 pr142 pr143 pu236 pu237 pu238 pu239 pu240 pu241 pu242 pu243 pu244 pu246 ra223 ra224 ra225 ra226 rb85 rb86 rb87 re185 re187 rh103
rh105 ru100 ru101 ru102 ru103 ru104 ru105 ru106 ru96 ru98 ru99 s32 s33 s34 s36 sb121 sb123 sb124 sb125 sb126 sc45 se74 se76 se77 se78 se79
se80 se82 si28 si29 si30 sm144 sm147 sm148 sm149 sm150 sm151 sm152 sm153 sm154 sn112 sn113 sn114 sn115 sn116 sn117 sn118 sn119 sn120
sn122 sn123 sn124 sn125 sn126 sr84 sr86 sr87 sr88 sr89 sr90 ta181 ta182 tb159 tb160 tc99 te120 te122 te123 te124 te125 te126 te127m te128
te129m te130 te132 th227 th228 th229 th230 th232 th233 th234 ti46 ti47 ti48 ti49 ti50 u232 u233 u234 u235 u236 u237 u238 u239 u240 u241 v w182
w183 w184 w186 xe123 xe124 xe126 xe128 xe129 xe130 xe131 xe132 xe133 xe134 xe135 xe136 y89 y91 zn zr90 zr91 zr92 zr93 zr94 zr95 zr96

[Start of Input](#)

Nuclidelds

1001 1002 1003 2003 2004 3006 3007 5007 5009 5010 5011 6000 7014 7015 8016 8017 9019 11022 11023 12024 12025 12026 13027 14028 14029
14030 15031 16032 16033 16034 16036 17035 17037 18036 18038 18040 19039 19040 19041 20040 20042 20043 20044 20046 20048 21045 22046
22047 22048 22049 22050 23000 24050 24052 24053 24054 25055 26054 26056 26057 26058 27058 27059 28058 28059 28060 28061 28062 28064
29063 29065 30000 31069 31071 32070 32072 32073 32074 32076 33075 34074 34076 34077 34078 34079 34080 34082 35079 35081 36078 36080
36082 36083 36084 36085 36086 37085 37086 37087 38084 38086 38087 38088 38089 38090 39089 39091 40090 40091 40092 40093 40094 40095
40096 41093 41094 41095 42092 42094 42095 42096 42097 42098 42099 42100 43099 44096 44098 44099 44100 44101 44102 44103 44104 44105
44106 45103 45105 46102 46104 46105 46106 46107 46108 46110 47107 47109 47111 48106 48108 48110 48111 48112 48113 48114 48116 49113
49115 50112 50113 50114 50115 50116 50117 50118 50119 50120 50122 50123 50124 50125 50126 51121 51123 51124 51125 51126 52120 52122
52123 52124 52125 52126 52128 52130 52132 53127 53129 53130 53131 53135 54123 54124 54126 54128 54129 54130 54131 54132 54133 54134
54135 54136 55133 55134 55135 55136 55137 56130 56132 56133 56134 56135 56136 56137 56138 56140 57138 57139 57140 58136 58138 58139
58140 58141 58142 58143 58144 59141 59142 59143 60142 60143 60144 60145 60146 60147 60148 60150 61147 61148 61149 61151 62144 62147
62148 62149 62150 62151 62152 62153 62154 63151 63152 63153 63154 63155 63156 63157 64152 64153 64154 64155 64156 64157 64158 64160
65159 65160 66156 66158 66160 66161 66162 66163 66164 67165 68162 68164 68166 68167 68168 68170 71175 71176 72174 72176 72177 72178
72179 72180 73181 73182 74182 74183 74184 74186 75185 75187 77191 77193 79179 80196 80198 80199 80200 80201 80202 80204 82204 82206
82207 82208 83209 88223 88224 88225 88226 89225 89226 89227 90227 90228 90229 90230 90232 90233 90234 91231 91232 91233 92232 92233
92234 92235 92236 92237 92238 92239 92240 92241 93235 93236 93237 93238 93239 94236 94237 94238 94239 94240 94241 94242 94243 94244
94246 95241 95242 95243 95244 96241 96242 96243 96244 96245 96246 96247 96248 96249 96250 97249 97250 98249 98250 98251 98252 98253
98254 99253 99254 99255 100255 1027058 1047110 1048115 1052127 1052129 1061148 1067166 1095242 1095244

[Start of Input](#)

Nuclear Science and Engineering Division

Argonne National Laboratory
9700 South Cass Avenue, Bldg. 208
Argonne, IL 60439

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC