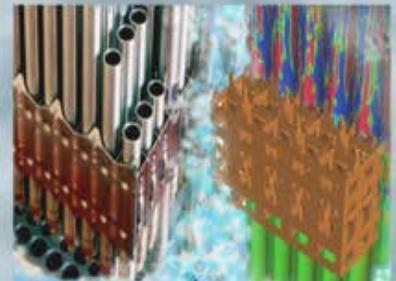
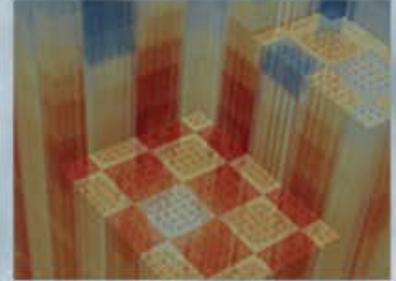


MPACT User's Manual

Version 2.0.0

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

February 20, 2015





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	Native MPACT Input	5
4	Description of MPACT Native Input Blocks and Cards	7
4.1	CASEID Block	7
4.2	DEPL Block	7
4.2.1	KERNEL Card	8
4.2.2	SUBSTEP Card	8
4.2.3	T_UNIT Card	8
4.2.4	TIME_STEP_METHOD Card	8
4.2.5	RESOXS_DT Card	8
4.2.6	BURNUP_DT Card	8
4.3	EDIT Block	9
4.3.1	PNUM Card	9
4.3.2	ISUM Card	9
4.3.3	CHECKPOINT Card	9
4.3.4	EDT_BURN Card	9
4.4	GEOM Block	9

4.4.1	FILE Card	10
4.4.2	MOD_DIM Card	10
4.4.3	PINMESH Card	11
4.4.4	PIN Card	14
4.4.5	MODULE Card	15
4.4.6	LATTICE Card	15
4.4.7	ASSEMBLY Card	16
4.4.8	CORE Card	17
4.5	MATERIAL Block	17
4.5.1	MAT Card	17
4.6	OPTION Block	19
4.6.1	BOUND_COND Card	20
4.6.2	ITER_LIM Card	20
4.6.3	CONV_CRIT Card	20
4.6.4	SOLVER Card	21
4.6.5	RAY Card	21
4.6.6	PARALLEL Card	22
4.6.7	VIS_EDITS Card	24
4.6.8	EXP_TABLE Card	24
4.6.9	VALIDATION Card	24
4.6.10	SCATT_METH Card	24
4.6.11	CMFD Card	25
4.6.12	BC_AVERAGE Card	26
4.6.13	UNDERRELAX Card	26
4.6.14	NODAL Card	26
4.6.15	POWER_EDIT Card	27
4.6.16	PARAM Card	28
4.6.17	VERBOSITY Card	28
4.6.18	AMPX_OUT Card	28
4.6.19	CRITBORON Card	28
4.6.20	XENON Card	29

4.6.21	RESTART Card	29
4.6.22	AXIAL_TL Card	30
4.6.23	INTERNALTH Card	30
4.6.24	CPM_INIT Card	30
4.7	STATE Block	31
4.7.1	RATED_POWER Card	31
4.7.2	CORE_POWER Card	31
4.7.3	RATED_FLOW Card	31
4.7.4	TINLET Card	31
4.7.5	BORON Card	32
4.7.6	PRESSURE Card	32
4.8	XSEC Block	32
4.8.1	XSLIB Card	32
4.8.2	ADDPATH Card	33
4.8.3	XSSHIELDER Card	33

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 Native MPACT Input

One can use MPACT's own input processor for reading ASCII formatted input files. The purpose of this chapter is to describe general rules and conventions of how the input file is processed. Some general formatting rules are:

- Input is generally free formatted, unless otherwise indicated by the specific syntax of a CARD. This means continuous blocks of white space are treated as a single block of white space, whether it is spaces, blank lines or tabs.
- All input lines must not exceed 256 characters.
- ``!'` is the comment symbol. In general it may appear at the beginning of a line or at the end of a line of input, unless otherwise indicated by a specific CARD. All text on a line after the comment symbol is ignored.
- ``.'` is a special symbol that indicates the end of input when it appears in the first column. No lines after this symbol are processed.
- ``*'` is a special symbol to indicate a token is to be repeated. The syntax for this symbol is: `"r*n"` where the token `n` is repeated `r` times. `r` must be an integer and `n` may be any single intrinsic type such as an integer, real, logical, or string. As an example, `"3*4"` would repeat integer entry 4 three times.
- If a string input entry contains a space then it must be enclosed by double quotation marks.
- Logicals must be indicated by the single characters `T` for true or `F` for false. These entries are not case sensitive.

- The input is not case sensitive (unless otherwise noted). The main exception to this is that any directories and/or filename are case sensitive. Capitalization is used throughout this document as a way to help the user distinguish words that have special meaning within MPACT.
- When inputting floating point numbers, any of the following formats are acceptable:

```
1
1.
1.0
1.0e-0
1.0e0
1.0e+00
1.0E+0
1.0E+000
1.0d0
```

The native input file is based on the concepts of BLOCKS and CARDS. The input is made up of several BLOCKS; each BLOCK is made up of several CARDS. Some general rules for BLOCKS and CARDS are:

- All BLOCK and CARD names are **NOT** case sensitive.
- All BLOCK names must start *in* the first column.
- All CARD names must start *after* the first column.
- No text must appear on the same line as a BLOCK name with the exception of the `CASEID` block
- BLOCKS may appear in any order.
- CARDS within a block may generally appear in any order within the BLOCK with the exception of the `GEOM` block.

Chapter 4

Description of MPACT Native Input Blocks and Cards

This section describes each of the blocks and cards that can be used in the input.

4.1 CASEID Block

The `CASEID` block is used to define a unique identifying string for the calculation. It is the first block of the input that is processed. It is also the only block that consists of a single line with input on the same line as the block name. It has no cards and must only appear once within a given input file. The case name is used to generate the filenames of any file created during execution of the case. Typically this includes the output file, log file, and visualization files. The syntax for the block is given below.

```
CASEID ``<case_name>'' [``<description>'' !<comment>]
```

The `<case_name>` is a required string argument and the description is optional if a more detailed description of the case is desired. If the `<case_name>` or `<description>` contain spaces then enclosing double quotation marks must be used. As with virtually all input cards, an optional comment may also be entered.

Note that if the case name contains spaces these are replaced by under-score characters for file naming purposes.

4.2 DEPL Block

The `DEPL` block is for specifying depletion input parameters.

4.2.1 KERNEL Card

This card is used to specify depletion kernel for the Depletion Module

```
[kernel <CRAM|BATEMAN|ORIGEN>] [!<comment>]
```

If the card is not specified, the default depletion kernel is BATEMAN.

4.2.2 SUBSTEP Card

This card is used to read the number of SUBSTEP for the Depletion Module

```
substep <user_depsubstep> [!<comment>]
```

4.2.3 T_UNIT Card

This card is used to specify the units for the burnup specified in the input. The default value is "GWDMT" if this card is not specified.

```
[ t_unit <DAYS | EFPD | MWDKG | GWDMT>] [!<comment>]
```

4.2.4 TIME_STEP_METHOD Card

This card is used to read the time step method for the Depletion Module

```
time_step_method [<P-C | None| SemiP-C |others >] [!<comment>]
```

4.2.5 RESOXS_DT Card

This card is used to read the delta time for calling subgroup calculation in the Depletion Module

```
resoxs_dt <user_resoxs_dt> [!<comment>]
```

4.2.6 BURNUP_DT Card

This card is used to read the number of substeps for the Depletion Module. There are two ways to define the time steps: 1 2 3 4 5 or 1:5:1. The combination of both methods is also recognized.

```
BURNUP_DT <t1 t2 t3:t4:t_interval> [!<comment>]
```

4.3 EDIT Block

The EDIT block is for specifying edit input parameters.

4.3.1 PNUM Card

This card is used to control printing isotope number density. The default value is false.

```
PNUM <T|F> [!<comment>]
```

4.3.2 ISUM Card

This card is used to control printing isotope summary. The default value is false.

```
ISUM <T|F> [!<comment>]
```

4.3.3 CHECKPOINT Card

This card is used to control the file name and format of the checkpoint file.

```
CHECKPOINT <format> [<filename>] [!<comment>]
```

4.3.4 EDT_BURN Card

This card is used to control burnup steps to print results. The burnup steps in this card should be within the time steps defined in the depletion block, otherwise the closest time step result will be printed. There are two ways to define the time steps: 1 2 3 4 5 or 1:5:1. The combination of both methods is also recognized.

```
EDT_BURN <t1 t2 t3:t4:t_interval> [!<comment>]
```

4.4 GEOM Block

The GEOM block is used for specifying the model geometry and spatial mesh. It is the most complicated part of the input. It is the fourth block to be processed when reading the input file. It contains the cards: MOD_DIM, PINMESH, PIN, MODULE, LATTICE, ASSEMBLY, CORE, and FILE. At least one instance of each of these cards must appear in the GEOM block, with the exception of the FILE card. Any of the cards may appear an arbitrary number of times within the block. The order of appearance of the cards is also important because the values entered for a card refer to objects that must

be defined in cards that are processed prior to the current card. The cards must appear in the following order within the block.

1. MOD_DIM
2. PINMESH
3. PIN
4. MODULE
5. LATTICE
6. ASSEMBLY
7. CORE

The optional `FILE` card can also appear anywhere in this block, and functions like a direct insertion of the text of the file into the current line. Therefore, care should be taken when using this card that the contents of the file adhere to the order of the overall input.

4.4.1 FILE Card

This card is the only optional card within the `GEOM` block and is included as a convenience. The purpose of this card is to insert the contents of another file into the current line. This has the convenience that for models with very long and complicated geometric descriptions the top level input file may be shortened for readability. The syntax for this card is:

```
file ``<path_file_name>'' [!<comment>]
```

- `<path_file_name>` - is a string containing the full or relative (to the main input file directory) path to the file and the file name with the extension.

4.4.2 MOD_DIM Card

This card is for specifying the dimensions of the ray tracing modules. All three dimensions must be specified regardless of whether using a 2-D or 3-D transport solver.

```
mod_dim <x> <y> <z> [!<comment>]
```

- `<x>` - the size of the x-dimension of all ray tracing modules. Valid values include any positive double-precision value. Units for this value are centimeters.
- `<y>` - the size of the y-dimension of all ray tracing modules. Valid values include any positive double-precision value. Units for this value are centimeters.

- `<z>` - the size of the z-dimension of all ray tracing modules. Valid values include any positive double-precision value. Units for this value are centimeters. The positive z-direction is opposite to the direction of gravitational force.

4.4.3 PINMESH Card

The `PINMESH` card describes the flat source region mesh to use for a pin cell in the reactor. It is the most complicated card of the input. Several types of pin mesh are definable, and they are described below. All pin shapes are cuboids. The type of pin is selected using a mnemonic for each pin type.

Rectangular Mesh Pin: The syntax for defining a pin mesh made up of a 3-D rectilinear grid is:

```
pinmesh <id> rec <xvals(:)> / <yvals(:)> / <zvals(:)> / <ndivx(:)> / <ndivy(:)>
      > / <ndivz(:)> [!<comment>]
```

- `<id>` - an integer id for this pin mesh. It must be unique amongst all `PINMESH` defined in the model. IDs may be nonsequential, and include any positive 32-bit integer.
- `<xvals(:)>` - The x-coordinates marking material interfaces within the pin. The array must have a size of at least 1, and the last value listed is taken to be the pin pitch in the x-direction. All values must be positive reals and they must be listed in ascending order.
- `<yvals(:)>` - The y-coordinates marking material interfaces within the pin. The array must have a size of at least 1, and the last value listed is taken to be the pin pitch in the y-direction. All values must be positive reals and they must be listed in ascending order.
- `<zvals(:)>` - The z-coordinates marking material interfaces within the pin. The array must have a size of at least 1, and the last value listed is taken to be the pin pitch in the z-direction. All values must be positive reals and they must be listed in ascending order. If a 2-D transport method is going to be used, then there should only be one `<zvals(:)>` and `<zvals(:)>=Pz`.
- `<ndivx(:)>` - The number of equally spaced flat source regions to use when dividing the grid defined by `<xvals(:)>`. The number of entries here must equal the number of entries in `<xvals(:)>`. All values are positive integers.
- `<ndivy(:)>` - The number of equally spaced flat source regions to use when dividing the grid defined by `<yvals(:)>`. The number of entries here must equal the number of entries in `<yvals(:)>`. All values are positive integers.
- `<ndivz(:)>` - The number of equally spaced flat source regions to use when dividing the grid defined by `<zvals(:)>`. The number of entries here must

equal the number of entries in `<zvals(:)>`. All values are positive integers. If a 2-D transport method is going to be used, then `<ndivz(:)>=1`.

Centered Cylindrical Mesh Pin: The syntax for defining a pin mesh made up of concentric cylinders is:

```
pinmesh <id> cyl <r(:)> / <pitch> / <zvals(:)> / <ndivr(:)> / <ndiva(:)> / <
  ndivz(:)> [!<comment>]
```

- `<id>` - an integer id for this pin mesh. It must be unique amongst all `PINMESH` defined in the model. IDs may be nonsequential, and include any positive 32-bit integer.
- `<r(:)>` - is an array of radii for indicating the different material interfaces. The array must have a size of at least 1. All values must be positive reals and they must be listed in ascending order.
- `<pitch>` - is the pitch of the pin. This assumes that the pin is square.
- `<zvals(:)>` - The z-coordinates marking material interfaces within the pin. The array must have a size of at least 1, and the last value listed is taken to be the pin pitch in the z-direction. All values must be positive reals and they must be listed in ascending order. If a 2-D transport method is going to be used, then there should only be one `<zvals(:)>` and `<zvals(:)>=Pz`.
- `<ndivr(:)>` - The number of equal-volume rings to use when dividing the concentric cylinders defined by `<r(:)>` into flat source regions. The number of entries here must equal the number of entries in `<r(:)>`.
- `<ndiva(:)>` - The number of equal-angle "pie-slices" to use when dividing each concentric ring defined by `<r(:)>+<ndivr(:)>` into flat source regions azimuthally. The number of entries here must equal the sum of the values in `<ndivr(:)>` plus 1.
- `<ndivz(:)>` - The number of equally spaced flat source regions to use when dividing the grid defined by `<zvals(:)>`. The number of entries here must equal the number of entries in `<zvals(:)>`. All values are positive integers. If a 2-D transport method is going to be used, then `<ndivz(:)>=1`.
- `<xMin>` `<xMax>` `<yMin>` `<yMax>` are optional, and if omitted are assumed to be a half-pitch in each direction (in other words, the pin is centered).

Quarter-Cylindrical Mesh Pin: The syntax for defining a pin mesh made up of concentric cylinders centered at a pin corner is:

```
pinmesh <id> qcyl <r(:)> / <pitch> <iquad> / <zvals(:)> / <ndivr(:)> / <ndiva(
  :)> / <ndivz(:)> [!<comment>]
```

- `<id>` - an integer id for this pin mesh. It must be unique amongst all `PINMESH` defined in the model. IDs may be nonsequential, and include any positive 32-bit integer.
- `<r(:)>` - is an array of radii for indicating the different material interfaces. The array must have a size of at least one. All values must be positive reals and they must be listed in ascending order.
- `<pitch>` - is the pitch of the pin. This assumes that the pin is square.
- `<iquad>` - an integer on [1,4] to indicate which corner to use for the center of rotation. The values correspond to the following corners
 - 1 - south-west corner
 - 2 - south-east corner
 - 3 - north-east corner
 - 4 - north-west corner
- `<zvals(:)>` - The z-coordinates marking material interfaces within the pin. - The array must have a size of at least 1, and the last value listed is taken to be the pin pitch in the z-direction. All values must be positive reals and must be listed in ascending order. If a 2-D transport method is going to be used, then there should only be one `<zvals(:)>` and `<zvals(:)>=Pz`.
- `<ndivr(:)>` - The number of equal-volume rings to use when dividing the concentric cylinders defined by `<r(:)>` into flat source regions. The number of entries here must equal the number of entries in `<r(:)>` minus 1.
- `<ndiva(:)>` - The number of equal-angle "pie-slices" to use when dividing each concentric ring defined by `<r(:)>+<ndivr(:)` into flat source regions azimuthally. The number of entries here must equal the sum of the values in `<ndivr(:)>` plus 1.
- `<ndivz(:)>` - The number of equally spaced flat source regions to use when dividing the grid defined by `<zvals(:)>`. The number of entries here must equal the number of entries in `<zvals(:)>`. All values are positive integers. If a 2-D transport method is going to be used, then `<ndivz(:)>=1`.

General Cylindrical Mesh Pin: The syntax for defining a pin mesh made up of a concentric cylinders centered at (0,0) with arbitrary pin boundaries is:

```
pinmesh <id> gcyl <r(:)> / <xMin> <xMax> <yMin> <yMax> / <zvals(:)> / <ndivr(:)> / <ndiva(:)> / <ndivz(:)> [!<comment>]
```

- `<id>` - an integer id for this pin mesh. It must be unique amongst all `PINMESH` defined in the model. IDs may be nonsequential, and include any positive 32-bit integer.

- $\langle r(\cdot) \rangle$ - is an array of radii for indicating the different material interfaces. The array must have a size of at least one. All values must be positive reals and must be listed in ascending order.
- $\langle xMin \rangle \langle xMax \rangle \langle yMin \rangle \langle yMax \rangle$ - Specify the boundaries of the pin.
- $\langle zvals(\cdot) \rangle$ - The z-coordinates marking material interfaces within the pin. - The array must have a size of at least 1, and the last value listed is taken to be the pin pitch in the z-direction. All values must be positive reals and must be listed in ascending order. If a 2-D transport method is going to be used, then there should only be one $\langle zvals(\cdot) \rangle$ and $\langle zvals(\cdot) \rangle = Pz$.
- $\langle ndiv_r(\cdot) \rangle$ - The number of equal-volume rings to use when dividing the concentric cylinders defined by $\langle r(\cdot) \rangle$ into flat source regions. The number of entries here must equal the number of entries in $\langle r(\cdot) \rangle$ minus 1.
- $\langle ndiv_a(\cdot) \rangle$ - The number of equal-angle "pie-slices" to use when dividing each concentric ring defined by $\langle r(\cdot) \rangle + \langle ndiv_r(\cdot) \rangle$ into flat source regions azimuthally. The number of entries here must equal the sum of the values in $\langle ndiv_r(\cdot) \rangle$ plus 1.
- $\langle ndiv_z(\cdot) \rangle$ - The number of equally spaced flat source regions to use when dividing the grid defined by $\langle zvals(\cdot) \rangle$. The number of entries here must equal the number of entries in $\langle zvals(\cdot) \rangle$. All values are positive integers. If a 2-D transport method is going to be used, then $\langle ndiv_z(\cdot) \rangle = 1$.

4.4.4 PIN Card

The PIN card is for applying a set of materials to a given PINMESH. The syntax for this card is:

```
pin <pin_id> <pin_mesh_id> / <mat_id_list> [!<comment>]
```

- $\langle pin_id \rangle$ - an integer id for this pin. It must be unique amongst all PIN defined in the model. IDs may be nonsequential, and include any positive 32-bit integer.
- $\langle pin_mesh_id \rangle$ - an integer id for a valid pin mesh id previously defined in a PINMESH card.
- $\langle mat_id_list \rangle$ - a list of integer IDs for valid materials defined in the MAT card of the MATERIAL block. The number of entries must be equal to the number of uniform material regions defined in the pin mesh. The order of the entries depends on the mesh type.
 - For Rectangular Mesh Pins the numbers are given for lexicographical ordering (e.g. left-to-right then top-to-bottom).

- For Cylindrical Mesh Pins the numbers are given in a "in-out" manner for each ring in the mesh. Only each specified radius needs a material definition, not sub-regions. If the pin mesh has multiple axial levels, material ids are specified for each level, again, not for sub-levels.
- For Quarter-Cylindrical Mesh Pins the ordering is the same as the Cylindrical Mesh Pins.

4.4.5 MODULE Card

This card is for defining a ray tracing module which is a 3-D array of pins. A 2-D array of pin IDs is input in the card, and then extruded. The pin boundaries must make a structured rectilinear grid within the module and the min/max dimensions of the pin grid must conform with the modular ray tracing dimensions defined in the `MOD_DIM` card. The syntax for the card is:

```
module <id> <nx> <ny> <nz> [!<comment>]
  <pid(1,1)> ... <pid(nx,1)>
  .
  .
  .
  <pid(1,ny)> ... <pid(nx,ny)>
```

- `<id>` - an integer id for this ray tracing module geometry. It must be unique amongst all `MODULE` defined in the model. IDs may be nonsequential, and include any positive 32-bit integer.
- `<nx>` - the number of pins along the x-direction. Must be a positive integer.
- `<ny>` - the number of pins along the y-direction. Must be a positive integer.
- `<nz>` - the number of pins along the z-direction. Must be a positive integer.
- `<pid(:,:)>` - a 2-D array of valid integer pin IDs defined previously with the `PIN` card. This array gets extruded `<nz>` times in the z-direction. The number of entries on a line must equal `<nx>` and the array must span `<ny>` lines.

The ray tracing module is not a natural engineering structure, so as a general guideline when building models to minimize computational resource usage the ray tracing module should represent the *smallest* unit of repeatable geometry in the model. In some special cases this may be a single pin-cell. For most practical cases this may be a quarter-assembly, and for the most general cases this should not exceed a full lattice description of a single assembly.

4.4.6 LATTICE Card

This card is used to define a lattice geometry. Lattice meshes are defined in terms of ray tracing modules and represent the geometry for the full x-y geometrical description

of an assembly. All lattices within the same assembly must have the same values for `<nx>` and `<ny>`.

```
lattice <id> <nx> <ny> [name ] [!<comment>]
<mid(1,1)> ... <mid(nx,1)>
.
.
.
<mid(1,ny)> ... <mid(nx,ny)>
```

- `<id>` - an integer id for this lattice geometry. It must be unique amongst all lattices defined by the `LATTICE` card in the model. IDs may be nonsequential, and include any positive 32-bit integer.
- `<nx>` - the number of pins along the x-direction. Must be a positive integer.
- `<ny>` - the number of pins along the y-direction. Must be a positive integer.
- `<name>` - an optional descriptive name may also be provided for referencing this lattice type. Presently the name is not used by any other part of the code.
- `<mid(:, :)>` - a 2-D array of valid integer ray tracing module IDs that were defined previously with the `MODULE` card. The number of entries on a line must equal `<nx>` and the array must span `<ny>` lines.

4.4.7 ASSEMBLY Card

This card is used for defining an assembly geometry. An assembly is defined in terms of lattices given as a 1-D array.

```
assembly <id> [<name>] [!<comment>]
<lattice_ids(:)>
```

- `<id>` - an integer id for this lattice geometry. It must be unique amongst all lattices defined by the `LATTICE` card in the model. IDs may be nonsequential, and include any positive 32-bit integer.
- `<name>` - an optional descriptive name may also be provided for referencing this assembly type. Presently the name is not used by any other part of the code.
- `<lattice_ids(:)>` - a 1-D array of valid integer lattice IDs that were defined previously with the `LATTICE` card. The number of entries on a line must be the same for all `ASSEMBLY` cards. The assembly is built using the leftmost lattice ID as the bottom and the rightmost as the top.

4.4.8 CORE Card

This card is used for specifying the CORE radial configuration. The syntax for this card is:

```
core [<symmetry_opt>] [!<comment>]
  <assembly_ID_map(:,:)>
```

- <symmetry_opt> - is an optional input indicating the core symmetry. - Presently only 360 symmetry (or no symmetry) is supported.
- <assembly_ID_map(:,:)> - is the 2-D map of the core assembly locations. The map must be internally continuous (i.e. no empty locations inside the boundary), but is allowed to have a "stair-case" like boundary. The values must be valid integer assembly IDs of assemblies defined previously with the ASSEMBLY card.

4.5 MATERIAL Block

The MATERIAL block is for specifying the materials to be used in the calculation. This block is the third block to be processed when the input file is read, and it is required. Currently materials are defined using the MAT card which is described below. This card may appear an arbitrary number of times within the block and must appear at least once.

4.5.1 MAT Card

The MAT card is for specifying the materials that are used in the calculation. In general it requires an identifying number, a type description, and a list of cross section record identifiers. Different materials may use different cross section library data, if more than one library is specified in the XSEC block.

```
mat <id> <type> [<name>] [<temp> <tunit>] [<density> <dunit>] [-> <libname>] [
  (<unit>)] <delim> <xlname> [<num>] [!<comment>]
  [<xlname> <num> [...]] [!<comment>]
  .
  .
  .
  [!<comment>]

  [<xlname> <num> [...]] [!<comment>]
```

Essentially there are two parts to the input of the card, the part before the delimiter symbol (<delim>) and the part after. The part before must include the card name (mat) and the first 2 parameters <id> and <type> which have the following meaning:

- `<id>` - an identifying integer for this material to be used in the `PIN` card. This field is required.
- `<type>` - an integer enumeration indicating the type of this material. This field is required. The enumerations are listed in the table below:

Type Value	Is Fluid	Is Depletable	Has Resonance Data	Is Fuel
0	F	F	F	F
1	T	F	F	F
2	F	T	T	T
3	F	T	T	F
4	F	F	T	F
5	F	T	F	F

The rest of the parameters up to the delimiter are optional and they include the following:

- `<name>` - an optional string name for the material. Names must start with a letter and must not be any of the recognized unit strings (e.g. a material name cannot be `at%`). The default value is `'MAT <id>'`. If specifying a name with space characters in it, it must be enclosed in double-quotes.
- `<temp> <tunit>` - the temperature of the material with the units. The card will accept degrees Celsius (denoted by `'C'`), degrees Fahrenheit (denoted by `'F'`) and degrees Rankine (denoted by `'R'`). The default unit is Kelvin (denoted by `'K'`). The default value is 293.15 K. Absolute temperature must be greater than zero.
- `<density> <dunit>` - the material density with the units. Only default units are implemented. The default unit is g/cc. Default value is 1.0 g/cc. Values must be greater than zero.
- `<libname>` - the XS library file name (without the path and with the file extension) specified in a `XSLIB` card in the `XSEC` block, which should contain the XS records specified for this material. The default library is the first library listed in the `XSEC` block. Note that this name must be preceded by a `'->'` symbol.
- (`<unit>`) - the optional input for signifying whether the given [`<num>`] are number densities, atom, or weight percents. The default units are number densities, and are represented by not having the optional (`<unit>`) input. The other available input units are:
 - (`%at`) or (`at%`) for atom percent
 - (`%wt`) or (`wt%`) for weight percent
 - If either of these values are present in the input, the values will be used to convert the percents into number densities in the code.

Additionally:

- a specified density may appear before the specified temperature, or vice-versa
- each of these parameters may only be specified once per material.
- units must always appear with temperature or density.

An error is thrown if any of the rules for this card are violated.

Two delimiters are allowable. If the XS record is one macroscopic XS type, then the delimiter symbol is ' : '. If more than one XS record constitutes this material, then the delimiter symbol is ' \ '. After the delimiter is a multi-line list of key-value pairs. The key and value must be given as a pair and each key in the list must be unique. There are several allowable formats for the key-value pair based on the units of the provided values, which is given as the next input after the ' \ ' delimiter.

- `<xlname> [<num>]` - is a key-value pair consisting of a XS record identifier and number density pair. `<xlname>` is the XS record identifier which must exist in the XS library associated with this material, it is treated as a string. The `<num>` part of the pair is optional, but if it is present for one `<xlname>` entry, it must be present for all. It is treated as a double-precision value. It must not be used if the ' : ' delimiter is used. If the other delimiter (' \ ') is used then the `<num>` field must be present. This field may occur with a variety of units. The list formats for the respective units that are allowable are:

- Number densities (default) have no unit tag.

```
... \ <xlname1> <num1>
    <xlname2> <num2> ...
```

- Atom percent have the (%at) unit tag before the ' \ ' delimiter.

```
... (%at) \ <xlname1> <num1>
           <xlname2> <num2> ...
```

- Weight percent have the (%wt) unit tag before the ' \ ' delimiter.

```
... (%wt) \ <xlname1> <num1>
           <xlname2> <num2> ...
```

In general, comments can be included at the end of a line, as long as they are the last item on the line. For the multi-line input, all blank lines or lines with just comments are allowed between continuing key-value pairs. The end of the card is not signaled until the next card or block is reached.

4.6 OPTION Block

The `OPTION` block is for specifying optional input parameters for which default values are not desired. This includes options for things like the maximum number of iterations,

convergence control, solver types and meshing parameters. It is the last block to processed when reading the input file. There are several cards each of which must only be used *once* within the block. They are described below.

4.6.1 BOUND_COND Card

This card is used for specifying the boundary conditions on each face of the simulation domain. The syntax is given below

```
bound_cond <west_bc> <north_bc> <east_bc> <south_bc> <top_bc> <bottom_bc>
```

The <bc> values are integers with the following enumerations:

- 0 - vacuum (default)
- 1 - reflective
- 2 - periodic

4.6.2 ITER_LIM Card

This card is used to control the maximum number of iterations for the different levels of iteration.

```
iter_lim <noutermax> <ngsweep> <ninner> [!<comment>]
```

- <noutermax> is the maximum number of outer iterations to perform. This is the same as a fission source iteration. Valid values are any 32-bit integer 1 or greater. The default value is 500. The calculation will terminate once this many iterations has been performed.
- <ngsweep> controls the number of upscatter-sweeps between fission source updates. Valid values are any 32-bit integer 1 or greater. The default value is 2.
- <ninner> the number of 1-group fixed source transport sweeps to perform between scattering source updates. Valid values are any 32-bit integer 1 or greater. The default value is 3.

4.6.3 CONV_CRIT Card

This card is used to define the convergence criteria. The calculation will terminate once all the criteria have been met (or the maximum iterations has been reached).

```
conv_crit <ecritk> <ecritphi> [!<comment>]
```

- `<ecritk>` is the convergence criteria for k-eff where the convergence is measured as the absolute difference in k-eff between successive outer iterations. - Valid values include any positive double-precision value. The default value 1.e-5.
- `<ecritphi>` is the convergence criteria for the scalar flux where the convergence is measured as the L2-norm of the difference of the multi-group scalar flux between successive outer iterations. Valid values include any positive double-precision value. The default is 1.e-4.

4.6.4 SOLVER Card

This card is for specifying the transport solver type.

```
solver <solvertype> <solverdim> [!<comment>]
```

The `<solvertype>` is an integer input with the following enumerations (default is 0):

- 0 - Do all pre-processing as if MOC but do not solve the case.
- 1 - MOC
- 2 - CDP
- 3 - Sn

The `<user_dimtype>` is an integer input with the following enumerations (default is 2):

- 2 - 2-D
- 3 - 3-D

4.6.5 RAY Card

This card is for specifying the angular quadrature and its order and the desired spacing between the characteristic rays.

```
ray [<spacing>] [<quad_name> [<order> [<order_theta>]]] [!<comment>]
```

- `<spacing>` the spacing between characteristic rays in centimeters. Valid values include any positive double-precision value. The default is 0.05 cm.
- `<quad_name>` the name of the angular quadrature to use. The table below shows the acceptable combinations of quadratures and orders. The default quadrature is Chebyshev-Chebyshev

- `<order>` the order of the angular quadrature (or specifically the the azimuthal angle if the quadrature is a product quadrature). The default value is 4.
- `<order_theta>` the order of the polar angle quadrature if a product quadrature is used, otherwise it is not needed. The default value is 4.

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]

4.6.6 PARALLEL Card

This card is used to specify the parallel environment at run time. The applicability of the specifications in this card depends on the problem input, machine, and current parallel domain decomposition algorithm.

```
parallel <nspace> <nangle> <nenergy> <nthreads> [<SpacePartitionType> [<
  SpacePartitionOpts>]] [!<comment>]
```

- `<nspace>` - Integer number of spatial domains to divide the problem into. - Default is 1.
- `<nangle>` - Integer number of angular domains to divide the problem into. - Default is 1.
- `<nenergy>` - Currently not enabled. Integer number of energy domains to divide the problem into. Default is 1.
- `<nthreads>` - Integer number of threads to use per MPI process. How the threads are utilized depends on the solvers. Default is 1. If 0 is entered for the number of threads, then the value returned by the OpenMP routine `OMP_GET_NUM_THREADS()` will be used. Typically this corresponds to the `OMP_NUM_THREADS` environment variable when it is set.
- `<SpacePartitionType>` - String representing the type of partitioning to do in space. Can be one of: "FULLCORE", "ASSEMBLY", "BLOCK", or "FILE". Default is "FULLCORE".
- `<SpacePartitionOpts>` - Options based on previous entry. See note below.

The FULLCORE partitioning method performs binary spatial partitioning (BSP) based on the modular geometry of the full core. This is the default. It does not necessarily give

the best load balance for a given number of processors for every grid, especially those with large prime factors. So if a better load balance is known for a given problem, one of the other methods may provide better parallel performance. It has no additional options for `<SpacePartitionOpts>`.

The ASSEMBLY partitioning method performs BSP first on the 2-D assembly grid of the core, and then BSP within each assembly. This partitioning guarantees that the problem may be run with `<nspace> == n*nasy`. It has no additional options for `<SpacePartitionOpts>`.

The BLOCK partitioning method takes the modular geometry grid of the full core and factors out a sub-block of size `xdim*ydim*zdim`, where each of `xdim`, `ydim`, `zdim` are integer factors of the full core modular geometry grid along the respective axes. This partitioning guarantees that the problem may be run with `<nspace> == nx*ny*nz / (xdim*ydim*zdim)`. BSP is also performed within each sub-block. - The additional options for `<SpacePartitionOpts>` when "BLOCK" is chosen are:

- `xdim` - size of the sub-block on the x-axis in grid units. Must be greater than 0. Default is 1.
- `ydim` - size of the sub-block on the y-axis in grid units. Must be greater than 0. Default is 1.
- `zdim` - size of the sub-block on the z-axis in grid units. Must be greater than 0. Default is 1.

The FILE partitioning method takes the modular geometry grid of the full core and partitions it based on a text file in which the partitioning information is provided. The text file specifies loosely how to construct a tree by specifying subsequent block sizes to "grow" on each leaf node of the tree. To illustrate this say the file has the following:

```
2 2 2
2 2 2
```

This would create a block that is 2x2x2 then grow a tree based on BSP resulting in 8 leaf nodes. From these 8 leaf nodes a new 2x2x2 block would be created, this 2x2x2 block would grow a tree based on BSP at each of the 8 leaf nodes of the existing tree. So this example would produce an oct-tree with 2 levels defining a 4x4x4 grid with 64 nodes.

So each line of the file adds to the line above it and the product of the 3 numbers on the line means there are that many leaf nodes at that level (multiplied with the number of nodes from all previous levels).

The additional option for `<SpacePartitionOpts>` when "FILE" is chosen is the full path name of the file. The default is `./partition.txt` The path may be absolute or relative to the present working directory.

4.6.7 VIS_EDITS Card

This card controls whether or not to edit the visualization files of the geometry/mesh description. A second logical is optional to edit flat source region data on the full domain. Input is a boolean with valid values of T or F. Defaults are false.

```
vis_edits <T|F> [<T|F>] [!<comment>]
```

4.6.8 EXP_TABLE Card

This card controls the type of exponential table or function to be used when evaluating the exponentials in the MOC equations. The LINEAR option uses linear interpolation to evaluate the exponential function. The POLAR option uses linear interpolation as well but also stores all polar angles in the table. The option uses the fortran intrinsic <EXP(x)> function and no lookup table is used. Input is a string with valid values of LINEAR, POLAR, or EXACT. The default value is LINEAR.

```
exp_table [<LINEAR|POLAR|EXACT>] [!<comment>]
```

4.6.9 VALIDATION Card

This card is for developers and is not intended for use by general users.

This card is marked for deprecation.

This card controls whether or not a secondary output will be generated. This output will be formatted for use in validation tests and is only ever needed for these tests. The input is a boolean with valid values of T or F, followed by an optional delimiter character. Acceptable values for this character are S (space) and C (comma). Space delimited is the default.

```
validation <T|F> <Delimiter Character> [!<comment>]
```

4.6.10 SCATT_METH Card

This card is used to specify the scattering order to use in the MOC calculation. The Pn notation implies the number of Legendre scattering moments to use. The higher the value of n the more moments get used and the more accurate the solution. Using more moments can dramatically increase run times. Currently the options are:

- P0
- P1
- P2
- P3

- P4
- P5
- TCP0 (transport corrected P0 approximation)
- LTCPO (limited transport corrected P0 approximation)

TCP0 is the recommended option as it provides improved accuracy with the fastest run times. Not all cross section libraries include higher order scattering data, so the simulation will use the lowest of the specified scattering order and order of the data.

The default is TCP0.

```
scatt_meth <P0|TCP0|LTCPO|P1|P2|P3|P4|P5> [!<comment>]
```

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

4.6.11 CMFD Card

This card controls options related to CMFD. The first parameter specifies the type CMFD acceleration to perform:

- T - Use conventional CMFD acceleration
- F - Disable CMFD acceleration
- I - Initializes solution with CMFD then performs no further CMFD
- Y - specifies an alternative CMFD formulation (research)

When CMFD is used, one may specify the CMFD solution algorithm. These options are:

- 1GSweep - sets up and solves 1-group linear system and performs Gauss-Seidel iteration in energy.
- MGNode - sets up the CMFD linear system for all space and groups with node major ordering, so matrix structure has group x group blocks.
- MGGroup - sets up the CMFD linear system for all space and groups with group major ordering, so matrix structure has nnode x nnode blocks.

If the MGNode solution algorithm is used, then eigenvalue deflation (Wielandt shift) may also be used. To specify the use of eigenvalue deflation set the option to T and specify the shift parameter `k_shift` to use in the deflation.

The default is to use conventional CMFD with 1GSweep and a 0.0 shift.

```
cmfd <T|F|I|Y> [<1GSweep|MGNode|MGGroup> [<T|F> [<k_shift>]]] [!<comment>]
```

4.6.12 BC_AVERAGE Card

This card is for research applications and is not intended for use by the general user. It only applies to the CDP transport kernel.

This card toggles whether or not to perform averaging of the angular flux at cell boundaries. Currently, the default is `<0 0 0>`, which means no average is done on all boundaries. Three integers indicate the number of rays(2D)/planes(3D) to be averaged on X-Y, X-Z and Y-Z faces.

```
bc_average <average_xy> <average_xz> <average_yz> [!<comment>]
```

4.6.13 UNDERRELAX Card

This card is used to specify a fixed under-relaxation coefficient for 2D1D for all energy groups.

```
underrelax <f_ur>
```

The `f_ur` factor is the underrelaxation coefficient. Acceptable values are real numbers greater than 0.0 and less than or equal to 1.0. 1.0 is the default.

4.6.14 NODAL Card

This card is used to specify the nodal method used for the 1-D solution in the 2-D/1-D method. Many of the options have only a research application and are not recommended for use by the general user. The options are:

- NEM - Use 4th order nodal expansion method formulation of diffusion equation.
- SANM - Use semi-analytic nodal expansion method formulation of diffusion equation.
- SP1 - Use NEM formulation of Simplified P1 equations.
- SP3 - Use NEM formulation of Simplified P3 equations.
- SP5 - Use NEM formulation of Simplified P5 equations.
- NEM-MG - (RESEARCH) Use NEM nodal formulation with full-group response matrix.
- Sn-0 - (RESEARCH) Use nodal formulation of Sn transport equations with flat source.
- Sn-1 - (RESEARCH) Use nodal formulation of Sn transport equations with linear source.

- S_n-2 - (RESEARCH) Use nodal formulation of S_n transport equations with quadratic source.
- S_n-3 - (RESEARCH) Use nodal formulation of S_n transport equations with cubic source.
- MOC - (RESEARCH) Use nodal formulation of MOC transport.

The default is NEM.

The remaining options apply to only the S_n kernels and are not recommended for use by the general user. `sn_type` specifies the type of S_n formulation to use with respect to source and transverse leakage treatment in the angular domain. The options are:

- AZIINT-ISOTROPIC
- AZIINT-AZIINT
- EXPLICIT-ISOTROPIC
- EXPLICIT-AZIINT
- EXPLICIT-EXPLICIT
- EXPLICIT-MOMENT
- MOMENT-MOMENT

The `splitRTL` option is a logical for whether or not to split the radial transverse leakage source term to maintain positivity of the total source. This only applies to the S_n and SP_n kernels.

The `nMom` option is used to specify the number of azimuthal moments to use in the EXPLICIT-MOMENT or MOMENT-MOMENT S_n solvers.

```
nodal <T|F> [<NEM|SANM|MOC|NEM-MG|SP1|SP3|SP5|Sn-0...Sn-3>] [<sn_type>] [<
  splitRTL>] [<nMom>] [!<comment>]
```

4.6.15 POWER_EDIT Card

This card is used to specify a cross section used for the "power" calculations `KAPPA--FISSION` is the standard power calculation whereas `FISSION` actually produces the normalized fission reaction rate distribution. The default is `KAPPA-FISSION`.

```
power_edit [<KAPPA-FISSION|FISSION>]
```

4.6.16 PARAM Card

This card is used to allow general access to the `mpactOptionBlockParams` parameter list.

```
param <plist path> <datatype> <val> [!<comment>]
```

- `<plist path>=""` is the path in the parameter list that you wish to specify. For example `foo->value`
- `<datatype>` tells the input processor what type of value you are specifying. The standard MPACT types are allowed (SIK,SLK,SNK,SRK,SSK,SDK and STRING). Any type (with the exception of STRING) may be followed by an 'A,' which will read the following value(s) into an array instead of a scalar
- `<val>` is the value to store in the specified path in the parameter list.

4.6.17 VERBOSITY Card

This card toggles whether or not to report specified exceptions to the log file. Currently, the default is for only errors and fatal errors to be printed to the log. Information, warnings, and debug warnings are suppressed.

```
verbosity [<INFO> <T|F>] [<WARN> <T|F>] [<DEBUG> <T|F>] [!<comment>]
```

4.6.18 AMPX_OUT Card

CURRENTLY DISABLED

This card is used to specify an output file in which to store an AMPX working library with flux-weighted cross sections. The cross sections are generated upon convergence of the flux solution. This is an experimental feature. The syntax is given below.

```
ampx_out <T|F> <file_name> <num_cart_1> <num_cart_2> ...
```

The `<T|F>` flag is used to turn the feature on and off. Somewhat redundant, but potentially convenient.

4.6.19 CRITBORON Card

This card is used to set the parameters for the critical boron search.

```
critboron <T|F> [<target keff>] [<relaxation factor>] [!<comment>]
```

The `<T|F>` input is the indicator to turn on/off critical boron search. By default the critical boron search is off. The `target keff` is an option to set the target keff for the critical boron search and must be a positive floating-point value; the default is 1.0. The `relaxation factor` is an option input to set the relaxation factor (a floating-point value) with a valid range of 0.0 to 1.0, the default is 1.0.

4.6.20 XENON Card

This card is used to specify the equilibrium xenon calculation option. The options are:

- EQ enables equilibrium xenon calculation
- TR enables transient xenon calculation (same as none)
- NONE disables equilibrium xenon calculation.

The default is NONE.

If equilibrium xenon is turned on an optional relaxation factor can be defined for this part of the iteration. In some problems the solution may oscillate if relaxation is not used. Typically these problems must be 3-D and involve depletion and use reflective boundary conditions. The default value is 1.0. Valid ranges are on the interval [0.0,1.0].

The 3rd optional argument is to enable the equilibrium Sm calculation along with Xe. The default is F.

```
xenon <EQ|TR|NONE> [<relaxation factor>] [<T|F>] [!<comment>]
```

4.6.21 RESTART Card

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 causes a checkpoint file to be written after every outer iteration.

The default is `I`.

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

```
restart <T|F|W|R|RW|I> [<file>] [!<comment>]
```

4.6.22 AXIAL_TL Card

This card is for research purposes only and is not intended for use by the general user.

This card is used to set the parameter for the axial transverse leakage.

```
axial_tl <T|F> [<ISO|LIN|QUAD|DIFF|P1|DP0|EXP>] [<FLAT|LFLAT|FLUXXSTR>] [!<comment>]
```

`<T|F>` is used to set `tl` splitting on/off. `[<ISO|LIN|QUAD|DIFF|P1|DP0|EXP>]` is used to define the axial `tl` angular distribution. `ISO` means isotropic, `LIN` means linear, `QUAD` means quadratic, `DIFF` means diffusion, `P1` means simplified P1. Default is `ISO`. The options `DP0` and `EXP` are research options. `[<FLAT|LFLAT|FLUXXSTR>]` is used to define the axial `tl` spatial distribution. 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. Default is `FLAT`.

4.6.23 INTERNALTH Card

This card is used to control the execution of the simplified internal T/H capability.

```
internalth <T|F> [<ZR|SS>] [!<comment>]
```

`<T|F>` is used to set internal TH on/off. `<ZR|SS>` is used to set the thermal conductivity correlation for the clad. `ZR` is for Zircaloy and `SS` is for stainless steel. The defaults for this card are `F` and `ZR`.

4.6.24 CPM_INIT Card

This card is for research applications and is not intended for use by the general user.

This card is used to control the flux initialization procedure.

```
cpm_init <T|F> [!<comment>]
```

<T|F> is used to set CPM initializer on/off. The default value is F.

4.7 STATE Block

The STATE block is for specifying input parameters that define the core's state conditions. This includes options for things like the core power and rated power. However, the block may be used more than once. The description of the cards is given below.

4.7.1 RATED_POWER Card

This card is used to specify the whole core rated thermal power. The units for this card are megawatts (MW). The default value is 1 W. It is assumed that if the rated power is not specified for the problem, that Zero Power conditions will be used.

```
RATED_POWER <rated_core_power> [!<comment>]
```

4.7.2 CORE_POWER Card

This card is used to specify the percentage of full core power. The units for this card are % and the input values are real numbers. The default value is 100.0. For depletion, multiple powers can be input. The last power will be used for the remaining steps. For non-depletion cases, only a single value is needed.

```
CORE_POWER <percent_rated_power(:)> [!<comment>]
```

4.7.3 RATED_FLOW Card

This card is used to specify the whole core rated coolant flow rate. The units for this card are kilograms per second (kg/s). The default value is 1 kg/s.

```
RATED_FLOW <rated_core_flow> [!<comment>]
```

4.7.4 TINLET Card

This card is used to specify the core inlet temperature. The units for this card are degrees Celsius (C). The default value is 300 C.

```
TINLET <tinlet> [!<comment>]
```

4.7.5 BORON Card

This card is used to specify the core boron concentration. The units for this card are parts per million boron (ppmB). The default value is 0.0 ppmB.

```
BORON <boron> [!<comment>]
```

4.7.6 PRESSURE Card

This card is used to specify the core pressure. The units for this card are absolute pounds per square inch (psia). The default value is 0.0 psia.

```
PRESSURE <pressure> [!<comment>]
```

4.8 XSEC Block

The XSEC block is for specifying which files contain the cross section data to be used during the calculation. It is the second block that is read when processing the input file. This block must only appear once for a given input. It contains two cards ADDPATH and XSLIB which can be used an arbitrary number of times within the block. Details of the cards are described below.

4.8.1 XSLIB Card

The XSLIB card is used to specify the name of a cross section library file and its format. Its usage is shown below.

```
xslib <xslibformat> <xslibName> [!<comments>]
```

The <xslibformat> and <xslibName> are string inputs. If the cross section library name contains spaces then double quotation marks must be used to enclose the string. The name must not include the path, but must include the file extension. For information about the library formats see the programmer's documentation. Allowable values for the xslibformat are:

- USER (Default)
- AMPX (Requires SCALE license)
- AMPX_MASTER (Not yet fully functional)
- ORNL (Requires SCALE license)

- HELIOS (Requires Helios license)
- DEPLETION (Not yet fully functional)
- ORIGEN (Requires SCALE license)

4.8.2 ADDPATH Card

This card is used to append the search path for the cross section libraries specified with the XSLIB card of the XSEC block. This card is primarily for facilitating portability of inputs across different user environments and machines. There are 4 default directories that are searched prior to the paths specified in the ADDPATH cards. The additional paths are searched in the order in which they are defined and subdirectories are not searched. The default paths that are always searched are:

- the present working directory (e.g. the directory from which the command was launched)
- the directory containing the main input file
- the directory containing the default output file
- the directory containing the executable

The syntax for this card is shown below.

```
addpath <path> [!<comment>]
```

The <path> is a string input. An error will be produced when the file cannot be located. If the name contains spaces then double quotation marks must be used to enclose the string. The <path> should end with a SLASH character, but if it is not present, one will be added. The SLASH characters in the <path> string will also be replaced by the correct SLASH character of the file system. Relative paths may be used and they are relative to the PWD environment variable.

4.8.3 XSSHIELDER Card

This card is used to control how MPACT performs the resonance self-shielding. The inputs to the card are a logical flag, the shielding method, and the subgroup set.

```
xsshielder [<T|F> [<shieldingmethod> [<isubgrpset>]]] [!<comment>]
```

- <T|F> The logical input that toggles the resonance self-shielding on (T) or off (F). The default value is T.

- `<shieldingmethod>` The name of the shielding method to use. The available options are ESSM or SUBGROUP. The default value is Subgroup.
- `<isubgrpset>` The set number (integer) used if the shielding method is subgroup. The default value is 4.