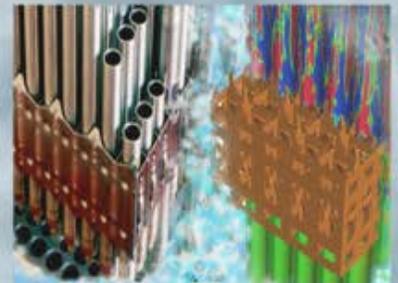
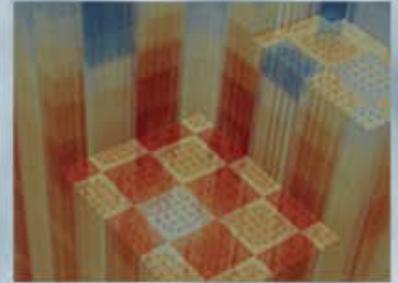


MPACT VERA Common Input User's Manual

Benjamin Collins
Oak Ridge National Laboratory

Brendan Kochunas
University of Michigan

February 20, 2015





VERA Common Input User's Manual

Version 2.0.0

February 20, 2015

Contributors (in alphabetical order)

- Dr. Benjamin Collins (ORNL)
- Prof. Thomas J. Downar (UM)
- Dr. Jess Gehin (ORNL)
- Andrew Godfrey (ORNL)
- Aaron Graham (UM)
- Daniel Jabaay (UM)
- Blake Kelley (UM)
- Dr. Kang Seog Kim (ORNL)
- Dr. Brendan Kochunas (UM)
- Prof. Edward Larsen (UM)
- Dr. Yuxuan Liu (UM)
- Prof. William R. Martin (UM)
- Dr. Scott Palmtag (ORNL)
- Michael Rose (UM)
- Thomas Saller (UM)
- Dr. Shane Stimpson (UM)
- Jipu Wang (UM)
- Dr. Will Wieselquist (ORNL)
- Mitchell T.H. Young (UM)
- Ang Zhu (UM)

Contents

1	Introduction	1
2	Executing MPACT	3
2.1	Standalone Serial Execution	3
2.2	Standalone Parallel Execution	4
3	Input File Structure	5
3.1	VERA Common Input	5
4	Using MPACT with the VERA Common Input	6
4.1	MPACT Block	6
4.1.1	Base Inputs	7
4.1.1.1	checkpoint_mode	7
4.1.1.2	checkpoint_file	7
4.1.1.3	jagged	7
4.1.1.4	ray_spacing	8
4.1.1.5	rod_treatment	8
4.1.1.6	vis_edits	8
4.1.2	2D/1D Inputs	8
4.1.2.1	nodal_method	8
4.1.2.2	split_TL	8
4.1.2.3	TL_treatment	9
4.1.3	CMFD Inputs	9
4.1.3.1	cmfd_angle_decomp	9

4.1.3.2	cmfd	9
4.1.3.3	k_shift	10
4.1.3.4	cmfd_num_outers	10
4.1.3.5	cmfd_solver	10
4.1.3.6	cmfd_up_scatter	10
4.1.4	Iteration Control Inputs	10
4.1.4.1	flux_tolerance	11
4.1.4.2	k_tolerance	11
4.1.4.3	num_inners	11
4.1.4.4	num_outers	11
4.1.4.5	scattering	11
4.1.4.6	up_scatter	11
4.1.5	Meshing Inputs	12
4.1.5.1	mesh	12
4.1.5.2	automesh_bounds	12
4.1.5.3	meshing_method	12
4.1.5.4	axial_mesh	13
4.1.6	Quadrature Set Inputs	13
4.1.6.1	quad_type	13
4.1.6.2	azimuthals_octant	13
4.1.6.3	polars_octant	13
4.1.7	XS Library Inputs	14
4.1.7.1	dep_filename	14
4.1.7.2	dep_substep	14
4.1.7.3	mats_file	14
4.1.7.4	mod_mat	14
4.1.7.5	subgroup_set	15
4.1.7.6	xs_filename	15
4.1.7.7	xs_shielder	15
4.1.7.8	shield_method	15
4.1.7.9	xs_type	15

4.1.8	Parallel Environment Inputs	15
4.1.8.1	par_method	16
4.1.8.2	num_angle	16
4.1.8.3	num_space	16
4.1.8.4	num_threads	16
4.1.8.5	par_file	16
4.1.9	Depletion Inputs	19
4.1.9.1	dep_edit	19
4.1.9.2	dep_filename	19
4.1.9.3	dep_kernel	19
4.1.9.4	depl_time_method	19
4.1.9.5	dep_substep	20
4.1.10	Thermal-Hydraulic Inputs	20
4.1.10.1	coupling_method	20
4.1.10.2	shielder_th	20
4.1.10.3	outers_per_TH	20
4.1.10.4	grid_treatment	21
4.1.10.5	average_ftemp	21
4.1.10.6	ctf_basename	21
4.1.10.7	ith_dhfrac	21
4.1.10.8	ith_hgap	21

Chapter 1

Introduction

The MPACT (**M**ichigan **P**Arallel **C**haracteristics based **T**ransport) code is designed to perform high-fidelity light water reactor (LWR) analysis using whole-core pin-resolved neutron transport calculations on modern parallel-computing hardware. The code consists of several libraries which provide the functionality necessary to solve steady-state eigenvalue problems. Several transport capabilities are available within MPACT including both 2-D and 3-D Method of Characteristics (MOC). A three-dimensional whole core solution based on the 2D-1D solution method provides the capability for full core depletion calculations.

Specific features available in the current release of MPACT are:

- Support for Microsoft Windows Operating Systems (32-bit and 64-bit)
- Support for Linux-based Operating Systems (32-bit and 64-bit)
- OpenMP parallelism for MOC sweeps
- Support for MPACT and AMPX working cross section library formats
- Steady-state eigenvalue calculations using power iteration
- 2-D and 3-D MOC transport solvers
- 2D-1D full core solution
- Depletion capability
- Generalized pressurized water reactor (PWR) geometry
- Export of computational and results mesh to VTK files
- Visualization via ViSiT

The purpose of this document is to provide users with sufficient background to be able to utilize MPACT for PWR design and analysis applications. For a more detailed description of the methods or software design, the reader is directed to the theory and programmer's documentation.

This user document is divided into several chapters. After this introduction, an overview is provided regarding code execution capabilities and limitations in both serial and parallel environments. Finally, a detailed description of the user input is provided.

For specific questions about the use of MPACT, the licensing of the code, or to report bugs users are encouraged to send an email to support@casl.gov. When reporting bugs, users are requested to attach the problematic input to the email and to provide information in the body of the email about the code version, machine, runtime environment and any other relevant details to permit debugging.

Explanation of Notation

In several of the code examples that follow in this document a specific syntax is used which can be described as:

- Words appearing in typewriter font within the normal text, such as `this` indicate the word is a reference to a something that used in an example. - Paragraphs or lines of text in this formatting are examples of usage.
- Words bracketed with '`<`' and '`>`' in examples such as `<token>` indicate a single token for which a value is expected. This value is typically some intrinsic data type such as an integer, string, real, or logical.
- Tokens that are bracketed by '[' and ']' in examples such as `[<token>]` indicate an optional value. Optional values may become nested such as `[<token1> [<token2>]]`
- A vertical bar '|' in an example indicates only one out of the set should be used. The set will be defined by one of the above sets of brackets. For example, `[<token1> | <token2>]` means that only one of `<token1>` and `<token2>` should be entered and that this entry is optional.
- Tokens appended by a '(:) ' indicate an array of values, where the number of ':' indicates the number of dimensions of the array. For example, `<matrix(:, :)>` is a token that is a 2-D array.

Chapter 2

Executing MPACT

Depending on how MPACT was configured, compiled, and installed, it may be executed in serial and/or parallel. The following sections outline the procedures for running the code in either serial or parallel. When run as a standalone analysis tool, MPACT is executed from the command line.

2.1 Standalone Serial Execution

The syntax for MPACT is shown below:

```
$> <path_to_MPACT>/mpact.exe [<input_file> [<output_file> [<log_file>]] | -  
      help]
```

All command line arguments are optional. The meaning of each is described as:

- `-help` - Displays the help message. This message describes the command line arguments and their usage.
- `<input_file>` - The name of the input file to process. If no input file is listed, then MPACT tries to process the file `mpact.inp` in the present working directory. The `<input_file>` may include an absolute or relative path to the file. This file must exist and be readable prior to execution.
- `<output_file>` - The file to use for writing the default output. If no file is listed then a file with `<casename>.out` will be created in the present working directory. In general, if the output file does not exist it will be created, and if it already exists it will be replaced **without** warning. `<output_file>` may include an absolute or relative path. The directory pointed to by the path must exist prior to execution.

- `<log_file>` - The file to use for writing the execution log information. If no file is listed then a file with `<casename>.log` will be created in the present working directory. In general, if the log file does not exist, it will be created, and if it already exists it will be replaced **without** warning. `<log_file>` may include an absolute or relative path. The directory pointed to by the path must exist prior to execution.

2.2 Standalone Parallel Execution

MPACT may only be executed in parallel if a parallel build has been installed. MPACT uses two kinds of parallel models. The first is the shared memory model which is based on the OpenMP standard (<http://www.openmp.org>) and the other is a distributed memory model which is based on the MPI Standard. If the MPACT executable is built with MPI then it is executed differently than in serial, but otherwise the serial description in the previous section is correct. When executing MPACT with MPI the command has the following syntax:

```
$> <mpirun_cmd> [<mpi_options>] <mpact_exe> [<mpact_options>]
```

- `<mpirun_cmd>` - This is the command used to launch MPI executables. This command can vary because different machines may have different implementations of the MPI library installed. Therefore, it is suggested the user consult the documentation for their cluster or workstation for this command and its arguments. For most implementations of MPI (such as OpenMPI, <http://www.open-mpi.org/>) the `<mpirun_cmd>` command is `mpirun`.
- `<mpirun_options>` - These are command line arguments for `<mpirun_cmd>`. Again, the user should consult their machine's documentation for usage.
- `<mpact_exe>` - The name of the MPACT executable that is installed. Typically, one should include the full path to the executable since the parallel execution environment may not have the same PATH setting as the run time environment in which the `<mpirun_cmd>` was invoked.
- `<mpact_options>` - The command line arguments for MPACT. See the - Serial Execution section of this chapter for a complete description.

Chapter 3

Input File Structure

3.1 VERA Common Input

MPACT has two input processors: one for processing the common input that is used with CASL's VERA code suite and another for processing MPACT's native input format.

The VERA common input is an ASCII input file that is processed to create an XML-like (eXtensible Markup Language) input file. The file describes the input hierarchically. - While there are several blocks of inputs created during the file processing, the block described in this manual applies only to the MPACT block of inputs.

Since this file format augments MPACT's base functionality, it requires additional packages for MPACT to be able to read it. These packages are TeuchosWrappersExt and Trilinos, and they expose functions and subroutines that facilitate the reading of data from the XML-like format into MPACT. While the user should not have to work with these details directly, it is important to be aware of the additional requirements in order to be able to efficiently diagnose issues.

One feature of the core simulator is that a single input is used to drive all of the multiphysics codes. The benefits of this approach are that users only need to understand and be proficient with one input, and it also ensures that all codes are working from a single, common geometry description of the problem to reduce errors.

This section describes using the cards that can be specified in the VERA common input to control functionality in MPACT.

Chapter 4

Using MPACT with the VERA Common Input

4.1 MPACT Block

The `MPACT` block in the VERA Common input is used to define all of the MPACT specific parameters needed. The inputs are broken up into various sections and groupings that apply to the same part of the code. The list of these sections are:

- Base Inputs
- 2D/1D Inputs
- CMFD Inputs
- Iteration Control Inputs
- Meshing Inputs
- Quadrature Set Inputs
- XS Library Inputs
- Parallel Environment Inputs
- Depletion Inputs
- Thermal-Hydraulic Inputs

All cards are alphabetized within each of their subsections.

4.1.1 Base Inputs

This section lists the inputs that are placed directly within the MPACT block in the V-ERA Common Input and do not correspond to a particular subsection. The rest of the subsections contain input cards pertaining to specified subsection.

4.1.1.1 `checkpoint_mode`

This card is used to control if the calculation is restarted from a checkpoint file. The options are:

- `T` specifies that the case will be started from a checkpoint file.
- `F` disables initialization of the checkpoint file.
- `W` specifies that a checkpoint file is to be written.
- `R` same as `T`
- `RW` same as `R` but after the checkpoint file is read it can be overwritten during the calculation.
- `I` specifies that a checkpoint file may be written through a user interrupt.

The user can send the interrupt signal to MPACT after execution has begun by creating a file named `MPACT_CHECKPOINT_FILE` in the simulation's working directory. This event causes a checkpoint file to be written after every outer iteration. Likewise, the removal of the file stops the writing of a checkpoint file.

The default is `I`.

```
checkpoint_mode [<T|F|R|W|RW|I>]
```

4.1.1.2 `checkpoint_file`

If a checkpoint file is to be read or written then the user can optionally specify the name of this file. If the name is not specified, then the default is `<CASEID>.mcp`.

```
checkpoint_file [<filename>]
```

4.1.1.3 `jagged`

This card is used to specify if the reflector region will be modeled using a jagged representation or filling the full square of the modeling domain with moderator material. The input type is a logical. The default value is `true`.

```
jagged [<true|false>]
```

4.1.1.4 ray_spacing

This card is used to specify the ray spacing for the MOC rays. The input type is a real. The recommended and default value for this input is 0.05 (cm).

```
ray_spacing [<spacing>]
```

4.1.1.5 rod_treatment

This card is used to toggle whether to use volume weighting for control rods to minimize the impact of control rod cusping in the solution. The input type is a logical. The default value is `false`.

```
rod_treatment [<true|false>]
```

4.1.1.6 vis_edits

This card is used to specify the level of visualization edits desired. There are three options; `none`, `core`, `fsr`. `None` will not print any visualization files. `Core` will only print pin level edits of power for the full core. `FSR` will print all available edits in the code including detailed flat source region edits. Note that the FSR edits will be very large and may consume considerable time to generate the visualization files. The input type is a string. The default value is `core`.

```
vis_edits [<none|core|fsr>]
```

4.1.2 2D/1D Inputs

This section lists the inputs that pertain to the 2D/1D section of the code, more specifically the axial solvers and solution methodology.

4.1.2.1 nodal_method

This card is used to specify the type of nodal axial solver that will be used to solve the 1D portion of the 2D/1D solution. The options include:

The input type is a string. The default value is `nem`.

```
nodal_method [<sanm|nem|nem-mg|sn-0|sn-1|sn-2|sn-3|sp1|sp3|sp5|none>]
```

4.1.2.2 split_TL

This card is used to specify whether transverse leakage splitting will be enabled for a calculation using a 2D/1D method. The input type is a logical. The default value is `true`.

```
split_TL [<true|false>]
```

Input Option	Full Name
SANM	Semi-Analytic Nodal Method
NEM	Nodal Expansion Method
NEM-MG	Multi-Group Nodal Expansion Method
SN-0	Discrete Ordinates with 0th Azimuthal Moment
SN-1	Discrete Ordinates with 1st Azimuthal Moment
SN-2	Discrete Ordinates with 2nd Azimuthal Moment
SN-3	Discrete Ordinates with 3rd Azimuthal Moment
SP1	Simplified Pn 1st Order
SP3	Simplified Pn 3rd Order
SP5	Simplified Pn 5th Order
NONE	Finite-Difference Method

4.1.2.3 TL_treatment

This card is used to specify the type of transverse leakage splitting. The options are `flat` and `lflat`. The `lflat` option checks the total/transport cross section. If the value is below the threshold, leakage will not be put into that region. This process is usually to avoid leakage in the fuel-clad gap. It will then redistribute the leakage to the other regions in that pin. The `flat` option does not perform this checking. The input type is a string. The default for this input is `lflat`.

```
TL_treatment [<flat|lflat>]
```

4.1.3 CMFD Inputs

This section lists the inputs that pertain to CMFD section of the code. In particular, it has inputs for changing the type of CMFD solver, and iteration specifications.

4.1.3.1 cmfd_angle_decomp

This card is used to specify whether or not the angular decomposition processors are used during CMFD setup/solve. The default for this treatment is `true`, and is recommended for better parallel load balancing. The input type is a string.

```
cmfd_angle_decomp [<true|false>]
```

4.1.3.2 cmfd

This card is used to specify which CMFD method will be used. There are three options: `cmfd`, `ycmfd`, and `none`. The `cmfd` option is the standard CMFD method. The `ycmfd` option is a modified CMFD method that enhances the stability of nonlinear iteration scheme. The `none` option disables CMFD. The `none` option can only be used in a 2-D problem. CMFD must be present for every 3-D problem because it is how the axial solution is computed. The input type is a string type. The default for this input is `cmfd`.

```
cmfd [<cmfd|ycmfd|none>]
```

4.1.3.3 k_shift

This card is used to specify a shifted eigenvalue problem which is only available with CMFD on and MGNode set as the solver type. A note about using `k_shift` is the value should be LARGER than the eigenvalue of the system. Even a value of 2 would provide some enhanced convergence properties over not using `k_shift`. The input type is a real. The default value is 0.0.

```
k_shift [<k_shift>]
```

4.1.3.4 cmfd_num_outers

This card is used to specify the number of outer eigenvalue iterations to perform during a CMFD acceleration calculation. The maximum value is always 200 in the first outer eigenvalue iteration regardless of user input. The input type is an integer. The default value is 50 for all subsequent outer eigenvalue iterations.

```
cmfd_num_outers [<n_outers>]
```

4.1.3.5 cmfd_solver

This card is used to specify which CMFD solver methodology will be used to solve the multigroup problem. Three options are currently available: `lgsweep`, `mgnode`, and `mggroup`. The `lgsweep` option sweeps through all of the energy groups one by one. The `mgnode` and `mggroup` options set up a full multigroup CMFD matrix in Node or Group major ordering respectively. The input type is a string type. The default value is `lgsweep`.

```
cmfd_solver [<lgsweep|mgnode|mggroup>]
```

4.1.3.6 cmfd_up_scatter

This card is used to specify the number of upscatter iterations when doing `lgsweep` CMFD. The input type is an integer. The default value is 2.

```
cmfd_up_scatter [<n_upscat>]
```

4.1.4 Iteration Control Inputs

This section lists the inputs that pertain to the `iteration_control` section of the code. These options control the iteration of the transport solution.

4.1.4.1 flux_tolerance

This card is used to specify the tolerance on the convergence of the 2-norm of the flux. The input type is a real. The default value is 1.0E-4.

```
flux_tolerance [<flux_tol>]
```

4.1.4.2 k_tolerance

This card is used to specify the tolerance on convergence of the eigenvalue. The input type is a real. The default value is 1.0E-5.

```
k_tolerance [<k_tol>]
```

4.1.4.3 num_inners

This card is used to specify the number of transport sweeps done during group sweeping every outer iteration. The input type is an integer. The default value is 3.

```
num_inners [<n_inner>]
```

4.1.4.4 num_outers

This card is used to specify the maximum number of outer eigenvalue iterations. The input type is an integer. The default value is 500.

```
num_outers [<n_outer>]
```

4.1.4.5 scattering

This card is used to specify what scattering method to use. P0-P5 are available along with TCP0 which is the standard P1 transport corrected P0 scattering method. The LTCP0 option is a limited transport corrected P0. The input type is a string type. The default value is TCP0.

```
scattering [<P0|TCP0|LTCP0|P1|P2|P3|P4|P5>]
```

NOTE: Both the TCP0 and LTCP0 options currently provide transport-corrected P0 solutions; however, LTCP0 will truncate any cross-section values encountered above 1 MeV.

4.1.4.6 up_scatter

This card is used to specify the number of upscattering iterations that occur during group sweeping. The input type is an integer. The default value is 2.

```
up_scatter [<upscatter>]
```

4.1.5 Meshing Inputs

This section lists the inputs that pertain to the mesh section of the code. The inputs here specify the axial mesh and the pin meshes for the problem.

4.1.5.1 mesh

This card is used to specify the radial and azimuthal mesh for each cell. Currently two cell types are used: fuel and gtube. Cells containing fuel materials are flagged to use the fuel mesh and all other cells use the gtube meshing. Currently insert, control, and detector rods have predefined mesh that cannot be overwritten. The input types are a string followed by two arrays of integers separated by a '/'.

```
mesh [fuel | gtube] <num_rad(:)> / <num_azi(:)>
```

where num_rad is the number of radial subdivisions in each ring specified in the cell and num_azi is the number of azimuthal regions in each sub-divided radial ring. In both cases, the last entry will be used for any remaining unspecified regions.

4.1.5.2 automesh_bounds

This card specifies the minimum and maximum desired axial mesh for the auto axial meshing. Any mesh region larger than the specified value will be broken up into smaller regions that are less than the maximum and larger than the minimum. Any mesh region smaller than the specified value will be homogenized and added to a neighboring mesh region until the value is above the minimum and below the maximum. The default value for this card when automeshing is enabled is 2.0 for the minimum and 10.0 for the maximum. The input type is two reals.

```
automesh_bounds [<min> <max>]
```

4.1.5.3 meshing_method

This card specifies the type of axial meshing to be used. The `useraxialmesh` option requires the use of the axial mesh card and no auto meshing is performed in this instance. The `matbound` option calculates the axial mesh just at the axial material boundaries of the problem, no further meshing is performed. The `nonfuel` option will take the material boundaries and automesh the regions below and above the fuel. The `all` option will take the material boundaries and automesh all regions. Both `nonfuel` and `all` options will use the minimum and maximum bounds specified (or default values) to determine the sizing. If this card is not present, the method will default to `useraxialmesh` if the `axial_mesh` card is present or it will default to `matbound` if the `axial_mesh` card is not present. The input type is a string type.

```
meshing_method [<useraxialmesh|matbound|nonfuel|all>]
```

4.1.5.4 axial_mesh

This card is used to specify the axial mesh used in the 2D1D simulation. For a 3-D problem, the input is an array of the thickness values of each axial section the user wishes to model. The input type is an array of reals. There is no default value. The axial mesh card is optional if the `meshing_method` card specifies an option other than `useraxialmesh`.

```
axial_mesh [<plane_thickness(:)>]
```

4.1.6 Quadrature Set Inputs

This section lists the inputs that pertain to the `quad_set` section of code. The inputs in this section control the transport "ray" discretization and number of angles for the transport solution.

4.1.6.1 quad_type

This card is used to specify the name of the angular quadrature to use for determining the angles at which the rays are traced throughout the problem. The table below shows the acceptable combinations of quadratures and orders. The input type is a string type. The default value is `CHEBYSHEV-CHEBYSHEV`.

Quadrature Name	Type	Order	Order Θ
CHEBYSHEV-CHEBYSHEV	Product	integers > 0	integers > 0
CHEBYSHEV-GAUSS	Product	integers > 0	integers > 0
CHEBYSHEV-BICKLEY	Product	integers > 0	1, 2, 3, or 4
CHEBYSHEV-YAMAMOTO	Product	integers > 0	1, 2, or 3
LEVEL-SYMMETRIC	General	even integers in [2,16]	N/A
QUADRUPLE-RANGE	Product	integers in [1,37]	integers in [1,18]

```
quad_type [<quad_type>]
```

4.1.6.2 azimuthals_octant

This card is used to specify the number of azimuthal angles per octant and corresponds to the Order column in the table above. The input type is an integer. The default value is 4.

```
azimuthals_octant [<num_azi>]
```

4.1.6.3 polars_octant

This card is used to specify the number of polar angles per octant and corresponds to the Order Θ column in the quadrature table above. Note that the number of polar angles

may be limited by the quadrature type used. Also, any non-product quadrature types will not use this input card (e.g. `LEVEL-SYMMETRIC`). The input type is an integer. The default value is 4.

```
polars_octant  [<num_pol>]
```

4.1.7 XS Library Inputs

This section lists the inputs that pertain to the `xs_library` section of code. The inputs here specify the name of the cross section library and its format, as well as the resonance shielding parameters.

4.1.7.1 `dep_filename`

This card is used to specify the name of the depletion file to use. The input type is a string type. There is no default value.

```
dep_filename  [<filename>]
```

4.1.7.2 `dep_substep`

This card is used to specify the number of substeps used in depletion. The input type is an integer. The default value is 1.

```
dep_substep  [<nsubstep>]
```

4.1.7.3 `mats_file`

This card is used to specify the name of the HDF5 material database file. This file is used to overwrite the isotopic and weight fraction values for typical VERA material types. The input type is a string type. There is no default value.

```
mats_file    [<filename>]
```

4.1.7.4 `mod_mat`

This card is used to rename the moderator material. The input type is a string type. The default value is `mod`.

```
mod_mat      <name>
```

4.1.7.5 subgroup_set

This card is used to specify the subgroup set. For most cases, 4 should be used, which is the default. In general, the numbering is from 1 to 9, with 1 being the most general and simplest set, and 9 being the most explicit set. The input type is an integer.

```
subgroup_set [<set>]
```

4.1.7.6 xs_filename

This card is used to specify the name of the cross-section file to use. The input type is a string type. There is no default for this input.

```
xs_filename [<filename>]
```

4.1.7.7 xs_shielder

This card is used to specify whether to shield the cross sections or not. This card allows two ways to specify the input as enabled or disabled. The input type is a string type. The default value is `true`.

```
xs_shielder [<t|f|true|false>]
```

4.1.7.8 shield_method

This card is used to specify the method used to shield the cross sections. The input options are `subgroup` and `essm`. The input type is a string type. The default value is `subgroup`.

```
shield_method [<subgroup|essm>]
```

4.1.7.9 xs_type

This card is used to specify the type of cross-section file to use. Currently, only `ORNL` is available. The input type is a string type. The default value is `NONE`.

```
xs_type ORNL
```

4.1.8 Parallel Environment Inputs

This section lists the inputs that pertain to the `parallel_env` section of code. The inputs here specify the number of processors to be used in various parallel decomposition schemes.

4.1.8.1 par_method

This card is used to specify the method of parallel decomposition. Currently, only `DEFAULT` and `EXPLICITFILE` are accepted. The input type is a string type.

```
par_method [<DEFAULT|EXPLICITFILE>]
```

4.1.8.2 num_angle

This card is used to specify the number of angle decomposition regions used in parallel execution. The angle parallelism duplicates the spatial domain. The input type is an integer. The default value is 1.

```
num_angle [<n_angle>]
```

4.1.8.3 num_space

This card is used to specify the number of spatial decomposition regions used in a parallel execution step. Depending on the method specified, this value can range from a subset of the number of planes in the model, to all of the planes, to all of the planes and any number of radial regions comprised of groups of quarter assemblies. The ability to decompose a problem by planes can be used with the `DEFAULT` partition method. Any partition that decomposes the problem radially requires the `EXPLICITFILE` partition method. The input type is an integer. The default value is 1.

```
num_space [<n_space>]
```

4.1.8.4 num_threads

This card is used to specify the number of threads used in parallel execution. The number of threads specified are used only during the MOC transport sweep. The input type is an integer. The default value is 1.

```
num_threads [<n_threads>]
```

4.1.8.5 par_file

This card is used to specify the parallel decomposition file if `EXPLICITFILE` is used. This is an advanced feature that is not recommended for most users. The input type is a string type. The default value is `partition.txt`.

```
par_file [<filename>]
```

The file structure itself has two header lines followed by the specification of the radial partition regions.

The first line has 3 values, the first is the number of MPACT ray trace modules in the x direction, the second is the number of ray trace modules in the y direction, and the third is the number of axial planes in the model.

The second line also has 3 values. The first two pertain specifically to how MPACT partitions ray trace modules in space, and these values should always be 0 and 1 respectively. The third value should be the number of radial partitions being subsequently specified.

The following lines should describe all radial partition regions for the problem including any regions that will be used with a jagged core. The input for each line is 6 integers. The first pair of integers are the starting and stopping module indices in the x direction, the second pair are the starting and stopping module indices in the y direction, and the last pair is for the z direction, but they are ignored currently and all radial partitions are assumed to be the same for each axial plane. The coordinate system point of origin when specifying the starting and stopping indices is the lower left (south-west) corner of the module. When specifying the starting and stopping indices, it is important to note that these are not necessarily the assembly positions. Typically, in the case of modeling a full reactor, the ray trace modules represent a quarter of an assembly. If this situation is the case, the number of ray trace modules in a given direction will be about twice the number of assemblies in that direction.

No comments are allowed in the file.

Also, it may be unclear to the user how many planes will be created in MPACT before the case is run. The output file has a summary of the Axial Mesh information, including the total number of planes. If the case crashes when using the partition file, check that the number of planes specified matches the value in the output file.

A sample explicit file is given below:

```

17 17 58
0 1 49
 1 3 1 2 1 1
 4 6 1 2 1 1
 7 9 1 2 1 1
 1 3 3 4 1 1
 4 6 3 4 1 1
 7 9 3 4 1 1
10 11 3 4 1 1
12 13 3 4 1 1
 1 3 5 6 1 1
 4 6 5 6 1 1
 7 9 5 6 1 1
10 11 5 6 1 1
12 13 5 6 1 1
14 15 5 6 1 1
 1 3 7 8 1 1
 4 6 7 8 1 1
 7 9 7 8 1 1
10 11 7 8 1 1
12 13 7 8 1 1
14 15 7 8 1 1

```

```

1 3 9 11 1 1
4 6 9 11 1 1
7 9 9 11 1 1
10 11 9 11 1 1
12 13 9 11 1 1
14 15 9 11 1 1
16 17 9 11 1 1
1 3 12 14 1 1
4 6 12 14 1 1
7 9 12 14 1 1
10 11 12 14 1 1
12 13 12 14 1 1
14 15 12 14 1 1
16 17 12 14 1 1
1 3 15 17 1 1
4 6 15 17 1 1
7 9 15 17 1 1
10 11 15 17 1 1
12 13 15 17 1 1
14 15 15 17 1 1
16 17 15 17 1 1
10 11 1 2 1 1
12 13 1 2 1 1
14 15 1 2 1 1
16 17 1 2 1 1
14 15 3 4 1 1
16 17 3 4 1 1
16 17 5 6 1 1
16 17 7 8 1 1

```

The radial partition map specified by the above example will look like the following:

```

34 34 34 35 35 35 36 36 36 37 37 38 38 39 39 40 40 !17
34 34 34 35 35 35 36 36 36 37 37 38 38 39 39 40 40 !16
34 34 34 35 35 35 36 36 36 37 37 38 38 39 39 40 40 !15
27 27 27 28 28 28 29 29 29 30 30 31 31 32 32 33 33 !14
27 27 27 28 28 28 29 29 29 30 30 31 31 32 32 33 33 !13
27 27 27 28 28 28 29 29 29 30 30 31 31 32 32 33 33 !12
20 20 20 21 21 21 22 22 22 23 23 24 24 25 25 26 26 !11
20 20 20 21 21 21 22 22 22 23 23 24 24 25 25 26 26 !10
20 20 20 21 21 21 22 22 22 23 23 24 24 25 25 26 26 !9
14 14 14 15 15 15 16 16 16 17 17 18 18 19 19 48 48 !8
14 14 14 15 15 15 16 16 16 17 17 18 18 19 19 48 48 !7
8 8 8 9 9 9 10 10 10 11 11 12 12 13 13 47 47 !6
8 8 8 9 9 9 10 10 10 11 11 12 12 13 13 47 47 !5
3 3 3 4 4 4 5 5 5 6 6 7 7 45 45 46 46 !4
3 3 3 4 4 4 5 5 5 6 6 7 7 45 45 46 46 !3
0 0 0 1 1 1 2 2 2 41 41 42 42 43 43 44 44 !2
0 0 0 1 1 1 2 2 2 41 41 42 42 43 43 44 44 !1

! 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

```

If the core is jagged, additional attention is required to keep track of the actual number of processors being used by MPACT. Even though the non-existent assemblies are "partitioned" in the explicit file, nothing there will be run. So the user cannot simply take

the third value from the second line and multiply it by the third value from the first line to get the total number of processors MPACT will use spatially for this case. In the example above, the third value from the second line must have the number of "jagged" partitions subtracted from it. In this case, the actual number of processors per plane becomes $49 - 8 = 41$. That number can then be multiplied by the number of planes for the to get 2378 processors to input into the `num_space` card.

4.1.9 Depletion Inputs

This section lists the inputs that pertain to the depletion section of the code. The inputs define the solvers and the methods that can be used.

4.1.9.1 `dep_edit`

This card is used to specify if the depletion `isum` and `pnum` files are written. the input type is a string type. The default value is `true`.

```
dep_edit  [<true|false>]
```

4.1.9.2 `dep_filename`

This card is used to specify the name of the depletion file to use. The input type is a string type.

```
dep_filename  [<filename>]
```

4.1.9.3 `dep_kernel`

This card is used to specify the depletion kernel to use. The options are `internal` or `origen`, which simply mean whether to use MPACT's internal depletion method, or to couple with ORIGEN and use it to perform the depletion calculation. The input type is a string type. The default value is `internal`.

```
dep_kernel  [<internal|origen>]
```

4.1.9.4 `depl_time_method`

This card is used to specify the time stepping method in depletion. The options are `p-c` and `semip-c` which stand for predictor-corrector and semi-predictor-corrector methods. The input type is a string. The default value is `p-c`.

```
depl_time_method  [<p-c|semip-c>]
```

4.1.9.5 dep_substep

This card is used to specify the number of substeps used in depletion. The input type is an integer. The default value is 1.

```
dep_substep [<nsubstep>]
```

4.1.10 Thermal-Hydraulic Inputs

This section lists the inputs that pertain to the TH section of code. The inputs here specify the parameters for the TH solution.

4.1.10.1 coupling_method

This card is used to indicate which TH coupling method should be used. The possible options are `internal`, `ctf`, `ctf_external`, or `none`. The `internal` option uses MPACT's internal TH solver. The `ctf` option internally couples COBRA-TF to MPACT, and `ctf_external` couples MPACT and COBRA-TF through the lime interface. The `none` option will run with!> constant input TH conditions. The input for this card is a string. The default value for this card is `internal` if the build does not have COBRA-TF configured. The default is `ctf` if the build is configured with COBRA-TF.

```
coupling method [<internal,ctf,ctf_external,none>]
```

4.1.10.2 shielder_th

This card is used to specify the behavior of the shielding of cross sections during a coupled calculation. The input values are `shield_max_outers`, `shield_min_dT`, and `shield_min_drho`. The first input is the maximum number of outer iterations for which MPACT will still perform cross-section shielding calculations following a TH update. The second input is the minimum change in temperature for which MPACT will still perform cross-section shielding calculations following a TH update. The third and last input is the minimum change in moderator density for which MPACT will still perform cross-section shielding calculations following a TH update. The input types for each are an integer, a real, and a real respectively. The default values for each input are 4 outers, 25.0 K, and 0.005 g/cm³ respectively.

```
shielder_th [<shield_max_outers> <shield_min_dT> <shield_min_drho>]
```

4.1.10.3 outers_per.TH

This card is used to indicate how many outer iterations MPACT should perform before performing an additional TH update. The input for this card is any positive integer. The default value is 1.

```
outers_per.TH [<outers_per.TH>]
```

4.1.10.4 grid_treatment

This card is used to indicate the method of applying the grid structure in a lattice on the mesh. The options are `homogenize`, `equal_mass`, and `equal_thickness`. - The `homogenize` option will take the mass specified in the grid card, calculate the moderator volume of the lattice where the grid is located, and use the two values to compute the density of the material. This option applies the grid material uniformly throughout the lattice. The `equal_thickness` option uses the grid mass and the corresponding grid material density to compute the total grid volume for that lattice. The volume is then used to determine the thickness the grid would be within each pincell, and is modeled as an additional rectangular mesh within the pin. The `equal_mass` option is similar to the `homogenize` option, but the material density is computed locally for each pin cell in the lattice. The input type for this card is a string. The default value is `homogenize`.

```
grid_treatment [<homogenize|equal_mass|equal_thickness>]
```

4.1.10.5 average_ftemp

This card is used to indicate whether average temperatures or radially-dependent temperatures should be apply to the fuel cross-sections regions after a TH update. The input to this card is a boolean value. The default value is `true`.

```
average_ftemp [<true|false>]
```

4.1.10.6 ctf_basename

This card is used to indicate the "basename" of the CTF input files for CTF coupling. The "basename" is the section of the CTF input filename(s) without any extensions. The input to this card is a string. The default value is "deck" when COBRA-TF is run in serial and "pdeck" when COBRA-TF is run in parallel.

```
ctf_basename [<ctf_basename>]
```

4.1.10.7 ith_dhfrac

This card is used to set the fraction of the power which is directly deposited in the moderator in internal TH calculations. It is ignored if feedback is off or if coupling with COBRA-TF is being used. The input is a real between 0.0 and 1.0. The default value is 0.02.

```
ith_dhfrac [<ith_dhfrac>]
```

4.1.10.8 ith_hgap

This card is used to set the gap conductance value for internal TH calculations. It is ignored if feedback is off or if coupling with COBRA-TF is being used. The input is a real in units of $W/m^2 K$. The default value is 4500.0

ith_hgap [<ith_hgap>]